# USING XQUERY ON MUSICXML DATABASES FOR MUSICOLOGICAL ANALYSIS

**Joachim Ganseman, Paul Scheunders**
IBBT - Visionlab
Dept. of Physics, University of Antwerp
Universiteitsplein 1, building N
B-2610 Wilrijk (Antwerp), Belgium
{joachim.ganseman, paul.scheunders}@ua.ac.be

**Wim D'haes**
Mu Technologies NV
Singelbeekstraat 121
B-3500 Hasselt, Belgium
support@mu-technologies.com

## ABSTRACT

MusicXML is a fairly recent XML-based file format for music scores, now supported by many score and audio editing software applications. Several online score library projects exist or are emerging, some of them using MusicXML as main format. When storing a large set of XML-encoded scores in an XML database, XQuery can be used to retrieve information from this database. We present some small practical examples of such large scale analysis, using the Wikifonia lead sheet database and the eXist XQuery engine. This shows the feasibility of automated musicological analysis on digital score libraries using the latest software tools. Bottom line: it's easy.

## 1 INTRODUCTION

MusicXML is an open file format developed by Recordare LLC [1]. Its development started around the year 2000 [2]. Some design decisions on the format are well explained in [3].

In 2002 a paper was presented on the XML conference [4] explaining how XQuery can be used to search for structures in a MusicXML document, but since then it has been very quiet on the front. In a 2007 poster, Viglianti [5] points out the possibilities of using XQuery for musicological analysis. Unfortunately, his text contains some errors - the most important one being the remark that XQuery would not support arrays. XQuery definitely does support arrays, it is even proved to be a Turing complete language [6]. Therefore has the same expressive power as languages like C++ or Java, and any computable function can be performed using XQuery.

In the meantime, XQuery 1.0 has made it to a W3C recommendation [7]. Galax [8] strives to be a reference implementation. eXist [9], Sedna [10], and Oracle's Berkeley DB XML [11] are native XML database management systems incorporating their own XQuery engines. All of those are open source and free. Both standards and software have thus matured.

Digital score libraries already exist, emanating from research (e.g. KernScores [12], based on HumDrum [13]) or the open source community (e.g. Mutopia [14], using Lilypond [15]). The Wikifonia project [16] uses MusicXML. It is a wiki-style collaborative environment for publishing lead sheets - i.e. reduced scores where arrangement details have been removed and only melody, lyrics and chord progression information is available. Wikifonia is at the moment of writing still relatively small. We worked with a database downloaded on April 1st 2008, which contained just over 200 songs.

To do automated musicological analysis, the HumDrum toolkit [13] is probably the most widely used today. It is very powerful, but it has its drawbacks. It works with input and output in its own plain text (ASCII) file format. Formatting output, converting to and from other formats, dynamically generating queries etc., require the creation of dedicated software or advanced scripts. To use it effectively, HumDrum requires a good deal of knowledge of UNIX style scripting, which is even more difficult to use on Windows platforms. On the other hand, XML formats are so generic that a huge set of software tools already exist to manipulate XML data in almost any way imaginable. Knowing that XQuery is Turing complete, we can state that it is theoretically possible to translate the whole HumDrum toolkit to XQuery.

There is not much software around that is capable of doing pattern analysis on MusicXML files: MelodicMatch [17] is at the moment of writing probably the most complete one. THoTH [18] also has some limited selection and extraction functionality based on the contents of a MusicXML file. Both software packages are closed source. Nevertheless, this kind of functionality is very desirable in music score software. For example, in regular PDF files, we are not impressed any more by functionality that allows us to search for a text pattern, highlighting all occurrences in the text itself. But in score editors, searching for a music pattern and highlighting the results in the score, is a functionality that is only noticeable by its absence.

In this paper we give some practical examples that illustrate how XQuery can be used to easily and quickly search large collections of MusicXML scores for interesting data, and return the results formatted in any way desired. We will point out some interesting directions for future projects. By showing that only a few proven, stable, off-the-shelf software components are needed for advanced score analysis, we think that advanced search and information retrieval functionality can be incorporated in music score software very quickly.

