Importing MusicXML files into Max/MSP

Technical Report: UL-CSIS-07-01

Keith Mullins & Margaret Cahill
Centre for Computational Musicology and Computer Music
Department of Computer Science and Information Systems
University of Limerick
Ireland
margaret.cahill@ul.ie

Abstract

A Max/MSP external object is created to allow musical scores to be imported into Max/MSP patches. This report details the programming of the object in Java and the parsing of the MusicXML file.

1. Introduction

The report describes the development of a Max/MSP object to facilitate the use of score files in this interactive graphical environment. The external object was written in the Java programming language. The object allows the user to import a musical score into a Max/MSP patch and provides access to particular musical elements of the score. The MusicXML format was for this project because of its widespread use and support across music applications. It has become a standard interchange format for musical scores and is also used by many music notation applications including Finale and Sibelius. Importing a MusicXML document, essentially a musical score, into an interactive music program like Max/MSP could have many benefits for the Max/MSP programmer. In addition to simply playing the score in MIDI or synthesised audio forms, elements of the score can be used as stimulus to trigger other events within a Max patch.

1.1 XML and MusicXML

XML (eXtensible Markup Language) is a markup language for documents containing structured information. It was created to structure, store and to send information and has become a W3C standard. Since XML data is stored in plain text

format, XML provides a software- and hardware-independent way of sharing data (*XML Tutorial*, 2006). There are now special XML-specific programming languages such as XQuery which is used to query XML documents in the same way that SQL is used to query database tables. There is even an XML version of HTML, the language used to display web pages, called XHTML which is designed to replace HTML. The W3C believe that in the coming years "XML will be the most common tool for all data manipulation and data transmission" (*XML Tutorial*, 2006).

MusicXML is a universal translator for common Western musical notation from the 17th century onwards. It is designed as an interchange format for notation, analysis, retrieval, and performance applications (Good, 2001). Since being created by Michael Good at Recordare, MusicXML has been supported by a number of commercial applications and is now considered an interchange standard for musical scores.

In the past sharing music between music applications was problematic. Many applications shared musical scores in proprietary formats or simply used MIDI. This meant that, if more than one piece of music software was used for composition or notation, the user could not easily share this music among applications, even of the same type. This was a huge inconvenience. In the 1980's electronic musical instrument manufacturers faced a similar problem. Synthesisers, from two different manufacturers for example, had no way of being able to work or play together. The fix came with the invention of the Musical Instrument Digital Interface (MIDI) format. Although MIDI works well for performance applications, it has many shortcomings when it comes to music notation applications. MIDI does not represent stem direction, beams, repeats and many other aspects of notation, nor does it know the difference between a C-sharp and a D-flat. Research Centres and Academic Institutes used various text-based formats such as EsAc, DARMS, Kern, ALMA, etc. to store musical score data but none of these formats was commercially supported and the variety of formats used made sharing and interchange difficult (Selfridge-Field, 1997). There have been other attempts in the past to produce a new interchange format, such as NIFF and SMDL, but for various reasons these formats were not successful. MusicXML has become an accepted widely used format for sharing and interchange because it is supported by the most commonly used notation applications. The table below shows a range of commercial musical applications that support MusicXML.

Application	Description	Read MusicXML	Writes MusicXML
Finale	music notation and performance software	X	X
Sibelius	music notation and performance software	X	X
Notion	music composition and performance software	X	X
Harmony Assistant	computer-assisted composition and editing program	X	X
Forte	notation, sequencing and recording software	X	X
QuickScore Elite	integrated 48-track scoring and sequencing program	X	X
Guitar Pro	multi-track tablature editor for guitar, banjo and bass	X	X
SmartScore	music scanning and notation software		X
ScoreMaker	music scanning and notation software		X

Table 1: A sample of musical applications that support the MusicXML format.

With MusicXML being accepted by all of these applications it is now possible for a user to create a score in Finale, save it as a MusicXML and send it to a friend. The friend can then use Sibelius to play the score created in Finale and edit it if they wish. Another example of this interoperability can be seen if a user of SmartScore scanned in some sheet music and saved it as MusicXML and imported the MusicXML into Finale. The user could then edit and play the score they have just scanned in which would have been impossible without MusicXML.

1.2 MusicXML structure

XML represents data in tags that are nested in a hierarchical structure as follows:

```
<root>
      <parent>
         <child_one/>
         <chile_two/>
      </parent>
</root>
```

Musical scores are two-dimensional with musical events occurring over time, whereas XML files are one-dimensional. To reconcile this, MusicXML represents scores either

part-wise (measures within parts) or time-wise (parts within measures). Musical scores can be used as a record of, a guide to, or a means to perform a piece of music. They are not simply just a collection of notes and melodies; they are much more than that. The musical score contains information such as tempo, dynamics, phrasing, and much more. There is a large amount of information contained within a musical score and MusicXML represents this information in XML tags.

Each tag in a MusicXML file represents some aspect of the score. For instance the tag `<note>` contains all the information about a note - such as pitch, duration, dynamics, slurs, articulations, etc.

1.2.1 Extracting specific tags only

One very useful aspect of using XML is that when parsing a MusicXML file it is possible to only extract specific tags depending on what elements of the score are required. It is very easy, for example, to find all the note pitches and durations of a score and ignore all other tags. Those tags representing musical elements of the score that are not needed will simply be skipped, saving processing time. Considering that there are a total of four-hundred and ninety-seven unique tags in MusicXML this ability to skip tags is very useful.

1.3 Why Max/MSP?

Composers who write computer music have a number of tools at their disposal that allow them to generate and manipulate sound on the computer quickly and easily. One of the easiest to use and most powerful tools of this nature is the software program Max/MSP. Max/MSP is a graphical programming environment for music, audio and multimedia. It offers the non-programmer the chance of creating complex systems that can be used to schedule real-time event- and signal-driven processes with very little difficulty. Importing a musical score into Max/MSP offers the Max/MSP programmer a new catalogue of opportunities for interaction, composition and performance.

