The CLAS System at the MediaEval 2017 C@merata Task

Stephen Wan
CSIRO Data61, Australia
Stephen.Wan@csiro.au

ABSTRACT

In this paper, we describe the 2017 CLAS system as entered into the C@merata shared task. This year, our aim was to use the challenge as a case study of how one manages natural language queries to structured data, and so we focused on the use of a NoSQL database for managing the retrieval of passages from musical scores. We also extended the 2015 CLAS system to handle queries about harmonies between specific parts, repetition of sequences, and thematic queries about sequences and imitation. Given that the queries were quite diverse, we explored the use of paraphrase methods to transform queries into a canonical form, where possible, which might then be parsed using a feature-based CFG. Our system achieved a measure precision and recall of 0.122 and 0.26 respectively.

1 INTRODUCTION

In this paper, we describe the CLAS submission for the 2017 C@merata shared task [1]. As in previous years, the task is one where a system must find portions of a musical score that match a natural language query. For example, the query "two eighth notes, an eighth note rest, three eighth notes, an eighth note rest and three eighth notes in measures 68-80, all in the Violoncello" (example from the 2016 data set) might be used to identify a portion of the score, as specified by the starting and ending bar numbers as well as the beat offsets in a manner prescribed in [1], such as:

Passage

- end_bar="79"
- end_beat_type="4"
- end_beats="4"
- end_divisions="1"
- end_offset="1"
- start_bar="78"
- start_beat_type="4"
- start_beats="4"
- start_divisions="1"
- start_offset="1"

Our general approach in 2017 is consistent with the earlier CLAS submissions in 2014 [2] and 2015 [3], which viewed the task as a Q&A problem with natural language queries posed in a controlled language. We view the task as a natural language query task to structured data, where the data has a temporal element and can be decomposed into multiple aligned streams of data. In terms of music, these streams correspond to different musical instruments or parts, temporally aligned. This year, however, we focused on using the shared task as a case study on natural language queries to structured data using database technologies.

The 2017 submission builds on the 2015 CLAS entry, which uses a feature-based Context-Free Grammar (CFG) to specify the controlled language for C@merata music queries. Parsing using NLTK [4] provides a feature structure corresponding to the key semantic elements of the query which is then used to retrieve results. In the 2015 CLAS system, the feature structure was used to find the matching events in the musical score. The Music21 [5] library was used to transform the XML version of a score into an array of music events, each represented with a set of attribute-value features. Events were then retrieved for a query, using feature unification between the query feature structure and the features of events.

This year, we varied our approach in the following ways:

1. Instead of making sequential passes through the music events for a piece of music, we used the nosql database (MongoDB) to store and retrieve musical events using mongoDB queries based on attribute-value sets.

2. We enhanced the 2015 feature-based CFG to recognise queries about:
    a. multiple parts
    b. repetition
    c. sequences and themes

3. We used a paraphrase or near-paraphrase back-off stage if the original query could not be parsed

Our system focuses on the queries based on music theory as opposed to those that would rely on music interpretation. In preparation for the 2017 shared task, we used the gold standard data from 2014-2016 as software development regression tests. This year, our CLAS system was able to achieve a measure precision and recall of 0.122 and 0.26 respectively.

MediaEval'17, September 2017, Dublin, Ireland

3 DATA STORAGE

3.1 Indexing with MongoDB

Instead of making a single pass of a music score (iterating through all possible notes) to find matching events using feature unification, we used a database to store and retrieve all music events.

Specifically, we used four tables:

1. Titles (and global score attributes)
2. Musical Events, one table per musical score
3. Sequences, one table per musical score
4. Analysis, one table per musical score

In the Titles (and global score attributes) database table, information mapping from the XML music score filename to an internal ID was kept, in addition to global information (determined using Music21 functions) such as the time signature, key signature, and the number parts. Examples of such records are presented in Figure 1.


```
{
   "_id":ObjectId("598ddda11d41c896ba36332b"),
   "name":"air_from_handels_water_music_suite.xml",
   "time_signature":{
      "1":[
         4,
         4
      ]
   },
   "key":"F major",
   "parts":4
}
{
   "_id":ObjectId("598dde0b1d41c896ba36364e"),
   "name":"and_the_glory_of_the_lord_from_handels_messiah.xml",
   "time_signature":{
      "1":[
         3,
         4
      ]
   },
   "key":"E major",
   "parts":18
}
```


Figure 1: Example JSON records from the Titles table.