## 2 USED TECHNOLOGIES

It is not our goal to provide an in-depth tutorial on MusicXML [1], XPath [19] or XQuery [7]. Numerous excellent books or online tutorials can be found on those subjects, and the websites of the respective projects contain valuable information. Nevertheless, to understand the examples in this paper, a small introduction is given in the next paragraphs.

MusicXML can encode scores in 2 ways: part-wise or time-wise. In a part-wise score, the file is roughly structured as follows: a score contains a number of parts, a part contains a number of measures, and a measure contains a number of notes. For a time-wise score, measure and part are switched in the hierarchy. An XSLT file (eXtensible Stylesheet Language Transformation) can be used to convert between the 2 formats, with a notable exception: multimetric music - where different parts have different time signatures - can only be properly encoded in part-wise scores. In practice, most MusicXML files today are part-wise encoded, and for the rest of this paper, we will assume part-wise file structure. Example files of MusicXML documents can be found on Recordare's website [1].

XPath is a language for retrieving information from an XML document. It is relatively easy to understand, and is used by XQuery to traverse XML documents. A complete reference of the latest W3C recommendation, version 2.0, can be found online [19]. Next to the document traversal functionality, XPath contains a wide range of operators for union and intersection, comparison, arithmetic, etc. A list of about 100 built-in functions completes the XPath language. Important to note is that an XPath query preserves document order.

XQuery extends XPath, making it more suitable for larger queries than XPath is. The most important extension is the so-called FLWOR (pronounce 'flower') construct - acronym for 'for, let, where, order by, return' - describing the typical form of an XQuery query. This is roughly analogous to the 'select, from, where' structure used in SQL (Structured Query Language). Results of XQuery FLWORs can be returned in any desired XML format, including XHTML, allowing immediate inclusion in a dynamic webpage. A second extension is the possibility to define functions yourself, functions which may be recursive, making XQuery Turing complete. Last but not least, *(: comments look like this :)*.

eXist [9] is an open source XML database management system, written in Java. It implements XPath 2.0, XQuery 1.0, and some other minor standards, recommendations and drafts. Several HTTP interfaces are provided, eXist can be easily deployed to run a whole database-driven website. In this paper, we used the latest stable release at the moment of writing, which is 1.2.0-rev7233. There was no specific reason to choose eXist over alternatives. It is a small project, easily set up on all kinds of platforms, with a very fast and complete XQuery engine and a simple graphical user interface. For a few queries to succeed, we needed to adapt the standard settings of eXist first: the maximum result size needed to be increased in the configuration file, and the maximum heap size of the Java virtual machine needed to be increased in the startup batch file.

## 3 QUERYING WIKIFONIA

In the next examples, we show some simple queries to retrieve basic information from the database. They can eventually be wrapped in a function to be used in more complex queries afterwards. To save the bandwidth of Wikifonia and speed up processing, we made a local copy of the database and worked from there. Wikifonia is at the moment of writing just a file repository, offering easy access to individual files with URLs that can be generated automatically. Iterating over all online documents and saving the results to a local database was done on April 1st, 2008, using the query in Listing 1, actually just a file dump.

Listing 1. Downloading the database. The URL of each document is composed using the *concat()* function that is built into XPath.

```
<wfdb> {
for $i in (0 to 1000)
let $s := concat("http://static.wikifonia.org/",
     $i,"/musicxml.xml")
return doc($s)
} </wfdb>
```

A better approach when working with large databases, is using 'collections', storing each file separately into the database. However, database manipulation functions are not part of the XQuery standard and can differ depending on the XQuery engine used. To keep things as uniform and clear as possible, we will keep working on a single file here, which we will call '*wfdb.xml*', that contains all songs. Since the database is relatively small, this is not problematic here: the file is only about 30 MB large and contains approximately 750000 lines of text.

