NEW FRAMEWORK FOR SCORE SEGMENTATION AND ANALYSIS IN OPENMUSIC

Jean Bresson
STMS: IRCAM-CNRS-UPMC, Paris, France
jean.bresson@ircam.fr

Carlos Pérez-Sancho
DLSI, Universidad de Alicante, Spain
cperez@dlsi.ua.es

ABSTRACT

We present new tools for the segmentation and analysis of musical scores in the OpenMusic computer-aided composition environment. A modular object-oriented framework enables the creation of segmentations on score objects and the implementation of automatic or semi-automatic analysis processes. The analyses can be performed and displayed thanks to customizable classes and callbacks. Concrete examples are given, in particular with the implementation of a semi-automatic harmonic analysis system and a framework for rhythmic transcription.

1. INTRODUCTION

Formalized musical models in the 20th Century introduced new ways of representing, understanding and analysing music [1, 2], which progressively and naturally led to computational approaches [3, 4]. Today, computation and computer-aided modelling make for a broad spectrum of possibilities in music description and analysis.

The term *computational analysis* currently spreads on diverse domains and purposes. In the "musicological" domain [5], computer systems and programs are used to assist music analysts in improving their work with new representational and computational possibilities (see for instance [6], [7] or [8]). More related to the *music information retrieval* domain is the extraction of high-level information or knowledge from scores or audio recordings, such as musical genre classification [9], chord extraction from audio [10], segmentation and phrase boundary detection [11], automatic (melodic and harmonic) analysis using machine learning, pattern recognition algorithms [12] or even cognitive approaches [13]. In these different domains and applications, computational aspects can be either fully automated or (more interestingly) integrate user interaction and guidance to the analysis process.

Aside to computational approaches, and somehow in a complementary way, other systems exist which focus on the visualisation of musical scores and analytical structures [14], but they principally rely on external preliminary analyses. Present computer systems offer useful tools for extended graphical and auditory rendering, and enable new interactions with the analysis models and processes.

Following seminal works by M. Mesnage and A. Riotte [15], different projects aimed at integrating formalized analysis in computer-aided composition software. In particular, previous research such as [3, 16, 17], as well as the "computer-assisted music analysis" project in PWGL [18, 19], paved the way to the recent extensions of the OpenMusic computer-aided composition environment (OM [20]) which we present in this paper.

2. SEGMENTATION AND GENERAL CONCEPTS

Music theory and analysis generally require and rely on preliminary segmentation of the musical material. In many cases actually, the segmentation itself is a crucial aspect of the analysis process.

Several theoretical frameworks have been proposed so far focusing on music segmentation [21, 22], as well as implementations of automatic segmentation systems [17, 23].

Depending on the analysis or on the musical material at hand, the concept of *segment* can be devised in very distinct ways. A segment can for instance be determined by a time interval, by a set of (adjacent or scattered) notes, chords, measures, sections, or even by a spectral region. Segments are therefore highly dependent on the analysis, but also on subjective choices from the analyst [21]. Hence, we try to be as generic as possible and from here on will consider as a segment any part of a score or musical structure, be it defined in time, by a set of score elements, or even by a structural or functional specification.

We also consider a simplified representation of the *analysis* concept as a set of segments related to a musical extract and containing specific information. From this point of view, an analysis is an augmented segmentation where each segment is enriched with more or less complex data related to the corresponding score contents (and segmentation itself can be seen as a particular analysis with no such additional data).

3. FRAMEWORK ARCHITECTURE

OpenMusic musical structures are organized as an object-oriented architecture [1] which has been extended to integrate segments and analyses (see Figure 1).

[1] These musical structures include all types of "score objects" in OM, such as *chord-seq*, *voice*, *poly*, etc. They can be created in the OM visual programming framework, or imported (and possibly processed/converted), e.g. from MIDI or MusicXML files.

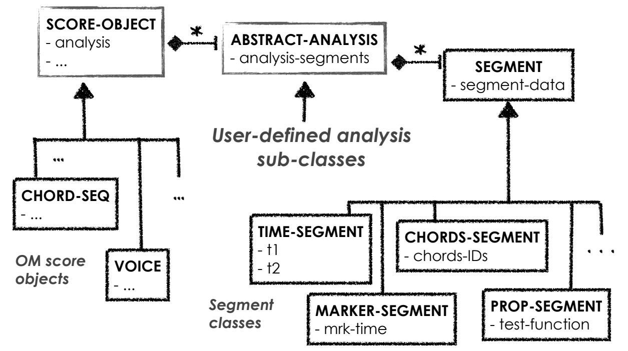


Figure 1. Class architecture for segmentation and analysis attached to the OpenMusic objects framework.