Once the score is available in Max/MSP it could be used in a variety of ways. The score could be played in full or partially using MIDI or synthesised audio. Elements of the score could be used to control other events in the patch such as triggering loops, audio processing, controlling image and video events etc. This score-driven approach offers many new and exciting possibilities for interactive performance and music-making with Max/MSP.

1.4 Other Max/MSP Applications that use MusicXML

Developed by Dr. Keith Hamel at the University of British Columbia, NoteAbility Pro is a professional music notation package for the Macintosh OS-X (NoteAbility Pro, 15 Jul 2006). This application can send and receive messages to and from Max/MSP. Dr. Hamel claims that NoteAbility can be used as a real-time sequencer to control Max/MSP, but in reality setting this up is quite laborious. It works by sending messages over a network from NoteAbility Pro to an object in Max/MSP. The user must physically draw a text-box in the notation editor at the exact point in the score where a message is to be sent to Max/MSP. The message must be written in this text-box. Using this method one could import a musical score into Max/MSP but it would mean that for every note, accent, dynamic marking and all other score features in a particular score, the user would have to create a text box and type the relevant message that should be sent to Max/MSP. This would take a huge amount a time and would mean that for every score to be imported into Max/MSP the user would have to repeat the whole process of making a text box and typing the message all over again.

Another technology that combines a score notation editor with Max/MSP is the Java Music Specification Language (JMSL) (AlgoMusic - Using JMSL with Max/MSP). JMSL is a Java-based development tool for experiments in algorithmic composition, live performance, and intelligent instrument design. JMSL can send and receive musical data to and from Max/MSP. It comes with a music notation editor which allows the user to create a score, and play it by using an instrument they have made in Max/MSP or by using any of the built in MSP instruments. However, it only allows the user to play the score. It does not let the user access the finer details within a score such as pitch, duration or the dynamic features of the score. These can be notated, and auditioned, but that is all. Therefore, there is no means by which the user can use a musical score as a controller, or as a means of interaction within their own Max/MSP patch through JMSL. With JMSL the user can only create a score using the JMSL notation editor and unfortunately they are unable to import their own score or MusicXML document into the editor.

2. Implementation

Max/MSP external objects are additional to the standard set that forms the software package and are created by programmers using the Max/MSP SDK. Programmers can add extra functionality to the program using external objects. Up until recently Max/MSP external objects could only be written in the C/C++ Programming language but since the 4.5 release these external objects can now be written in Java also. When deciding what programming language would be used for this project many compelling arguments were found in the "Writing Max Externals in Java" manual (LaFata, 2006).

Java externals are cross-platform so you don't need to develop and maintain a single binary distribution for both the Windows and OSX versions of Max/MSP. Java handles all memory management, which means that the programmer does not need to worry about memory allocation issues present in C/C++ or the program crashing as a result of dereferencing bogus pointers. Java has a large class library and there is a wealth of code available on the Internet to quickly provide functionality that would have taken much longer to program in C/C++.

Accessing the functionality of Java externals in a Max/MSP patch is slightly different to accessing the functionality of an external written in C/C++. C/C++ externals are loaded directly into the Max/MSP application at runtime, but Java externals are loaded on the Max/MSP environment and thereafter communicate with the environment though a proxy external called **mxj**. MXJ allows calls from Max/MSP to be dispatched to the Java code through Java Virtual Machine and vice versa.

The external object created to import the MusicXML file into Max/MSP is called ShowXML and is written in Java. The name of the object needs to be passed as an argument to the mxj object in order to access the functionality of ShowXML (see Figure 1.)

Figure 1: The ShowXML object in a Max/MSP patch.

If the external object ShowXML had been written in C/C++ the user would simply enter the name of the object into an empty object box. In order for Max/MSP to read

the Java external it must first be compiled and the relevant Java class stored in the following folder:

- **Windows:** C:\Program Files\Common Files\Cycling ‘74\java\classes
- **OS X:** /Library/Application Support/Cycling ‘74/java/classes/

2.1.1 MaxObject Class

For the mxj-bridge to instantiate an instance of a Java class, the class needs to be a subclass of `com.cycling74.max.MaxObject`. By subclassing `MaxObject` the programmer has access to a whole host of functions that allow the programmer to communicate with the Max/MSP application. The relevant code is shown in Code Excerpt 1.

```
import com.cycling74.max.*;

public class ShowXML extends MaxObject {
     /* insert code here */
}
```

Code Excerpt 1: Making the class for the new object a subclass of MaxObject.

When the user loads the ShowXML Java external into a patch the constructor for that external is called. Within the constructor the programmer can specify the number of arguments for the object. Example code for entering a filename is shown in Code Excerpt 2.

```
import com.cycling74.max.*;

public class ShowXML extends MaxObject {

     public ShowXML() {
          bail( “Error: Argument needed” );
     }

     public ShowXML( String fName ) {
          /* insert code here */
     }
}
```

Code Excerpt 2: The ShowXML constructor is designed to take a filename as an argument

The `bail()` method of `MaxObject` is, in this case, used to prevent instantiation of the object and print an error message to the Max console informing the user why the instantiation failed. In order for the object to be instantiated correctly the user must enter an argument such as that shown in Figure 2.

Figure 2: The ShowXML extern correctly instantiated in the Max patch.

2.1.2 Inlets and Outlets

This is probably the most important aspect of the constructor method. The programmer must let the mxj-bridge know how many inlets and how many outlets it should create for the class. To do this we use the `declareInlets` and `declareOutlets` methods of `MaxObject`. These methods both take an integer array as arguments that describes the types of inlets or outlets to be created in left to right order. The following code (Code Excerpt 3) describes how to create an external object that has one inlet (that can take Lists, Strings, Integers and Floats) and one outlet (that can only send Integer values through it):

```
public ShowXML( String fName ) {
      declareInlets( new int[]{ DataTypes.ALL });
      declareOutlets( new int[]{ DataTypes.INT });
}
```

Code Excerpt 3: The ShowXML constructor with methods for creating inlets and outlets.

2.1.3 Handling interaction between objects

There are many different types of message that can be between objects in Max/MSP. It is very important that any Java external that this project creates is aware of these messages and has is programmed to handle them. The following is an overview of the types of messages that can be received by Max/MSP objects.

