# OMDN at the MediaEval 2015 C@merata Task

Donncha S. Ó Maidín
Department of CSIS
University of Limerick
Limerick, Ireland
Donncha.OMaidin@ul.ie

## ABSTRACT

The task is concerned with using a natural language query and a file containing an encoding of a music score to produce a list of locations in the score that match the query. This task is achieved by (1) parsing the input query string using string processing, (2) formation of matching templates and (3) performing the search on the score and (4) reporting results in specified format. Searches on the score are performed using CPNView [2,3,4], a score container-iterator representation is generated from MusicXML code.

## 1. INTRODUCTION

The focus of this work is on searches of sequences of notes and rests. The tackling of this sub-set of the queries involving the matching of strings of notes and the rests, can be broken down into elements representing the notes or the rests separated by operators that link elements together. Some examples are Ab2 followed by Eb3 (operator ' followed by '), Bb5, G5, F5, E5, Eb5 (operator ', '), F# E G F# A (operator ' ').

### 1.1 Operators

The three different operators are contained in the above strings; (1) ' followed by ', (2) ', ' (comma + space), and (3) ' ' (space). In order to correctly parse the string, it is essential to take into account the context-sensitive nature of some of these operators.

### 1.2 Element Core

The core forms the unqualified part of the query element that identifies some basic feature of a note or rest, such as pitch, pitch class and duration, either singly or in combination.

| Type | Examples |
|---|---|
| pitch class (note) | A, Bb, C sharp |
| Pitch (note) | A3, Bb4, C4 sharp |
| pitch class and duration (note) | A crotchet, eighth note A, A eighth note |
| pitch and duration (note) | A#4 crotchet, crotchet A4 sharp. |
| Duration (note) | sixteenth note, quaver |
| Rest | sixteenth note rest, quaver rest |

### 1.3 Element Pre-Qualifiers

Element pre-qualifiers precede the Element Core. They serve a number of functions. These include the specification of repetition such as 'melody lasting 8 ....'; ' melody of seven ...'.

Also the pre-qualifier is used to specify an attribute of an element, either on its own or together with a repetition specifier:

trill on a quaver A. [attribute + ' on a ' + core element]
slurred .. [ adjectival attribute + core element]
ten staccato …[ repetition + attribute + core element]

### 1.4 Element Post-Qualifiers

Post-qualifiers are used to limit the search to elements with a specific instrument or clef, time signature, articulation, or location: in the Violins 2, in the treble clef, in bars 22-32, in the right hand, in 3/4 time, on the word "..."

### 1.5 CPNView

Common Practice Notation View, or CPNView is used to answer a subset of the questions in the C@merata challenge [1]. CPNView formed the main topic of a PhD dissertation [2]. The name CPNView was not used in the dissertation, but appeared in in later publications [3][4][5].

The meaning of symbols in a score depends on their preceding context. The emphasis a performer places on a note, for example is influenced by its position in the bar and by the time signature. Such contexts may be modified by an attached symbol such as marcato placed on an otherwise unstressed note. Contextual mechanism is employed in pitch representation in which key signature, clef and accidental alterations play a part. In CPNView, the user is freed from the need to keep track of such scoping concerns, as contexts are made available automatically by the iterator class.

CPNView models a score as an objected-oriented container. The CPNView model is designed to provide a value-neutral and objective representation of a score from common-practice notation. The score's internal content is available using iterators. The iterators and their member functions can be viewed as paralleling the actions of a human reader. A score object is created by specifying a file path:

*Score score(path);*

This model requires no user knowledge of how the score is represented in a file. CPNView representation is built from a software component that imports from files in a number of different standard encodings.

Access to the internals of the score is facilitated by an iterator object:

*ScoreIterator cursor(score);*
or *ScoreIterator cursor(score, 1);*

The first instance creates an iterator that initially points to the start and is used to visit all of the objects in a score in time order. Where the score contains multiple staves, this is an appropriate

*MediaEval 2015 Workshop*, September 14-15, 2015, Wurzen, Germany.

iterator for harmonic analysis. The second form, with an additional parameter 1 in this case is used to iterate a single stave.

In either case the iterator can be made to step through all of the objects in the score using the *step* member function. The *step* function returns a value *true* as long as a succeeding object exists. The following code skeleton makes all objects available, in sequence to any code that replaces the ellipsis.

```
while ( cursor.step()) {...}
```

If it is required to visit only the notes in the score, a parameter may be given to the step function as in the following code to count the notes in a score.

```
long count = 0;
while ( cursor.step(NOTE)) {count++;.}
```

A *locate* member may be used to place the score iterator in an arbitrary position. For example the iterator may be positioned at the start of bar 20 by means of

```
cursor.locate(BAR, 200);
```

The ScoreIterator object has a comprehensive range of member functions to retrieve all of the information that is contained within the score.