## 3.1 Database contents and statistics

The number of note elements (this also includes rests, grace notes and cue notes) in the each song can be counted using the query in Listing 2. The largest song contains 1764 notes and the smallest only 5. Wrapping that code in the built-in *sum()* function and adapting it slightly informs us of a total of 50846 `<note>` tags in the database. Only the tag name needs to be adapted in the query to return the count of any tag in a MusicXML file, parts of a file, or the whole database.

Listing 2. Count the number of notes in each song. Element and attribute constructors are used in the return clause for mark-up.

```
for $i in doc("wfdb.xml")//score-partwise
let $j := count($i//note)
order by $j descending
return element{"song"}{attribute{"title"}
    {$i//movement-title/text()},
    attribute{"note-count"}{$j}}
```

Next, we can retrieve songs that meet certain criteria. The query in Listing 3 finds the song with the least number of notes and prints all notes that occur in that song. At the same time, it demonstrates the ability of XQuery to use nested queries. The query in Listing 4 returns a list of titles of those songs that have no rests in them.

Listing 3. Get the song with the least number of notes

```
for $i in doc("wfdb.xml")//score-partwise
let $j := $i//movement-title [ count($i//note) eq
  min( for $x in doc("wfdb.xml")//score-partwise
     return count($x//note)
  )]
return $j/..//note
```

Listing 4. Get the titles of songs with no rests

```
for $i in doc("wfdb.xml")//score-partwise
return $i[count($i//rest) eq 0]//movement-title
```

A music database will most likely contain some redundancies. The same song can be present in different editions, covered by other musicians, or there can be reductions for other instruments in separate files. These are just a few of the possibilities that cause the same song to be included in the database multiple times. Executing the query in Listing 5 checks only the titles, revealing that there are 3 songs called "Summertime" stored in the database and 9 other songs stored 2 times - as shown in Listing 6. Similarly, the query in Listing 7 allows us to find composers that have several compositions in the database. We found 12 "Traditional" songs in the database, 6 by Duke Ellington, 4 by A.L. Webber, etc.

Listing 5. Get the songtitles that are stored more than once in the database.

```
for $i in distinct-values(doc("wfdb.xml")//
    movement-title/text())
let $c := count(doc("wfdb.xml")//movement-title
    [text() = $i])
order by $c descending, $i
return ( element{"song"}{attribute{"count"}
    {$c}, $i} )[$c gt 1]
```

Listing 6. Part of the results of the query in Listing 5

```
<song count="3">Summertime</song>
<song count="2">Bernie's Tune</song>
<song count="2">Cherokee</song>
<song count="2">Could It Be Magic</song>
<song count="2">Everybody Hurts continued</song>
```

Listing 7. Sort authors that have more than 1 song in the database. The *lower-case()* function was used to iron out inconsistencies in capital letter usage.

```
let $multiset := doc("wfdb.xml")//creator
  [@type="composer"]/lower-case(text())
let $set := distinct-values($multiset)
for $i in $set
let $c := count(index-of($multiset, $i))
order by $c descending, $i
return ( element{"author"}{attribute{"count"}
  {$c}, $i} )[$c gt 1]
```

Eventually, we want to go looking for musically significant information. Using the same techniques as used above, Listing 8 returns the titles of all songs that are written in c minor. A more complex variant of this query, searching through all chords in a piece, is of special interest to people who are learning the guitar and only know a limited number of chords. They could then search for songs that only contain this limited set of chords, and thus find music they can already play completely despite a limited knowledge.

Listing 8. Find titles of all songs in c minor.

```
let $f := "-3"
let $m := "minor"
for $i in doc("wfdb.xml")//key
```

```
  [fifths/text() eq $f][mode/text() eq $m]
return $i/ancestor-or-self::score-partwise//
    movement-title
```

## 3.2 Database statistics

Using the built-in aggregation and arithmetic functions, one can generate very detailed statistics of the database contents. A first rudimentary example is presented in Listing 9. It iterates over the nominators and denominators of the time signatures that are present in the database, and generates all possible time signatures out of them. For each of these time signatures is calculated how many times it occurs in the database. This is converted to a percentage value and the results are ordered by that value. To keep the example simple, Listing 9 does not contain code to keep non-occurring time signatures from being generated. Instead, at the end the results having a count of 0 are removed from the list using the selection `[$c gt 0]`. A part of the results is presented in Listing 10. Note that time signatures can change in the middle of a song. Due to the query requesting all time signatures in the database wherever they occur, these time signature changes are also taken into account here.