In this object-oriented framework, the analysis (or segmentation) is also a class. We define the general abstract class *abstract-analysis*, of which any analysis or segmentation implemented in the environment should be a subclass.

All musical objects now have a new attribute (or *slot*) called *analysis* containing a list of *abstract-analysis* instances. As stated in the previous section, we mostly consider an analysis as a set of more or less specific and informative segments: the main slot of *abstract-analysis*, labelled *analysis-segments* and inherited by all analysis subclasses, therefore contains a list of segments. Of course, subclasses of *abstract-analysis* can define new attributes and behaviours, for instance to determine relations between the segments. Note that the *analysis* slot value is a list, so that multiple segmentations can be attached to a score object, making for several interpretations or several types of analyses of a same extract potentially coexist.

Following the previous scheme, segment classes as well are all subclasses of a unique predefined class called *segment*. The main slot of this class is called *segment-data*. It is an open entry allowing the different analysis classes to store data in the segments. Some predefined segment types have been defined, extensible if needed to encompass particular segmentation or analysis needs: *time-segment* is defined as a simple time interval by two additional attributes *t1* and *t2*; *marker-segment* is even simpler and contains a single temporal landmark *mrk-time*; *chords-segment* consists of an ordered set of chord indices corresponding to chords in the analyzed sequence; *notes-segment* is defined by a set of notes from this sequence (possibly taken from different chords and with no specific order); etc.

As we will show further on, each type and subtype of segment can have specific behaviours and particularities at the time of computing analysis data, drawing in the score editor or during user-interactions. Interesting personalized segment classes would also include functional (or "structural") specification determining an implicit selection of particular regions in the score. [2] In Figure 1, *prop-segment* is a prototype class defining segments determined by a given property (*test-function*): each score component verifying the corresponding property (tested by this function) would implicitly be considered belonging to the segment.

[2] This is one of the main features of the system proposed and implemented in [18] using a pattern matching syntax and a constraint-based system.

4. PROTOCOL AND INTERACTION

In order to create a new type of segmentation or analysis, one must define a subclass of *abstract-analysis*. In the Appendix of this article is listed the Common Lisp code for the definition of an example analysis class (called *my-analysis*), which will implement most of the features described below.

First, it is possible to specialize the analysis class for specific types of score objects (e.g. *voice*, *chord-seq*...). Instances of the analysis can then be attached to a compatible score object, either via (visual) programming, or using new commands available in the score editor (see Figure 2). A new editing mode in the OM score editors enables all the segmentation and analysis features.

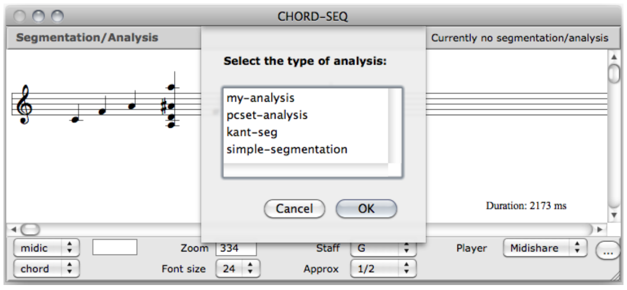


Figure 2. Attaching an analysis class to a *chord-seq* in the editor. The analysis types list presents all compatible subclasses of *abstract-analysis*.

Different analyses and segmentations may coexist on a same object, but one will always be at the "front" (*current segmentation*) and handle drawing and user interaction. In the editor, a keyboard short-cut allows to switch between the different segmentations/analyses attached to the object.

Several options and methods can be defined in order to specify or specialize the behaviour and interface of the analysis. Analyses can perform either segmentation, analysis, or both in a same run. Depending on the implemented features, the corresponding options will be proposed in the OM editor menus (see Figure 3).

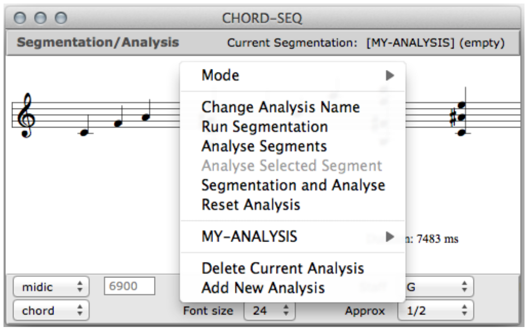


Figure 3. New actions available after attaching an analysis instance to a score object.