- **Bang Message:** To get a Java external to respond to a bang message the programmer simply overrides the `bang()` method of `MaxObject` like so:

```
public void bang() {
     post( “Received Bang” );
}
```

Code Excerpt 4: Overriding the bang() method to create a chosen response to a bang.

Figure 3: A Bang message in Max/MSP being sent to ShowXML

- **Int and Float Messages:** Handling these messages is very similar to handling bang messages except that the programmer will also be receiving a value in addition to the message. When the user connects a number box to an inlet of the Java external and changes its value the object will receive an int or float message from an integer or float box respectively. To handle these primitive numerical messages the programmer needs to override the `inlet()` methods of `MaxObject`. There is one method for an int message and one for a float message. Example:

```
public void inlet( int i ) {
     post( “Received ” + i + “ in inlet ” + getInlet() );
}
public void inlet( float f ) {
     post( “Received ” + f + “ in inlet ” + getInlet() );
}
```

Code Excerpt 5: Overriding the inlet() method to handle integer and float input values.

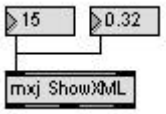

Figure 4: An Int and Float message in Max/MSP being sent to ShowXML

The `getInlet()` method is used to determine which inlet an incoming message has been passed though. If only a float inlet method exists in the Java external (`public void inlet( float f )`), mxj will coerce int messages to float messages automatically so that if the user attaches an int box to the object, mxj will convert the int value to a float of the same value. The same holds true in the other direction as well.

- **Arbitrary Messages:** These are the most important messages for this project. To expose a message that a Java external will respond to in the Max environment the programmer can declare a method of the same name as the message that needs to be handled with the public visibility modifier and a return type of void. So if the user types "start" into a message box and connects that message box to an inlet of the Java external the programmer can write a method called "start" to respond every time this message is sent to the object. The method would be as follows:

```
public void start() {
     /* insert code here */
}
```

Code Excerpt 6: Declaring a new start() method to create a user-defined input message.

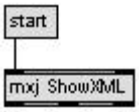

Figure 5: An arbitrary message (start) in Max/MSP being sent to ShowXML

The user may also pass arguments along with these arbitrary messages. The following demonstrates how to handle these arguments:

```
public void start( Atom[] args ) {
     if( args.isInt() ) {
          /* do something */
     } else if( args.isFloat() ) {
          /* do something else */
     }
}
```

Code Excerpt 7: Passing a number of arguments to a user-defined method.

Figure 6: An arbitrary message (start) with arguments in Max/MSP being sent to ShowXML

Atom is used to accept lists (arrays) of all kinds of datatypes. Therefore, a Java external could accept a list that includes a mixture of ints, Strings, and floats.

- **Anything Message:** If the programmer defines a method with the signature below this means that whenever the Java external receives a message which cannot be resolved to any other message, the anything method will be called with the `msg` parameter set to the name of the unresolved message and the arguments passed in as the `args` Atom array. The anything method allows the programmer to implement a catchall message handler, or create a message dispatching framework.

```
public void anything( String msg, Atom[] args ) {
     /* insert code here */
}
```

Code Excerpt 8: Using the anything() method to catch unresolved messages.

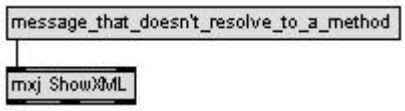

Figure 7: An anything message in Max/MSP being sent to ShowXML

2.1.4 Sending messages to other objects

There are numerous methods that can be used to output messages to the Max environment. Here follows a list of these methods:

- **Bang:** The `outletBang` method of `MaxObject` sends a bang out whatever outlet that it receives as an argument. In this example a 0 represents the left most outlet.

```
public void bang() {
     outletBang( 0 );
}
```

Code Excerpt 9: Sending a bang out the left inlet.

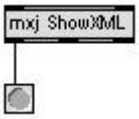

Figure 8: A bang message in Max/MSP being sent from ShowXML

- **Outlet:** The outlet method of MaxObject is used to send values out the outlets of an object. This method takes two arguments. The first of these is an int representing the outlet that you wish to send the message through, and the second argument is the actual message itself. The following example sends the value 5 out the right outlet of an object.

```
public void bang() {
     outlet( 2, 5 );
}
```

Code Excerpt 10: Sending a value out the right outlet of an object.

Figure 9: An outlet message in Max/MSP being sent from ShowXML

The following list, of variations of the `outlet` method, represents some of the datatypes that can be sent through the outlet:

```
public void outlet( int index, boolean b );
public void outlet( int index, char c );
public void outlet( int index, short s );
public void outlet( int index, String msg );
public void outlet( int index, int[] i_array );
public void outlet( int index, String[] s_array );
/* and so on */
```

Code Excerpt 11: The datatypes that can be sent out the outlets of an object.

2.2 Parsing the Document

The ShowXML object is capable of extracting any number of tags from a MusicXML file. The number of tags, and the selection of tags to be extracted, is decided upon by the user. It has been shown earlier in this document how to create an external object in Java but before an object is constructed some decisions need to be made. The first is to decide what parser to use.

2.2.1 DOM, SAX and Xerces

To parse an XML document means to extract the data that is located between the `<opening>` and `</closing>` tags of the XML file. The following example shows an XML file containing a person's address and also the information that is extracted from parsing that file:

```
<address>
   <lineone>Apt 23</lineone>
   <linetwo>The Street</linetwo>
   <city>New York</city>
   <zip>12345</zip>
   <country>America</country>
</address>

Parsed Data:
Apt 23
The Street
New York
12345
America
```

Code Excerpt 12: An example of an XML file and data parsed from it.

DOM and SAX are two API's that allow XML work to be done within Java. The **D**ocument **O**bject **M**odel is used to parse an XML file and does so by storing the XML file as a tree-like structure in the computer's memory. This enables you to move forwards and backwards through the XML file from within Java. The **S**imple **A**PI for **X**ML works differently to DOM. It too is used to parse an XML file, but rather than storing the whole file in memory, it simple skims through the file once allowing you to abstract the data you need quickly and easily. DOM and SAX are not parsers themselves; they are simply API's that allow you to work with XML files in Java

(DOM can be used with other programming languages). The actual parser that will be used in this project is called Xerces, which can be downloaded freely from http://xml.apache.org.

