

# An Automaton-Based Filtering System for Streaming MusicXML

Ai-Ti Chiu and Jia-Lien Hsu

**Abstract**—In the research field of music information retrieval, a precise and efficient query/filter system are critical to provide flexible services and applications. Meanwhile, the MusicXML is considered as one of the most popular data format for better structured description on music objects. With the emerging streaming data model, we propose an efficient filter system against streaming MusicXML.

User queries are specified in XPath expressions. Three key issues are investigated in this system, including *query notes of crossing measures, polyphonic music query, and the order of query notes*. We propose an automaton-based method to efficiently resolve user queries against streaming MusicXML.

## I. INTRODUCTION

In the researches of processing XML, most researchers focus issues on XML filtering and XML stream processing. In [1], [2], [4], [5], the corresponding authors proposed their own data structure to represent user queries. According to the proposed data structure, There are two types of works, automaton-based methods and index-based methods. Most researches apply automaton-based method, because it allows more complicated user queries. Automaton-based system also queries/filters more complicated XML documents. As long as user queries are transformed into an automaton, the incoming XML documents are navigated through the automaton to identify the matched queries and users.

In [2], single XPath expression (user query) constructs a hierarchical pushdown transducer (HPDT). The authors defined pushdown transducer template for each location step in a XPath expression. Then, lots of pushdown transducer template are combined into one hierarchical pushdown transducer. In [1], XPush Machine has been proposed to process streaming XML documents. They also focus issues on processing many XPath expressions in the same time. Each of the XPath expressions has many predicate expressions need to be evaluated.

Our method is based on XPush Machine. However, XPush Machine is not robust for processing streaming MusicXML documents. Fig. 1 shows the system architecture. XPush Machine is constructed based on user queries, and incoming streaming MusicXML is filtered by XPush Machine to evaluate the There are three key issues when apply XPush Machine to process streaming MusicXML, including “query notes of crossing measures,” “polyphonic music query,” and “the order of query notes.”

**Query notes of crossing measures:** An user query is a piece of music object, and notes in an user query may cross

This research was partially sponsored by the Republic of China, National Science Council under Contract No. NSC-94-2213-E-030-014.

Ai-Ti Chiu and Jia-Lien Hsu are with Department of Computer Science and Information Engineering, Fu Jen Catholic University, Hsinchuang, Taipei Hsien 24205, Taiwan, R.O.C. (email: alien@csie.fju.edu.tw).

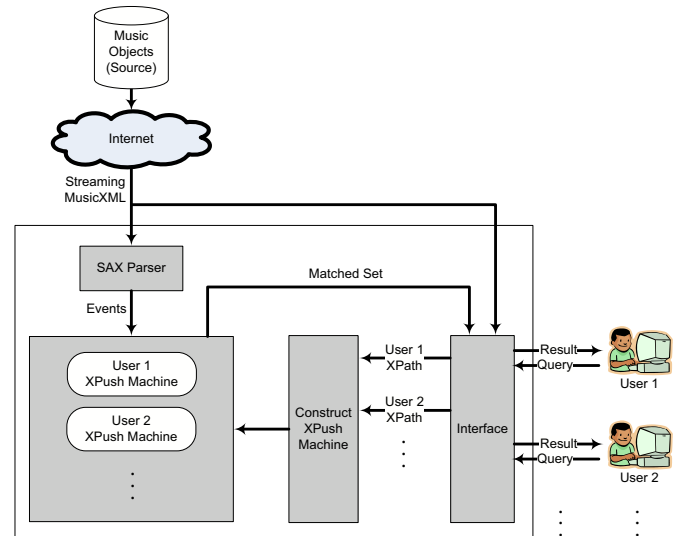


Fig. 1. The system architecture.

many measures in the music object. In other words, such query can be viewed as query crossed sibling nodes of XML tree.

**Polyphonic music query:** An user query can be piece of polyphonic music. To address this issue, we need to realize that how polyphonic music is described by MusicXML. MusicXML uses some key tags to represent that notes appear in the same time, including `<chord>` and `<backup>`.

**The order of query notes:** The range of a note pitch is from A to G. If the order of user query were: A B C, and the order of incoming streaming MusicXML notes were: A B C, or A C B, or B A C, or B C A, or C A B, or C B A. In this situation, XPush Machine will match all six incoming streaming MusicXML. But there is only one correct answer which is the first one (A B C) above.