In Figure 3 are also visible other general features to handle the analyses (reset, remove, add new analysis, etc.) Additional actions can be defined for the analysis class using the *get-analysis-menu-items* method redefinition (see the sample code in the Appendix, ll. 82-85).

4.1 Segmentation

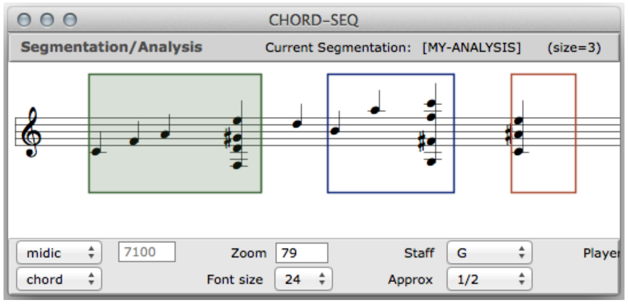
The segmentation phase can be either automatic or performed interactively by the user. If the method *compute-analysis-segment* is defined for an analysis class, the corresponding action can be made available in the score editor. The only requirement of this method is to return a list of segments (that is, instances of any subclass of *segment*).

The example analysis implemented in the Appendix, for instance, performs segmentation by returning a segment of type *chords-segment* for every single chord in the score (ll. 28-34).

Interactive segmentation or modifications of existing/computed segmentations depend on the analysis types. In order to control and personalize this process, the main user action and callbacks (e.g. "command-click", keyboard actions, etc.) can be redefined for the different analysis classes and generate or modify segments from the musical object at hand and relevant corresponding callback parameters (mouse position, current selection, etc.)

A number of actions related to the segmentation, such as segments display, selection, or deletion, can also be implemented in specific ways for the different analyses. [3]

In order to facilitate the implementation of analyses, however, a set of behaviours are already defined for the main existing segment types. The analysis classes can therefore be attached to a segment type and just rely on its default behaviours. In our example (see Appendix, l. 22) the analysis segments type is specified to be *chords-segment*. The default *chords-segment* behaviours and interactions will therefore apply: for instance, pressing key '*s*' (for "segment") with a number of chords selected in the score at this moment will create and add to the analysis a new segment containing these chords. In Figure 4 several *chords-segments* have been created in *my-analysis*.



CHORD–SEQ

Segmentation/Analysis   Current Segmentation: [MY-ANALYSIS] (size=3)

midic   7100   Zoom 79   Staff G   Player

chord   Font size 24   Approx 1/2

Figure 4. Creation of *chord-segments* in *my-analysis*.

Note that predefined actions themselves can be specialized for specific segment and/or analysis types. The mouse click default behaviour, for instance, is a test performed on the different segments of an analysis in order to determine if one (or several) of them is/are to get selected: in this case, the test can be adapted to a segment or analysis by overloading the method *segment-clicked-p* which determines whether a segment shall be selected when the editor window is clicked at a given position.

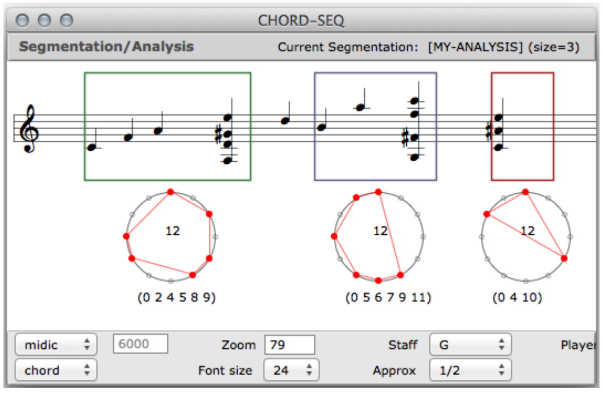
[3] The multiple dispatch provided by the Common Lisp Object System allows to specialize the callback methods with all of their arguments, hence generally here for both the segment and analysis types.

4.2 Analysis

The second important step after segmentation is the analysis of the different segments. As described previously, the *segment* class includes a data store (*segment-data*) to be used freely by the different analyses. Hence, we basically consider that the analysis consists in computing and attaching some data to the different segments. The nature and computation of this data is totally open and is actually what mostly differentiates analyses from one another.