It was decided that SAX would be used in this instance for two main reasons. The first was that SAX is much faster than DOM at parsing XML documents. The second reason for using SAX over DOM is that the whole MusicXML file may not be needed in this project and using DOM means that the whole document gets stored in memory. SAX allows tags to be skipped thus using less memory and processing power and lets the user extract only the data needed for its intended purpose.

2.2.2 Parsing the SAX way

When parsing with SAX there are at least three Java methods that must be overridden to make full use of the SAX API. The three methods are:

```
public void startElement( String namespaceURI,
                          String localName,
                          String qName,
                          Attributes atts );

public void characters ( char[] ch, int start, int len );

public void endElement( String namespaceURI,
                        String localName,
                        String qName );
```

Code Excerpt 13: These three methods are needed when using SAX to arse an XML document.

Uisng SAX to parse the document in Code Excerpt 1, the `startElement` is `<lineone>`, `characters` is `“Apt 23”` and `endElement` is `</lineone>`. In other words to SAX the XML file takes the following structure:

```
<startElement>
   <startElement>characters</endElement>
   <startElement>characters</endElement>
   <startElement>characters</endElement>
   <startElement>characters</endElement>
   <startElement>characters</endElement>
</endElement>
```

Code Excerpt 14: An example of a SAX-parsed document.

The other two methods that can be overridden are the startDocument() and endDocument() methods but are not as critical as those listed above. These two methods are called when the root element has been opened (`<root>`) and when the root element has been closed (`</root>`) respectively. Incidentally the root element in a MusicXML document is either `<scort-partwise>` or `<score-timewise>` and the difference between these has been explained earlier in this document. The following is a short example of a Java program that uses SAX to access the contents of the <lineone> element shown above and display it to the screen.

```
public boolean isLineOne;

public void startElement( String namespaceURI,
                          String localName,
                          String qName,
                          Attributes atts ) {

     if( localName.equals( “lineone” ) {
          isLineOne = true;
     }
} /* end of startElement */

public void characters ( char[] ch, int start, int len ) {

     if( isLineone == true ) {
          String str = new String( ch, start, len );
          System.out.println( “Line One: ”  + str );
     }
} /* end of characters */


public void endElement( String namespaceURI,
                        String localName,
                        String qName ) {

     if( localName.equals( “lineone” ) {
          isLineOne = false;
     }
} /* end of endElement */
```

Code Excerpt 16: Parsing the document shown in Code Excerpt 12.

The `startElement()` gets called first, followed by the `characters()`method and then `endElement()`. If `startElement()` finds the `<lineone>` tag it sets `isLineOne` to `true` so that when `characters()` is called next and it sees that `isLineOne` is `true` it knows to extract the incoming data because it is needed. `endElement()` then sets `isLineOne` back to `false` and the whole process starts again.

2.2.3 ShowXML constructor and methods

Within the constructor the number of inlets and outlets are declared. ShowXML has a single inlet that will accept any type of message. The number of outlets will be determined by the user at run-time. The user will have to pass a number to the constructor indicating the number of outlets the object should have.

Figure 10: The ShowXML object with six outlets.

As can be seen from the above image there are six outlets present. There is an additional default information outlet that reports the values of attributes belonging to the external object. This can be turned off by calling the method `createInfoOutlet( false)` within the constructor.

There are five main methods in ShowXML: `open()`, `parse()`, `go()`, `stop()` and `time()`. The following illustration is a typical patch using ShowXML that will extract and display all the pitch and their respective duration values from a MusicXML document.

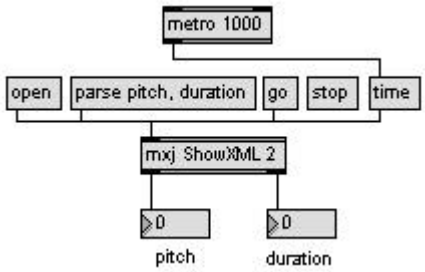

Figure 11: A patch that outputs each pitch and duration pair of a MusicXML document.

The `open()` method will open a file dialog box to allow the user to select a MusicXML file from their hard drive. `open()` calls the `openDialog()` method of

`MaxSystem` to do this. `openDialog()` returns the file name and address as a String and this is stored in the ShowXML attribute `fName`.

The `parse()` method will call `parseThis()` from the `ShowXMLParse` class (described later) and passes `fName` as an argument. `parseThis()` then begins to parse the MusicXML document using SAX. It is important to note that `parse()` takes in two arguments above. The arguments it takes in are the names of the tags it should parse. In this example the values for pitch and duration will be the ones that are extracted and no others.

Once the MusicXML document has been parsed, the `go()` method will begin to send the contents of that document out through the left and right outlets. The `stop()` method simply stops this sending process.

Every time the `time()` method is called it sends the parsed information through the outlets at regular/irregular intervals based on how often it is called (in the above example it is called every one second by the metro object). It will only send the information one note/chord at a time – from here on this information is called the information-set. For example, if a MusicXML file contained only four notes (C3, D3, A3, F3) and the duration values (4, 8, 4, 16), and it was parsed using the above patch the following information would be sent out the relevant outlets at intervals of 1 second.

- Pitch: C3 Duration: 4 – information-set [C3, 4]
- Pitch: D3 Duration: 8 – information-set [D3, 8]
- Pitch: A3 Duration: 4 – information-set [A3, 4]
- Pitch: F3 Duration: 16 – information-set [F3, 16]

Pitch is the left-most outlet in the patch above and Duration is the right-most outlet in the patch above.

There is also a `reset()` method that resets a counter within the external object that controls which information-set should be sent through the outlet(s) next. If the `reset()` method was called after the information-set [A3. 4] had been sent through the outlets the next information set that would be sent would be [C3, 4].

2.3 ShowXMLParse class

Unfortunately it is not possible to do everything that this project requires ShowXML to do with a single object. In Java, a class can only extend one other class. In order to parse a file the class that parses the XML document must extend the `DefaultHandler` class. Since ShowXML already extends the `MaxObject` class a new class must be written for parsing documents.

```
import org.xml.sax.helpers.*;

public class ShowXMLParse extends DefaultHandler {
     /* insert code here */
}
```

Code Excerpt 16: Creating the ShowXMLParse class as a subclass of DefaultHandler.