Listing 9. Database statistics on time signatures.

```
let $t := count(doc("wfdb.xml")//time)
for $i in distinct-values(doc("wfdb.xml")//beats)
  for $j in distinct-values(doc("wfdb.xml")//
      beat-type)
  let $c := count(doc("wfdb.xml")//time
      [beats eq $i][beat-type eq $j])
  let $p := $c div $t * 100
order by $p descending
return element{"time"} {
  attribute{"beats"}{$i},
  attribute{"beat-type"}{$j},
  element{"count"}{$c},
  element{"percentage"}{$p}
} [$c gt 0]
```

Listing 10. Results of the query in Listing 9

```
<time beats="4" beat-type="4">
    <count>238</count>
    <percentage>49.6868475991649269</percentage>
</time>
<time beats="2" beat-type="4">
    <count>98</count>
    <percentage>20.4592901878914405</percentage>
</time>
<time beats="3" beat-type="4">
    <count>88</count>
    <percentage>18.37160751565762</percentage>
</time>
```

The same can be done for key signatures. In the code in Listing 11, all key signatures from 7 flats to 7 sharps are generated and both minor and major mode are considered. The first part of the result of this query is presented in Listing 12. Here too, note that key signatures can change in the middle of a song, and those key alterations are also present in the results. With some extra code this can be avoided if desired.

Listing 11. Database statistics on key signatures.

```
let $t := count(doc("wfdb.xml")//key)
for $i in (-7 to 7)
  for $j in ("minor","major")
  let $c := count(doc("wfdb.xml")//key
      [fifths eq string($i)][mode eq $j])
  let $p := $c div $t * 100
order by $p descending
return element{"key"} {
  attribute{"fifths"}{$i},
  attribute{"mode"}{$j},
  element{"count"}{$c},
  element{"percentage"}{$p}
} [$c gt 0]
```

Listing 12. Results of the query in Listing 11

```
<key fifths="0" mode="major">
    <count>64</count>
    <percentage>21.3333333333333333</percentage>
</key>
<key fifths="-1" mode="major">
    <count>39</count>
    <percentage>13</percentage>
</key>
<key fifths="-3" mode="major">
    <count>34</count>
    <percentage>11.3333333333333333</percentage>
</key>
```

A last query which is quite complex, is shown in Listing 13. It requests all lyrics of each song, sorts them by verse number, puts the syllables together if necessary so that they form words and sentences, and outputs them into what can be called a 'lyric library'. It can handle the special case that when there is only 1 verse present, it does not need to be numbered explicitly in MusicXML. Lyrics are only printed if there are any.

Listing 13. Extracting lyrics from each song and compiling them into a library.

```
<library> {
for $i in doc("wfdb.xml")//score-partwise
let $tit := $i//movement-title/text()
let $aut := $i//creator[@type="composer"]/text()
return
<song>{attribute{"title"}{$tit}}
  {attribute{"composer"}{$aut}}
{
  let $lyr := $i//lyric
  let $nrv := if(empty($lyr/@number)) then 1
    else xs:integer(max($lyr/@number))
  for $cur in (1 to $nrv)
  let $ver := $lyr[ if($nrv gt 1) then
```

```
    @number = $cur else true() ]/text
  let $s := string-join(
    for $syl in $ver return concat($syl/text(),
      if ($syl/../syllabic = ('begin','middle'))
      then '' else ' '), '')
  return
   <lyric>{attribute{"verse"}{$cur}}{$s}
      </lyric> [not(empty($lyr))]
} </song>
} </library>
```