A natural language query that searches for all of the D notes and prints details of each note arrived at is achieved by

```
while ( cursor.step(NOTE))
if ( cursor.getAlpha() == 'D' )
cout << cursor << "\n"; (1)
```

In addition to modeling a score, CPNView has a set of components that facilitate processing musical information. They include container/iterator classes for Lists, Sets and Stores.

A specialised class exist for calculating pitch class sets. The pitch class object is based on a modified version of the classification system of Alan Forte [6]; see also [2].

## 2. APPROACH

### 2.1 Parsing the Input String

A string, S(n) with n elements separated by n-1 operators is processed as follows. A list consisting of search template records is formed where each string element generates a corresponding search record, in sequence.

| Step | Form | Action |
|---|---|---|
| 1 | S(1) | Parse, create a search template record, and add the template record to the template list, exit and perform search |
| 2 | S(n) | Structure the string as [ S(1) + operator + S(n-1) ], process S(1) to create a template list record as in step (1) and recursively apply step (2) to S(n-1) until exit from step (1). |

### 2.2 Search Template

The template is a C++ record consisting of native C++ types, and various convenient types from CPNView, as well as a comparisonType that is used to select the type of search. The comparisonType value is created from an analysis of the core of the search string element. The part of the template used for matching note and rest core elements contains identifiers for the comparison type, for pitch, duration and attribute details.

Search details from the pre- and post element qualifiers are encoded in additional fields of the search template. These include the use of a CPN Set class for storing attributes of notes, search range, and various fields that arise in a search including instrument, tempo, clef, word, and others.

### 2.3 The Search

The task of matching sequence of notes and rests involves matching members of the template list and exhaustively searching the score for sequences of notes and rests that match the list of templates.

```
For each stave
    A:  set n = 1
    B:  set m = 1
        set nn = n
    C:  visit element nn on stave and
                check for match with template list record m
        if element does not match template list record go to D:
        if template list at end record a match and go to D:
        else increment nn and m and go to C:
    D:  if more elements in stave, increment n and go to B:
        else if all staves processed exit
        else move to next stave
```

## 3. RESULTS AND DISCUSSION

Although the implementation of pre- and post-qualifiers and of inter-element operators had previously produced valid results, these had to be disregarded at the time of submission due to malfunction in the CPNView/MusicXML component that was under development. Previously functioning harmonic analysis had to be abandoned for similar reasons.

| Number of searches performed | Number of matches | Invalid results from misnumbered bars | Disputed results |
|---|---|---|---|
| 36 | 87 | Mozart's An Chloe & Horn Duo | Nos. 23 32 42 |

## 4. CONCLUSION

### 4.1 Observation

With the exception of the two invalid sets of results other case achieved scores of 1.0 in beat precision, beat recall, measure precision and measure recall. Other cases arose from either incompatible bar numbering or from case where the results are disputed.

### 4.2 Recommendations for a Line of Questioning

The rules of either strict or free counterpoint are well documented. These might provide a material for constructing future questions. In a future iteration of C@merata, some questions relating species counterpoint might be considered, starting with first species counterpoint. Success in this endeavour, especially if the five counterpoint species are eventually explored might lead to the development of useful tools for composition students.

## 5. REFERENCES

[1] Sutcliffe, R., Fox., C, Hovy, E., Root, D.L., Lewis, R. Task Description v2 C@MERATA 15: Question Answering on Classical Music Scores. On http://csee.essex.ac.uk/camerata

[2] Ó Maidin, D.S. 1995. A Programmer's Environment for Music Analysis. PhD Thesis, University College Cork, 1995.

[3] Ó Maidin, D.S. and Cahill, M. 2001. Score Processing for MIR. In *Proceedings of the International Society for Music Information Retrieval,* Bloomington, Indiana 2001, pp.59-64.

[4] Ó Maidin, D.S. 1998. A Geometrical Algorithm for Melodic Difference. In *Computing and Musicology II. Melodic Similarity. Concepts, Procedures, and Applications,* ed Walter B. Hewlett and Eleanor Selfridge-Field (Cambridge Massachusetts and London, The MIT Press, 1998), pp. 65-72.

[5] Forte, A. 1973. *The Structure of Atonal Music.* (New Haven and London: Yale University Press, 1973).