Having ShowXMLParse extend `DefaultHandler` means that ShowXMLParse can override the methods mentioned in 2.2.2 above.

2.3.1 Parsing the MusicXML document

Parsing a MusicXML document is no different to parsing a regular XML document. It is simply a matter of finding the element you want and extracting the data between the opening and closing tags of that element. At this point in the project a decision needed to be made as to whether the ShowXML external object would parse a MusicXML document and send the information-sets through the outlets in real-time or not. One of the main arguments against parsing in real-time is that parsing would take some matter of time to begin, and to find the first tag to be parsed. This time may only be a matter of milliseconds but if the Max patch that the object is working in requires the parsing to be in sync with other objects in the patch then this could prove to be quite troublesome. If the object does not send the information-sets perfectly in time with the rest of the patch then it is of no use to the Max programmer. Another weakness in real-time parsing is that once the MusicXML document has been parsed the information is gone and ShowXML is in effect disabled within the patch. The document would then have to be re-parsed, if the Max patch so required, in order to re-enable ShowXML. For these reasons it was decided to store the information garnered from parsing the MusicXML document.

2.3.2 Storing the MusicXML data

It was decided to create another object, a Java Bean, called `ShowXMLDataStore` to store the MusicXML data. Each information-set would be stored in a `ShowXMLDataStore` object. It was then decided to create an `ArrayList` to store this collection of `ShowXMLDataStore` objects. The following is an illustration of how this works in the ShowXML object:

MusicXML Sample:

```
<note>
  <pitch>
    <step>G</step>
    <octave>4</octave>
  </pitch>
  <duration>2</duration>
  <notations>
     <dynamics>
        <p/>
     </dynamics>
  </notations>
</note>
```

Pseudo code to store MusicXML Sample:

```
List list = new ArrayList();
ShowXMLDataStore s = new ShowXMLDataStore();

s.setStep(“G”);
s.setOctave(4);
s.setDuration(2);
s.setDynamics(“p”);
list.add(s);
```

Code Excerpt 17: A MusicXML sample and pseudo code to store the information.

Each MusicXML element must have a “set” and “get” method in `ShowXMLDataStore` as the above example shows – the fact that `ShowXMLDataStore` only has these “get” and “set” methods is what makes it a Java Bean. When the document is finished parsing `list` will contain all of the information-sets that are to be sent through the ShowXML outlets.

2.3.3 Retrieving the MusicXML data

The retrieval of the MusicXML data is done inside the `time()` method of `ShowXML`. Firstly, `ShowXML` must get a handle on `list` that has been populated with a number of `ShowXMLDataStore` objects inside the `ShowXMLParse` class. `retrievalList` belongs to ShowXML and it contains all of the information-sets that were extracted from the MusicXML document (see Code Excerpt 18).

```
List retrievalList = ShowXMLParse.list;
```

Code Excerpt 18: retrievalList stores the information sets

The following is a sample Max patch that uses ShowXML to extract the information that was stored. The Java code to store this information is shown is Code Excerpt 19.

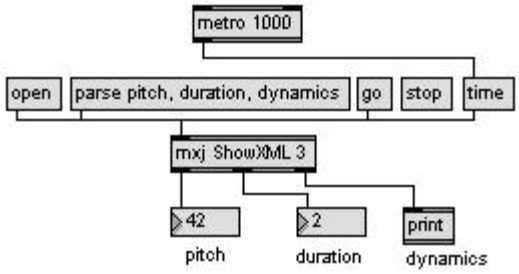

Figure 12: Max patch to retrieve pitch, duration and dynamics from MusicXML document.

```
List retrievalList = ShowXMLParse.list;
for( int i=0; i< retrievalList.size(); i++ ) {
   ShowXMLDataStore s = new ShowXMLDataStore();
   s = (ShowXMLDataStore) retrievalList.get(i);
   outlet( 0, s.getPitch() );
      outlet( 1, s.getDuration() );
      outlet( 2, new String( "" + s.getDynamics() );
}
```

Code Excerpt 19: Java code to store parsed information from the MusicXML document.

Pitch is determined by using a combination of the `<step>`, `<octave>` and `<alter>` tags in a MusicXML document. Also, it is important to note that G3 is equal to 42 in MIDI because MusicXML classes Middle-C (usually 60 in MIDI) as C4, rather than C3. Therefore, G3 is in reality actually G2.

2.4 Using Threads to control time

The timing aspect of this project is crucial as ShowXML must send the information-sets at particular intervals determined by an outside source (in the above example it is determined by the metro object). There are a number of ways of controlling when ShowXML sends the information-sets. One way to accomplish this is to use the Max/MSP scheduler by programming with the MaxClock API. This will allow ShowXML to send messages at regular intervals to other objects in the Max patch. The problem with this is that using the Max/MSP scheduler for computationally intensive tasks can cause Max/MSP to freeze. The Max patch could also become slower and under-perform as a result of the scheduler being overrun. It is possible to leverage the built-in threading capabilities of the Java programming language to offload some of the overhead of computationally intensive event processing from the Max/MSP scheduler to the Java Virtual Machine. Using Java Threads provides a solution to the problem of controlling time-intensive tasks without overworking the Max/MSP scheduler into near collapse.

Creating threads in Java is a very easy process. The following code snippet shows how the previous example could have been handled in a Thread in such a way that it follows the instructions of when to start/stop from the metro object.

```
List retrievalList = ShowXMLParse.list;
int whereInList = 0;
boolean isSuspended = false;
Thread t = new Thread() {
   public void run() {
         if( isSuspended ) t.resume();
      ShowXMLDataStore s = new ShowXMLDataStore();
      s = (ShowXMLDataStore) retrievalList.get(whereInList);
      outletHigh( 0, s.getPitch() );
         outletHigh( 1, s.getDuration() );
         outletHigh( 2, new String( “” + s.getDynamics() );
         whereInList++;
         isSuspended = true;
         t.suspend();
    }
};
```

Code Excerpt 20: Using threads to store the parsed information.