### 3.3 Querying musical structure

Finding all occurrences of a single rhythmic or melodic motive, or a chord sequence, is a basic task when making a music theoretical analysis of a piece of music. In the next example, we work on the score of the 'Blue Danube' ('An der schönen blauen Donau'), the well-known waltz by Johann Strauss jr. The code in Listing 14 extracts all occurrences of the rhythmic pattern consisting of three notes of the same length followed by a note that is 3 times as long as the previous ones. This allows not only to find the pattern of 3 quarters followed by a dotted half note, but also faster or slower occurrences of the same rhythm - the comparison is based on relative duration values. This specific code example requires that all notes appear next to each other and in the same voice. The returned value indicates the position where the motive can be found. By simply removing the selection of a specific song, we can search the whole database for the specified motive.

Listing 14. Finding a rhytmic motive in a single song.

```
let $bd := doc("wfdb.xml")//score-partwise
    [movement-title = 'Blue Danube']
let $notes := $bd//note
for $i in (0 to count($notes))
let $s := subsequence($notes, $i, 4)
  (: now select those subsequences with the
     last note duration being 3 times that of
     the previous ones :)
let $durs := $s/duration
let $voic := distinct-values($s/voice)
where count($s/rest) = 0
and count($voic) = (0,1)
  (: '=' computes an intersection,
     whereas 'eq' compares single values :)
and $durs[1]/text() eq $durs[2]/text()
and $durs[2]/text() eq $durs[3]/text()
and number($durs[4]) eq number($durs[1]) * 3
return
<motive>
  <measure-start>{$s[1]/../@number/string()}
     </measure-start>
  <note-start>{index-of($s[1]/../note,$s[1])}
     </note-start>
</motive>
```

In [4] an XQuery function is presented that calculates MIDI pitch values of notes. The *where* clause in Listing 14 can be replaced by the code in Listing 15 which uses the aforementioned function. This example searches for the melodic motive of a large triad, followed by repetition of the last note. The triad can begin at any pitch, since only the intervals are taken into account.

Listing 15. Code Excerpt: Finding a melodic motive.

```
let $pits := $s/pitch      (: all pitch values :)
where count($s/rest) = 0      (: no rests :)
and count($voic) = (0,1)
and count($pits) = 4
and MidiNote($pits[1])+4 eq MidiNote($pits[2])
and MidiNote($pits[2])+3 eq MidiNote($pits[3])
and MidiNote($pits[3]) eq MidiNote($pits[4])
```

In [5] is pointed out that the generation of permutations of a sequence is of importance in certain music analysis tasks. A function can be written to generate all permutations of a sequence: XQuery allows defining recursive functions. But since permutation generation has a time complexity of order *O(n!)*, this is only usable on a very small scale. Also, when considering all possible permutations of a chord, transpositions need to be eliminated: this can be accomplished by calculating pitch values modulo 12.

## 4 FUTURE WORK

Musicological analysis is much more than just finding motives. The creation of an XQuery function database, a 'toolbox' for often used analysis tasks is one of the several practical projects that could (and, we think, needs) be undertaken to further automate large-scale analysis. An example of a practical tool that can be made available to the public is a web interface on top of the code presented here. This is not difficult to accomplish, since most XML database systems can be run as database back-end.

Of special interest are also regular expressions, the main power of the HumDrum toolkit [13]. The standard UNIX tools that it uses provide advanced regular expression processing on plain text files, but implementing this functionality in Xquery for XML files might be tricky - though it is theoretically possible.

Beyond merely statistical and structural analysis lies the domain of data mining. Here we try to isolate those patterns in the database that occur often, without having to define a specific pattern first. The Apriori algorithm [20] used to mine association rules in transactional databases has been translated to XQuery by Wan and Dobbie [21], and could be applied to a MusicXML database to search for structures that often appear together.

## 5 CONCLUSION

In this work we have shown the feasibility of using XQuery to mine a repository of music scores. We gave several examples of simple queries that can be run on a MusicXML database. They can form the basis for more complex queries. The strength of the XQuery language, the scalability of XML databases, the growing software support for the MusicXML format, combined with an increasing availability of digital scores, enable powerful musicological analysis with only a few lines of code. Awareness of the possibilities is an important factor in getting these XML-based technologies further developed. They allow musicologists, musicians, and the occasional music hobbyist to extract more significant information from an ever growing set of available scores in a very accessible way.