For each musical score, we dynamically created a Music Events table. The table name is the unique identifier specified in the Titles table. The table for a score stores music events which could either be of a note or a chord type. Events contain offsets and other attributes. For note events, for a score consisting of multiple parts, the notes of each part were read sequentially to form the note events. For chord events, each part was also transformed into a series of Music21 Chord objects using the corresponding function in the Music21 library. Metadata for each chord was then stored in the table. Finally, the same functionality for creating chords at each offset was performed on the entire score, and then this list of chord objects was indexed. An example of a note record is presented in Figure 2 and a chord record in Figure 3.

For some queries, chords (or harmonies) across specific parts were required. To avoid computing every single permutation of parts, the specific combination was checked for at query time, and if it did not exist, the relevant parts were extracted from the XML, merged into a temporary score and then the entire score "chordified", and the results indexed dynamically. Querying then resumed as above.

The Sequences table for a score stores passages or sequences detected in the music. These were found by segmenting series of notes using rests, using a maximum sequence length of 8 bars.


```
{
  "_id":ObjectId("598ddda11d41c896ba36332c"),
  "name":"A",
  "letter":"A",
  "accidental":"",
  "pitch_class":9,
  "octave":4,
  "bar":1,
  "offset":0,
  "length":0.75,
  "part":0,
  "lyric":null,
  "freq":440,
  "articulation":[
   "DOWNBOW","BOWING","TECHNICALINDICATION","ARTICULATION","MUSIC21OBJECT","OBJECT"
  ],
  "expression":null,
  "solfeg":"",
  "type":"note",
  "dynamic":"piano",
  "tie":"",
  "slur":"1",
  "voice":"VIOLIN I",
  "clef":"G",
  "voice_num":"0",
  "stream":"notes",
  "ordering":0
}
```


Figure 2: Example of a note JSON record from the Music Events table for "`air_from_handels_water_music_suite.xml`" which in this case was stored in table "db.T598ddda11d41c896ba36332b" (notice the identifier following the "T" is the same as in the first record of Figure 1).

```
{
   "_id":ObjectId("598ddda11d41c896ba36348a"),
   "name":"C3-dominant seventh chord",
   "bar":3,
   "offset":2,
   "length":0.75,
   "key":"F major",
   "chord_fn":"5",
   "chord_match":3,
   "inversion":0,
   "bass":"C3",
   "root":"C3",
   "notes":[
      "B-",
      "E",
      "G",
      "C"
   ],
   "notes_with_octave":[
      "B-4",
      "E4",
      "G3",
      "C3"
   ],
   "ordered_pitch_classes":[
      0,
      4,
      7,
      10
   ],
   "root_fn":"5",
   "function":"5",
   "type":"dominant seventh chord",
   "raw_type":"dominant seventh chord",
   "ties":4,
   "passing":true,
   "dynamic":null,
   "intervals":[
      0,
      2,
      3,
      4,
      5,
      6
   ],
   "interval_names":[
      "Minor Tenth",
      "Perfect Fifth",
      "Major Sixth",
      "Perfect Unison",
      "Major Tenth",
      "Diminished Fifth",
      "Minor Fourteenth"
   ],
   "stream":"chords",
   "ordering":19
}
```


Figure 3: A chord JSON record from the Music Events table for "`air_from_handels_water_music_suite.xml`".


```
{
  "_id":ObjectId("598de46b1d41c896ba374f8f"),
  "name":"ABS::++E-_3.0000_0.0000++D_1.0000_0.0000++
F_1.0000_1.0000++D_1.0000_2.0000++C_1.0000_0.0000++E-
_1.0000_1.0000++C_1.0000_2.0000++B-_1.0000_0.0000",
  "key_type":"absolute",
  "seen_parts":[
    "FLUTE 1 2",
    "BASSOON 1 2"
  ],
  "motif_length":8
}
{
  "_id":ObjectId("598de46b1d41c896ba374f90"),
  "name":"DIA::++-2_1.0000_0.0000++3_1.0000_1.0000++
-3_1.0000_2.0000++-2_1.0000_0.0000++3_1.0000_1.0000++
-3_1.0000_2.0000++-2_1.0000_0.0000",
  "key_type":"diatonic",
  "seen_parts":[
    "FLUTE 1 2",
    "BASSOON 1 2"
  ],
  "motif_length":8
}
```