This code snippet belongs in the `time()` method of ShowXML which is called every second by the metro object. The code that is to be executed in the Thread must be placed inside the `run()` method and in turn the `run()` method must be placed inside the `Thread()` as shown above. The method `suspend()` will pause the Thread running, in other words it will stop the information-sets from being sent through the outlets until the `time()` method gets called again in which case the `resume()` method restarts the process. The attribute `whereInList` is used to keep track of what information-set should be passed through the outlet next. When using Java threads for time-intensive tasks it is important to use the method `outletHigh()` instead of `outlet()`. In Max/MSP there are two schedulers – a high-priority scheduler and a low-priority scheduler. When the `outlet()` method is called from within a Java Thread it is automatically sent to the low-priority scheduler and may not get processed straight away whereas `outletHigh()` is processed straight away because it is sent to the high-priority scheduler of Max/MSP. Thus, because this project uses Java Threads to output data that is time-critical it is imperative that `outletHigh()` is used.

The external object ShowXML can read in a MusicXML document, parse it and send the relevant information (decided upon by the Max programmer) through as many outlets as are needed.

3. Conclusion

The aim of this project has been to import a musical score, in MusicXML format, into the graphic programming environment of Max/MSP and to make the elements of that score available to the user. The implication of this is that by allowing the user access to elements of a musical score such as pitch, duration, dynamic markings, articulations, etc. presents them with the opportunity of using the musical score as a controlling device capable of triggering other events with the Max/MSP patch. The external object created in this project provides the user with a way of importing a musical score, in the standard interchange format for music notation, into Max/MSP. It also allows the user to obtain any particular elements of a score they wish. In this way the user can do much more than simply play the musical score, they can use its unique components to create interesting applications and pieces of music. A certain number of elements were chosen to showcase the potential abilities of

ShowXML and in the future it is hoped that the rest of the elements will be available to the Max/MSP programmer. Information on the elements that are available can be seen in the comments in the .java files supplied in Appendix A.

As this project involved working with XML, future work might be spent investigating whether an XML Schema would be better for validating MusicXML documents rather than a DTD. XML Schemas had not been invented or standardised when MusicXML was first created but now XML Schemas are a W3C standard and are seen as a replacement for DTDs. Had an XML Schema been used in this project it would have saved a lot of cumbersome programming work because a technology like the Java API for XML Binding (JAXB) could have been used instead of parsing the XML document using SAX. The useful thing about using JAXB is that instead of parsing an XML document using DOM or SAX and extracting the data that is needed by coding in the actual element names (explained in 2.3.2 above) the XML document is simply bound to a Java object and the programmer can obtain the relevant information through this object. What's more is the programmer does not have to create this object - it is created by compiling the XML Schema and so saves programming time.

5.References

AlgoMusic (2006) Using JMSL with Max/MSP [online]
Available: http://www.algomusic.com/jmsl/ [accessed 13 Jun 2006]

Byrd, D., Crawford, T. 'Problems of music information retrieval in the real world', Information Processing and Management, 38.

Good, M. "MusicXML: An Internet-Friendly Format for Sheet Music". In XML 2001 Conference Proceedings, Orlando, Florida, December 9-14, 2001.

Castan, G., Good, M. and Roland, P. "Extensible Markup Language (XML) for Music Applications: An Introduction." In The Virtual Score: Representation, Retrieval, Restoration, W. B. Hewlett and E. Selfridge-Field, eds., MIT Press, Cambridge, MA, 2001, pp. 95-102. Computing in Musicology 12.

Good, M. "Representing Music using XML". In Proceedings of ISMIR 2000, October 23-25, Plymouth, Massachusetts, 201.

LaFata, T. 'Writing Max Externals in Java' Version 0.3. [online], Available: **http://pcm.peabody.jhu.edu/~wright/stdmp/docs/WritingMaxExternalsInJava.pdf** [accessed June 2006]

L.A. Tech (2006) SAX Parsing [online],
Available: http:// www2.latech.edu/~box/ds/intro_xml.ppt [accessed 8 Jul 2006]

MusicXML (2006) MusicXML 1.1 Tutorial [online],
Available: http://www.musicxml.org/xml/musicxml-tutorial.pdf [accessed 4 Jun 2006]

Music XML (2006) MusicXML Alphabetical Index [online],
Available: http://www.musicxml.org/xml/musicxml-index.html [accessed 4 Aug 2006]

MusicXML (2006) MusicXML 1.1 Note DTD Module [online],
Available: http://www.musicxml.org/dtds/note.html [accessed 15 Jun 2006]

NoteAbility Pro (2006) NoteAbility Pro [online],
Available: http://debussy.music.ubc.ca/NoteAbility/index.html [accessed 15 Jul 2006]

O'Reilly Network, On Java (2006) Simple XML Parsing with SAX and DOM [online],
Available: http://www.onjava.com/pub/a/onjava/2002/06/26/xml.html
[accessed 6 Jun 2006]

Pfisterer, Matthias. SimpleAudioPlayer.java from Java Sound Resources [online],
Available: **http://www.jsresources.org/examples/SimpleAudioPlayer.java.html**
[accessed 10 Aug 1006]

Selfridge-Field, Eleanor. 'Beyond Codes: Issues in Musical Representation.' Beyond MIDI: The Handbook of Musical Codes. Ed. Eleanor Selfridge-Field. Cambridge, MA: MIT Press, 1997.