## 6 REFERENCES

[1] Recordare LLC, "MusicXML definition, version 2.0," Available at http://www.recordare.com/xml.html, accessed April 1, 2008.

[2] Michael Good, "Representing music using XML," in *Proc. 1st International Symposium on Music Information Retrieval (ISMIR 2000)*, Plymouth, Massachusetts, USA, Oct. 23-25 2000.

[3] Michael Good, "Lessons from the adoption of MusicXML as an interchange standard," in *Proc. XML 2006 Conference*, Boston, Massachusetts, USA, Dec. 5-7 2006.

[4] Michael Good, "MusicXML in practice: issues in translation and analysis," in *Proc. 1st International Conference on Musical Applications Using XML (MAX 2002)*, Milan, Italy, Sept. 19-20 2002, pp. 47–54.

[5] Raffaele Viglianti, "MusicXML: An XML based approach to automatic musicological analysis," in *Conference Abstracts of the Digital Humanities 2007 conference*, Urbana-Champaign, Illinois, USA, Jun. 4-8 2007, pp. 235–237.

[6] Stephan Kepser, "A simple proof for the turing-completeness of XSLT and XQuery.," in *Proceedings of the Extreme Markup Languages 2004 Conference*, Montréal, Quebec, Canada, Aug. 2-6 2004.

[7] World Wide Web Consortium (W3C), "XQuery 1.0: An XML Query Language - W3C Recommendation 23 January 2007," Available at http://www.w3.org/TR/xquery/, accessed April 1, 2008.

[8] The Galax Team, "Galax," Available at http://www.galaxquery.org/, accessed April 1, 2008.

[9] Wolfgang Meier and contributors, "eXist - open source native XML database," Available at http://exist.sourceforge.net/, accessed April 1, 2008.

[10] Modis Team, "Sedna," Available at http://www.modis.ispras.ru/sedna, accessed March 31, 2008.

[11] Oracle Corporation, "Oracle Berkeley DB XML," Available at http://www.oracle.com/technology/products/berkeley-db/xml/index.html, accessed June 19, 2008.

[12] Craig Stuart Sapp, "Online database of scores in the Humdrum file format," in *Proc. 6th International Conference on Music Information Retrieval (ISMIR 2005)*, London, UK, Sept. 11-15 2005, pp. 664–665.

[13] David Huron, "Music information processing using the Humdrum toolkit: Concepts, examples, and lessons," *Computer Music Journal*, vol. 26, no. 2, pp. 11–26, 2002.

[14] Chris Sawer and David Chan, "Mutopia," Available at http://www.mutopiaproject.org/, accessed April 1, 2008.

[15] Han-Wen Nienhuys, Jan Nieuwenhuizen, and contributors, "GNU Lilypond," Available at http://lilypond.org/, accessed April 1, 2008.

[16] Wikifonia Foundation, "Wikifonia," Available at http://www.wikifonia.org/, accessed April 1, 2008.

[17] Philip Wheatland, "MelodicMatch music analysis software," Available at http://www.melodicmatch.com/, accessed June 19, 2008.

[18] Steve Carter, "THoTH," Available at http://www.frogstoryrecords.com/dev/thoth, accessed June 19, 2008.

[19] World Wide Web Consortium (W3C), "XML Path Language (XPath) 2.0 - W3C Recommendation 23 January 2007," Available at http://www.w3.org/TR/xpath20/, accessed April 1, 2008.

[20] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proc. 20th International Conference on Very Large Data Bases*, Santiago, Chile, Sept. 12-15 1994, pp. 487–499.

[21] Jacky W. W. Wan and Gillian Dobbie, "Extracting association rules from XML documents using XQuery," in *Proc. 5th ACM international workshop on Web information and data management*, New Orleans, Louisiana, USA, Nov. 7-8 2003, pp. 94–97.