Figure 4: A sequence in both its absolute note and diatonic interval forms that occurs in two parts, here Flute and Bassoon from "beethoven_symphony_3_movement_iii_muse. xml".


```
{
  "_id":ObjectId("598de4691d41c896ba3748f2"),
  "name":"ABS::++E-_3.0000_0.0000++D_1.0000_0.0000++
F_1.0000_1.0000++D_1.0000_2.0000++C_1.0000_0.0000++E-
_1.0000_1.0000++C_1.0000_2.0000++B-_1.0000_0.0000",
  "voice":"FLUTE 1 2",
  "clef":"G",
  "voice_num":"0",
  "key_type":"ABS",
  "type":"motif",
  "start_bar":26,
  "start_offset":0,
  "end_bar":29,
  "end_offset":1,
  "end_duration":2,
  "note_length":8,
  "duration":10
}
```


Figure 5: Metadata for the sequence with id=598de46b1d41c896ba374f8f in the score "beethoven_ symphony_3_movement_iii_muse.xml".

Sequences were represented with unique identifiers representing the entirety of the passage, using each of three possible key generators: the unique names of the notes in the

sequence and their lengths (but not their octaves offsets), a variant of this using relative displacement between notes in terms of semitones, and a diatonic variant using interval classes. Each sequence had an associated start and end point (specified as bar and offset), which was stored in the table.

The Analysis table for each score keeps track of which sequence unique identifiers occur in different parts of the score, allowing some representation of thematic sequences (those that appear in multiple parts). An example of a sequence from the Analysis table is presented in Figure 4 and the corresponding metadata from the Sequence table is presented in Figure 5.

3.2 Querying the Database

Each record in the table (aside from the Titles table) is an event represented as a set of attribute-value features. Using the Python programming language and the PyMongo[1] library which provides an interface to mongoDB[2], queries are essentially comparisons between dictionary objects. In this way, the queries are very similar to the feature unification method in the 2015 CLAS system. There is a subtle difference however. Feature unification would allow under-specifications on either of the two structures being compared to unify as long as there were no direct contradictions. This is not the case with mongoDB queries. If a query is overly specific and includes attributes not present in the event, then a match is not possible. Thus, each feature structure first had to be transformed into an equivalent NoSQL query to account for this.

Querying for sequences of notes was performed by first using a query for the first note of the sequence and then performing an ordered series of queries, each checking if the event at the next timestep corresponded to the relevant sequence note. This was handled through a recursive function that worked its way down a list of elements in the query.

Using mongoDB to provide search facilities greatly increased the speed at which matches could be found, utilizing a database index for direct lookup instead of doing a sequentially pass of the score for each query.

3.3 Forming NoSQL Queries

If we are searching for a single note, then the query would be the set of attribute-value features for that note as represented by a Python dictionary. Such queries were simple to implement with a NoSQL database like mongoDB. Others, however were more complex.

For melodic intervals, these are treated as sequences. In this case, for every note in each part, we search for the absolute pitch and octave of a note that would correspond to the interval based on the current note. Thus for melodic intervals, "search" becomes a single pass through the score.

For harmonic intervals, we search through the chord events, which at the time of indexing is broken down into its component interval classes between all note pairs in the chord. Each of these intervals is stored, at indexing time, as a list (with an equivalence encoded for augmented fourths, diminished fifths, and tritones). The mongoDB query is then a list membership test for the string name of the interval of interest. We note that the analysis of the chord into its component intervals is provided by Music21 and the success of this method depends on that library's ability to correctly analyse the chord. For harmonic intervals and chords, we added search constraints like there needing to be at least 2 notes in the chord event.

For chord queries specifying the exact notes of the chord, this is implemented using tests for membership of the notes in the chord event.

Cadences are treated as sequences of chord events. Tests are based on the features of adjacent chords such as its chord function (as a roman numeral) and the notes one would expect to see (derived from the chord function).

Whenever string matches are required in the mongoDB query, these are implemented as regular expressions to allow "fourth" to match to "perfect fourth", or "Violin" to match to "Violin 1".