W3Schools (2006) XML Tutorial [online],
Available: http://www.w3schools.com/xml/default.asp [accessed 2 Aug 2006]

Appendix

ShowXML.java

```
import com.cycling74.max.*;

public class ShowXML extends MaxObject {

   public String fName;
   public static boolean isParsing = false;
   public static boolean running = false;
   private static int ctr = 0;
   private static Atom[] outlet;
   MaxSystem sys;
   Thread t;
   ShowXMLDataStore note;

   public static int whereInXML = 0;
   public static int divisions = 0;
   public static int beats = 0;
   public static int tempo = 500;

   public ShowXML() {
      bail( "Error: You must enter an integer value as an " +
            "argument that represents the number of outlets that
ShowXML will have" );
   }

   public ShowXML( int i ) {
      declareIO( 1, i );
      createInfoOutlet( false );
      sys = new MaxSystem();
      ctr = 0;
   }

   public void parseMe() {
      final String file = this.fName;
      post( "PARSING >> " + this.fName );
      isParsing = true;
      ShowXMLParse.play = this;
      Thread t = new Thread() {
         public void run() {
```

```
            ShowXMLParse.ParseThis( file );
        }
    };
    t.start();
}

public void anything( String s, Atom[] args ) {
    if( s.startsWith( "parse" ) ) {
        ShowXMLParse.tags = args;
        outlet = args;
        parseMe();
    }
}

public void inlet( int speed ) {
    tempo = speed;
}

public void reset() {
    ctr = 0;
}

public void time( int val ) {
    note = new ShowXMLDataStore();

    if( ctr == 0 ) {
        outlet( 0, new Atom[] { Atom.newAtom(divisions),
               Atom.newAtom(beats),
               Atom.newAtom(ShowXMLParse.dataList.size()) } );
    }
    if( ctr < ShowXMLParse.dataList.size() ) {
        note = ( ShowXMLDataStore ) ShowXMLParse.dataList.get(ctr);

        for( int i=0; i<outlet.length; i++ ) {
            if( outlet[i].toString().equals( "pitch" ) ) outlet( i,
                 note.getPitch() );
            if( outlet[i].toString().equals( "articulations" ) )
                 outlet( i, note.getArticulation() );
            if( outlet[i].toString().equals( "duration" ) ) outlet(
                 i, note.getDur() );
```

```
            if( outlet[i].toString().equals( "dynamics" ) ) outlet(
                 i, note.getDynamic() );
            if( outlet[i].toString().equals( "pitchInText" ) )
                 outlet( i, note.getPitchText() );
        }
        ctr++;
    }
}

public void time() throws InterruptedException {
    t = new Thread() {
        public void run() {
            note = new ShowXMLDataStore();
            note = ( ShowXMLDataStore )
            ShowXMLParse.dataList.get(ctr);

            for( int i=0; i<outlet.length; i++ ) {
            if( outlet[i].toString().equals( "pitch" ) ) outlet( i,
                  note.getPitch() );
            if( outlet[i].toString().equals( "articulations" ) )
                  outlet( i, note.getArticulation() );
            if( outlet[i].toString().equals( "duration" ) ) outlet(
                  i, note.getDur() );
            if( outlet[i].toString().equals( "dynamics" ) ) outlet(
                  i, note.getDynamic() );
            if( outlet[i].toString().equals( "pitchInText" ) )
                  outlet( i, note.getPitchText() );
            }

            ctr++;
            try {
                t.suspend();
            } catch( Exception e ) {
                MaxObject.post( "Error: " + e.getMessage() );
            }
        }
    };
    if( ctr != ShowXMLParse.dataList.size() ) t.start();
    else t.stop();
}
```

```
public void go() {
    running = true;
    ctr = 0;
}

public void go( int value ) {
    if( value == -1 ) {
        running = true;
        ctr = 0;
        t = new Thread() {
            public void run() {
                for( int j=0; j<ShowXMLParse.dataList.size(); j++ )
                {
                    note = new ShowXMLDataStore();
                    note = ( ShowXMLDataStore )
                    ShowXMLParse.dataList.get(ctr);
                    int lengthToSleep = 0;

                for( int i=0; i<outlet.length; i++ ) {
                   if( outlet[i].toString().equals( "pitch" ) )
                        outlet( i, note.getPitch() );
                   if( outlet[i].toString().equals( "articulations"
                        ) ) outlet( i, note.getArticulation() );
                   if( outlet[i].toString().equals( "duration" ) )
                        outlet( i, note.getDur() );
                   if( outlet[i].toString().equals( "dynamics" ) )
                        outlet( i, note.getDynamic() );
                   if( outlet[i].toString().equals( "pitchInText" )
                        ) outlet( i, note.getPitchText() );
                }
                ctr++;

                lengthToSleep = ( tempo / divisions ) *
                note.getDur();

                post( ":" + lengthToSleep );

                try {
                    Thread.sleep ( lengthToSleep );
                } catch( Exception e ) {
                    post( "Error: " + e.getMessage() );
```

```
                }
                }

            }
        };
        t.start();

    }
}

public void open() {
    this.fName = sys.openDialog();
}

}
```

ShowXMLDataStore.java

```
public class ShowXMLDataStore {

    public String pitchText;
    public int dur;
    public int articulation;
    public int dynamic;
    public int pitch;

    public String getPitchText() {
        return pitchText;
    }

    public void setPitchText(String pitchText) {
        this.pitchText = pitchText;
    }

    public int getDur() {
        return dur;
    }

    public void setDur(int dur) {
        this.dur = dur;
    }

    public int getPitch() {
        return pitch;
    }

    public void setPitch(int pitch) {
        this.pitch = pitch;
    }

    public ShowXMLDataStore() {}

    public int getArticulation() {
        return articulation;
    }

    public void setArticulation(int articulation) {
```

```
        this.articulation = articulation;
    }

    public int getDynamic() {
        return dynamic;
    }

    public void setDynamic(int dynamic) {
        this.dynamic = dynamic;
    }

}
```