## II. PRELIMINARIES

- MusicXML: MusicXML describes a music object based on XML. MusicXML inherit XML property that makes it became a standard music description format in music querying or analyzing system. For example, Fig. 2 shows the first measure of Ludwig van Beethoven, “An die ferne Geliebte, op.98.” Fig. 3 shows the corresponding MusicXML.
- XPath: XPath expressions view XML documents as tree and locate some tags in the XML documents. For example,  $XP1$  is a XPath expression.

$$XP1 = /measure/note[pitch/step/text() = “B”]$$

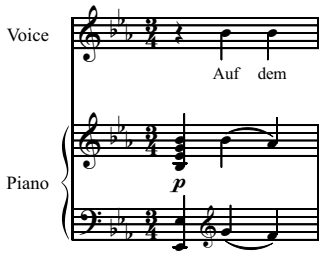


Fig. 2. Ludwig van Beethoven, “An die ferne Geliebte, op.98,” measure 1. [3]

```

<?xml version="1.0" standalone="no" ?>
<!DOCTYPE score-partwise (View Source for full doctype...)>
<score-partwise version="1.0">
+ <work>
+ <identification>
+ <part-list>
- <part id="P1">
- <measure number="1">
- <note>
  <rest />
  <duration>24</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
- <note>
- <pitch>
  <step>B</step>
  <alter>-1</alter>
  <octave>4</octave>
</pitch>
  <duration>24</duration>
  :
  :
</note>

```

Fig. 3. The corresponding MusicXML of Fig. 2 (Because of space limitation, only representative tags are shown in the figure.). [3]

This XPath expression want to locate the “notes” which have “step” grandchild, and text of step must be “B”.

### III. MAIN RESULTS

#### A. Query Notes of Crossing Measures

The reason why user query notes may cross measures is that user does not know the attribute of <measure> in MusicXML. It means that this situation happens frequently and must be solved. This issue can be viewed as query crossed sibling nodes of XML tree. A stack used in XPush Machine maintains current matched states of automaton against incoming streaming XML. If we applied XPush Machine, when the first </measure> is met and user query is not all evaluated, XPush Machine will return “not match” to user. But, if user query notes cross measures, we have to retain the matched states in the first measure and combined with other matched states in the next measure. We push the current matched states into stack when processing the first </measure> and apply XPush Machine to process the following data. When the next </measure> is met, we union all contents in the stack (means we combined all matched states). Then the system can evaluate user query correctly, and this address query notes cross measures issue.

#### B. Polyphonic Music Query

There may have some notes are played in the same time, called *chord*. So, user query may include chord. MusicXML uses <chord> and <backup> to present these notes. If these notes were connected with one stem, then MusicXML will mark a <chord> in each of them except the first note. If these notes were not connected, then MusicXML will mark a <backup> to identify how many duration should the consequent notes trace back.

We employ an auxiliary array to handle this. The array  $p$  ( $p[3 \times m]$ , where  $m$  denotes the number of notes in the current measure) is initialized again when encountering each <measure>. When the first note in a measure is met,  $p_{1,1}$  is set to 0. Then  $p_{1,2}$  is set to the duration of current note, and  $p_{1,3}$  is set to the pitch (from A to G) of current note.

The other notes in the same measure will be processed following.  $p_{2,1}$  is set to  $p_{1,2}$ .  $p_{2,2}$  is set to the sum of current note’s duration and  $p_{2,1}$ . This means we aggregate duration of current measure for future used.  $p_{1,3}$  is set to the pitch of current note.

When encountering the <chord> and <backup>, the array will be processed following. If note in row  $i$  included <chord>, then  $p_{i,1}$  is set to  $p_{i-1,1}$ .  $p_{i,2}$  and  $p_{i,3}$  will maintain the former procedure. If note in row  $i$  included <backup>,  $p_{i,1}$  is set to the value subtracted number between <backup> from  $p_{i-1,2}$ . Then  $p_{i,2}$  and  $p_{i,3}$  still maintain the former procedure.

#### C. The Order of Query Notes

When we send a query: A B C, we need to match the exact order of note sequence in streaming MusicXML. We construct Alternating Finite Automata (AFA) [1] based on user query, hence we know what order of the matched states should be. When system filtering incoming streaming MusicXML, it evaluates that to push the states into stack or not based on the user query order. It makes sure that system filtering incoming streaming MusicXML in the correct order.

### IV. CONCLUSIONS

In this paper, we identify three issues when querying music objects. These issues may not only exist when posing queries against MusicXML documents but also general XML documents. We propose an efficient method based on XPush Machine for querying/filtering streaming MusicXML documents.

### REFERENCES

- [1] A. K. Gupta and D. Suciu, “Stream Processing of XPath Queries with Predicates,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003, pp. 419–430.
- [2] F. Peng and S. S. Chawathe, “XPath Queries on Streaming Data,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003, pp. 431–442.
- [3] MusicXML <http://www.recordare.com/>
- [4] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu, “Processing XML Streams with Deterministic Automata,” in *Proceedings of the 9th International Conference on Database Theory*, 2002, pp. 173–189.
- [5] X. Gong, W. Qian, Y. Yan, and A. Zhou, “Bloom Filter-based XML Packets Filtering for Millions of Path Queries,” in *Proceedings of the 21st International Conference on Data Engineering*, 2005, pp. 890–901.