4 QUERY EXTENSIONS

In this system, we extended the feature-based CFG and the query generation components to cater to three new types of queries:

1. Harmonies between multiple parts
2. repetition
3. sequences and themes

4.1 Harmonies between Multiple Parts

The extension for queries indicating multiple parts specifically relates to harmonies, such as "minor third between Quintus and Tenor in bars 11-18". In this case, the part names "Quintus" and "Tenor" are mapped to a generic symbol "PART_NAME", which can be resolved later to the original values. Special grammar rules for part names in this configuration ("between *part1* and *part 2*") are used to keep track of semantic features that flag that processing of the query should use logic relating to a harmony between parts. When such flags are encountered during resolution of the query, the system first checks to see if the required combination of parts has been indexed in the database. If not, this then triggers the dynamic indexing of chords in the specified parts (as mentioned above).

We note that queries such as "B4 in the left hand followed by C5 in the right hand in bars 7-8" are covered by a mechanism from the 2015 system in which the query is divided into two portions (split by the phrase "followed by"). Each is an atomic query that could be found in different parts (here, left hand versus right hand, assuming a simple mapping from left to bass clef and right to treble clef).

4.2 Repetition

Queries like "six crotchet notes repeated twice" are treated as a sequence of "six crotchet notes" which is then copied a second time. In this system, handling of repetition requires that the

[1] https://api.mongodb.com/python/current/
[2] www.mongodb.com

number of copies be specified. That is an unbounded repetition query, such as "alternating fourths and fifths in the Oboe in bars 1-100" is not handled. To allow such queries, the system arbitrarily transforms such as query to be equivalent to "repeated twice". This copying of sequences extends the 2015 system which allowed copies of single notes to be specified in the query.

4.3 Sequences and Themes

To make use of the sequence and analysis tables described above, we extend our grammar to accept relevant queries, focusing on repeated sequences or imitation between parts. For such queries to be resolved, we search the sequence and analysis tables outlined in Section 3. The analysis table shows when a sequence has been repeated verbatim, with the same note names (but allowing variation in the octave), or transposition with relative intervals between sequence notes, specified in both a diatonic and chromatic form.

The analysis table shows when any of these sequences has been repeated between parts, and includes metadata that allows trivial sequences below a certain length to be filtered out. Once repeated sequences are found, the start and end offsets are retrieved from the relevant sequence table.

5 SIMPLE PARAPHRASE AND CONSTRAINT RELAXATION

In this system, we introduced a new method to allow graceful degradation from the case where the grammar did not cover the input query. In such a case, a prioritised set of paraphrase rules was used to see if the query could be transformed into a form that was covered by the grammar.

These paraphrase rules included synonyms, such as the mapping from "Violin 1" and "Vln 1", and phrasal equivalents, such as "from bars X-Y" and "in bars X-Y".

The rules also included "movement" of certain phrases such that occurred at set positions. For example, constraints about the bars ("from bar X-Y") was moved to the end, as was expected by the grammar.

We can thus think of the grammar covering the canonical form of a query. For example, constraints about multiple parts in queries such as "cello and viola playing dotted minims an octave apart in bars 40-70" was mapped to "dotted minim octave between cello and viola in bars 40-70". Similarly, for repetition in queries such as "repeated Bb4 whole note" was transformed into a canonical form like: <note noun phrase><repetition constraint>.

Finally, we encoded near paraphrases to handle concepts that were similar to those already covered by the grammar. For example, we used a mapping from "ascending in single steps" to "ascending". For some queries, the additional information was deemed to be of little value (given the default behaviour of the system) and was thus dropped. For example, "in a row" was simply deleted given that most sequence queries require that the target notes occur in series.

Ideally, these rules would be learnt from data. We see this a future work. Here we simply provide a way for paraphrase rules, once acquired, to be used however they are obtained (by manual inspection or machine learning).

6 RESULTS AND ANALYSIS

Our overall approach for preparing this year's submission was to use the gold standard results as regression tests, specifically using the 2016 test set for the development of new features. The recall and precision at the beat level for the three preceding years of data is shown in **Table 1**. While, this is the result of development on the 2014-2016 test sets, the performance is based on generic mechanisms without memorisation of answers. We note that these score for 2014 and 2015 are slightly under the official report performance of the CLAS system for those years. This is due to system differences between the 2017 system and the earlier systems: the 2015 system was a blend of the 2014 system and the use of the feature-based CFG. For the 2017 system, the earlier chunking system of 2014 has been dropped altogether.