ShowXMLParse.java

```
import java.util.ArrayList;
import java.util.List;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.apache.xerces.parsers.SAXParser;
import com.cycling74.max.Atom;

public class ShowXMLParse extends DefaultHandler {

    private static boolean getChars = false;
    private static boolean isArticulation = false;
    private static boolean isDivisions = false;
    private static boolean isBeats = false;
    private static int isPitchText = 0;
    private static int isPitch = 0;
    private static int isDur = 0;

    private static boolean isStep = false;

    private static boolean isOct = false;
    private static boolean isAlter = false;
    public boolean isRunning = true;
    public static boolean isDynamic;

    public int row = 0;
    public int whichTagIsBeingParsed;
    private static int pitch = 0;

    public static Atom[] tags;
    public static String[] tagsToParse;

    public String step = "";
    public String dur = "";
    public String oct = "";
    public String alter = "";
    private int articulation;
    private int dynamic;

    public static int ctr = 0;
    static ShowXML play;
```

```
    public static List dataList;
    public static ShowXMLDataStore store;

    public ShowXMLParse() {
        super();
    }

    public void startElement( String namespaceURI, String localName,
String qName, Attributes atts ) {

        if( localName.equals( "divisions" ) ) isDivisions = true;
        if( localName.equals( "beats" ) ) isBeats = true;

        if( ShowXMLParse.isDynamic ) {
            if( localName.equals( "p" ) ) this.dynamic = 1;
            if( localName.equals( "pp" ) ) this.dynamic = 2;
            if( localName.equals( "ppp" ) ) this.dynamic = 3;
            if( localName.equals( "f" ) ) this.dynamic = 4;
            if( localName.equals( "ff" ) ) this.dynamic = 5;
            if( localName.equals( "fff" ) ) this.dynamic = 6;
            if( localName.equals( "mp" ) ) this.dynamic = 7;
            if( localName.equals( "mf" ) ) this.dynamic = 8;
            if( localName.equals( "sf" ) ) this.dynamic = 9;
            if( localName.equals( "sfz" ) ) this.dynamic = 9;
            if( localName.equals( "sfp" ) ) this.dynamic = 10;
            if( localName.equals( "sz" ) ) this.dynamic = 11;
        }

        if( ShowXMLParse.isArticulation ) {
            if( localName.equals( "accent" ) ) this.articulation = 1;
            if( localName.equals( "strong-accent" ) ) this.articulation
                = 2;
            if( localName.equals( "staccato" ) ) this.articulation = 3;
             if( localName.equals( "staccatissimo" ) )
                this.articulation = 4;
        }

        if( isDur == 1 ) {
            if( localName.equals( "duration" ) ) {
                isDur = 2;
```

```
        }
    }

    if( isPitch == 1 || isPitchText == 1 ) {
        if( localName.equals( "step" ) ) isStep = true;
        if( localName.equals( "alter" ) ) isAlter = true;
        if( localName.equals( "octave" ) ) isOct = true;
    }

}

public void endElement( String namespaceURI, String localName,
String qName ) {
    if( getChars ) getChars = false;
    if( localName.equals( "duration" ) && isDur == 2 ) isDur = 1;
    if( localName.equals( "step" ) ) isStep = false;
    if( localName.equals( "alter" ) ) isAlter = false;
    if( localName.equals( "octave" ) ) isOct = false;
    if( localName.equals( "divisions" ) ) isDivisions = false;
    if( localName.equals( "beats" ) ) isBeats = false;

    if( localName.equals( "note" ) ) {
        store = new ShowXMLDataStore();
        store.setDynamic( this.dynamic );
        store.setArticulation( this.articulation );
        if( this.dur.length() > 0 ) {
            store.setDur( Integer.parseInt( this.dur ) );
        }
        if( isPitch == 1 ) {
            store.setPitch( findPitch( step, alter, oct ) );
        }
        if( isPitchText == 1 ) {
            if( alter.length() > 0 ) {
                if( Integer.parseInt( alter ) == 1 )
                    store.setPitchText( step + "#" + oct );
                else store.setPitchText( step + "b" + oct );
                alter = "";
            } else store.setPitchText( step + oct );
        }
        dataList.add( store );
        this.dynamic = 0;
```

```
        this.articulation = 0;
    }
}

public void startDocument() {}

public void endDocument() {
    ShowXML.post( "<- PARSING COMPLETE " );
    ShowXML.isParsing = false;
}

public void characters ( char[] ch, int start, int len ) {

    if( isDur == 2 )
        this.dur = new String( ch, start, len ).trim();
     if( isStep == true ) this.step = new String( ch, start, len
         ).trim();
     if( isAlter == true ) this.alter = new String( ch, start, len
         ).trim();
     if( isOct == true ) this.oct = new String( ch, start, len
         ).trim();
     if( isDivisions ) ShowXML.divisions = Integer.parseInt( new
         String( ch, start, len ).trim() );
        if( isBeats ) ShowXML.beats = Integer.parseInt( new String(
     ch, start, len ).trim() );
}

public static int toMIDI( String step ) {
    if( step.equals( "C" ) ) {
        return 0;
    } else if( step.equals( "D" ) ) {
        return 2;
    } else if( step.equals( "E" ) ) {
        return 4;
    } else if( step.equals( "F" ) ) {
        return 5;
    } else if( step.equals( "G" ) ) {
        return 7;
    } else if( step.equals( "A" ) ) {
        return 9;
    } else {
```

```
            return 11;
        }
    }

    public static int findPitch( String step, String alter, String
oct ) {
        pitch = toMIDI( step );
        if( alter.length() > 0 ) {
            pitch =+ Integer.parseInt( alter );
            isAlter = false;
        }

        if( oct.length() > 0 ) {
            pitch += ( Integer.parseInt( oct ) + 1 ) * 12;
            isOct = false;
        }

        step = "";
        alter = "";
        oct = "";

        return pitch;
    }

    public static void ParseThis( String fName ) {
        ShowXMLParse s = new ShowXMLParse();
         SAXParser p = new SAXParser();
         p.setContentHandler(s);

         for( int i=0; i<tags.length; i++ ) {

            if( tags[i].toString().equalsIgnoreCase( "pitchInText" ) )
{
                isPitchText = 1;
            }

            if( tags[i].toString().equalsIgnoreCase( "pitch" ) ) {
                isPitch = 1;
            }

            if( tags[i].toString().equalsIgnoreCase( "duration" ) ) {
```

```
            isDur = 1;
            ShowXML.post( "Is Duration" );
         }

         if( tags[i].toString().equals("dynamics") ) {
            isDynamic = true;
         }

         if( tags[i].toString().equals("articulations") ) {
            isArticulation = true;
         }
      }

      try {
         dataList = new ArrayList();
         p.parse( fName );
      } catch ( Exception e ) {
          ShowXML.post( "Error: " + e.getMessage() + "\n" +
            e.toString() + "\n" + e.getCause() );
      }
      ctr = 0;
   }
}
```