Table 1: Performance on data from prior years.

	Recall	Precision
2016	0.387	0.350
2015	0.565	0.541
2014	0.786	0.720

For the 2017 results, our system achieved the following scores:
Beat Precision: 0.099
Beat Recall: 0.212
Measure Precision: 0.122
Measure Recall: 0.260

Table 2 reports the results by question type. We note that the system is does best for harmony queries, followed by melodic queries, as shown by the recall scores. This is unsurprising given then the 2017 was developed with a focus on queries related to music theory. For the more complex texture and synch queries, the system does not do so well.

Although a system for sequences and themes was included this year, we note that the method for segmenting sequences in this system is quite simplistic and may not correspond to analyses by a musicologist. Furthermore, the grammar coverage for these queries was not exhaustive.

Table 2: 2017 results per question type.

	Measure Recall	Measure Precision	Beat Recall	Beat Precision
1 Melod	0.475	0.062	0.328	0.043
N melod	0.213	0.176	0.167	0.137
1 harm	0.156	0.636	0.156	0.636
N harm	0.304	0.263	0.290	0.250
Texture	0.103	0.176	0.103	0.176
Follow	0.052	0.133	0.026	0.067
Synch	0.000	0.000	0.000	0.000

7 DISCUSSION

The 2017 system is based on the feature-based CFG system from 2015. We note that the queries have become more and more complex to the extent that one might question if treating the queries as coming from a controlled language is still viable. This may be one reason why the performance of the system has degraded compared to previous year's submissions.

From software engineering point of view, we note that the grammar for the queries is increasingly difficult to manage with each subsequent year's set of queries. Whether the paraphrase component is an effective way to manage this diversity of queries remains to be tested.

The move to the mongoDB search engine did introduce efficiencies in running large sets of queries from prior years as regression tests. Previously, the 2015 system would have required hours to complete given that each query required a pass through the score. Currently, each year's queries take a few minutes to answer. This would be faster if not for the passes needed through the scores for melodic queries. We note that melodic intervals too could be annotated upfront at indexing time, an optimization which we simply did not have time to implement.

However, some analyses require batch processing of the score to identify features of interest. For these, the analysis, potentially time consuming, has to be performed first ahead of database indexing.

8 FUTURE WORK

In future work, we intend to further extend the system to answer musicology related queries as opposed to just music theory queries. This will involve additional preprocessing of the scores in advance. For example, currently thematic sequences are currently delimited using rests. However, additional cues in the score, such as phrase boundaries and lyrics, could also help provide other possible segmentations.

9 CONCLUSIONS

We describe the CLAS system as entered into the 2017 C@merata shared task. In this system, we focused on the use of a NoSQL database for managing the retrieval of passages from musical scores. We also extended the 2015 CLAS system to handle queries about harmonies between specific parts, repetition of sequences, and thematic queries about sequences and imitation. We also explored the use of paraphrase methods to transform queries into a canonical form, where possible, which might then be parsed using a feature-based CFG.

ACKNOWLEDGMENTS

We thank the organisers of the C@merata 2017 challenge for their hard work and efforts in running the event.

REFERENCES

[1] Sutcliffe, R. F. E., Ó Maidín, D. S., Hovy, E. (2017). The C@merata task at MediaEval 2017: Natural Language Queries about Music, their JSON Representations, and Matching Passages in MusicXML Scores. Proceedings of the MediaEval 2017 Workshop, Trinity College Dublin, Ireland, September 13-15, 2017.

[2] Wan, S. (2014). The CLAS System at the MediaEval 2014 C@merata Task. In the Working Notes Proceedings of MediaEval 2014 Workshop. Barcelona, Catalunya, Spain, October 16-17, 2014, CEUR-WS.org

[3] Wan, S. (2015). The CLAS System at the MediaEval 2015 C@merata Task. In: Mediaeval 2015; 14-15 September 2015; Wurzen, Germany. Mediaeval; 2015. 1-6.

[4] Bird, S., Loper, E., Klein, E. (2009). Natural Language Processing with Python. O'Reilly Media Inc.

[5] Cuthbert, M., Ariza, C. (2010). music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. Proceedings of the International Symposium on Music Information Retrieval, pp. 637–42, 2010.