The analysis classes can redefine or hook this process at different levels, and let the default behaviours operate for the rest. The main and easier way to go for that is to overload the method *analyse-one-segment*, meant to generate the analysis data for every segment. An example is given in the Appendix with our analysis class (ll. 40-53).

Another important point is the display of the analysis data in the different segment. Here again, redefining the *draw-segment-data* method allows every type of analysis to provide the adequate visualization depending on what has been stored in the segment's *analysis-data*. Figure 5 shows the result of *my-analysis*, as implemented in the Appendix (ll. 55-69). An object of type *n-cercle* [24] has been computed, attached and is displayed below every segment. [4]



CHORD–SEQ

Segmentation/Analysis   Current Segmentation: [MY-ANALYSIS] (size=3)

midic   6000   Zoom 79   Staff G   Player

chord   Font size 24   Approx 1/2

Figure 5. Displaying the segments data in *my-analysis*.

In some cases, the segmentation and analysis steps are intertwined in the analysis process, making it impossible to differentiate them in a two-steps process. [5] Our framework allows to accommodate such procedure and proposes adapted commands and callbacks, so that the implemented algorithms can return a set of analysed segments as a result of a unique processing phase. In this case, analyses can just redefine the method *compute-and-analyse-segments* and implement a global analysis process generating segments and data from the whole score.

5. APPLICATIONS

In this section we present additional examples of analyses implemented in the presented framework, making more advanced uses of the interactions and callbacks.

[4] *my-analysis* is actually a simplified model of pitch class set analysis.
[5] This is the case of the harmonic analysis described in section 5.1.

5.1 Supervised Harmonic Analysis

The *harmonic-analysis* project is an interactive tool based on the analysis model by Pardo and Birmingham [25], which has been implemented in Java as a MIDI file analysis server application, and integrated in OM using the framework described in this paper. [6]

5.1.1 *Analysis Principles*

The *harmonic-analysis* algorithm performs an automatic segmentation of the analysed sequence and proposes a set of weighted possibilities for the chord label of each segment (this is an example where analysis and segmentation of the score are considered and performed as a single step). First, the whole melody is divided into a set of minimal segments delimited by note onsets, which are then grouped considering all possible combinations. For each possible segment, all the chords in the analysis vocabulary are scored using a rule-based scoring function, and finally the segmentation with highest score is selected as the output of the algorithm following a graph search strategy.

For computational issues, at the current stage of this project only the greedy version of the algorithm (*HarmAn* search heuristic) has been implemented, which does not consider the whole search space, while keeping a chord recognition rate very close to the complete algorithm.

5.1.2 *Interaction*

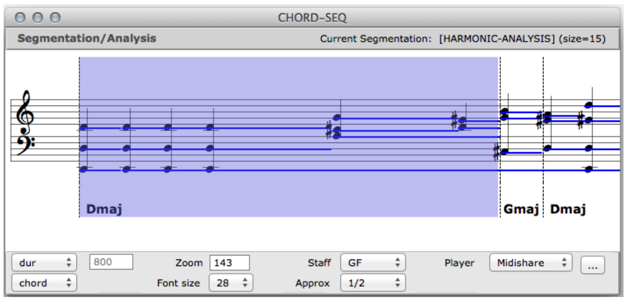
Once the analysis is performed the user can "validate" the successive segments or correct them. The correction of a segment can consist either in choosing another chord label than the top-weighted proposed solution for this segment, or in changing its boundaries (hence a correction at the level of the segmentation itself).

The correction and validation process is performed by the user following a left-to-right approach. Every correction made triggers a re-computation of the analysis from that segment on, and the validation of a segment implies that the user considers that all previous segments are also correct.

5.1.3 *Integration in the OpenMusic Framework*

The harmonic analysis software communicates with OpenMusic via OSC messages following a simple client/server protocol. When the *harmonic-analysis* class is attached to a *chord-seq* object in OM, a MIDI representation of the sequence is sent to the analysis server. The only operation available at this stage is the "global" segmentation *and* analysis: following the process described above, these two operations are interleaved and performed together by the analysis system.

The segment type associated to *harmonic-analysis* is a subtype of *time-segments*, hence defined by absolute begin and end times (*t1* and *t2*) but additionally constrained with the implicit time segments in the score determined by the note begin and end times. The segments are instantiated by the segmentation/analysis process and contain as *segment-data* a list of harmonic proposals sorted by weight. The first element of this list (therefore, the selected chord label) is displayed with the segment in the score editor (see Figure 6).

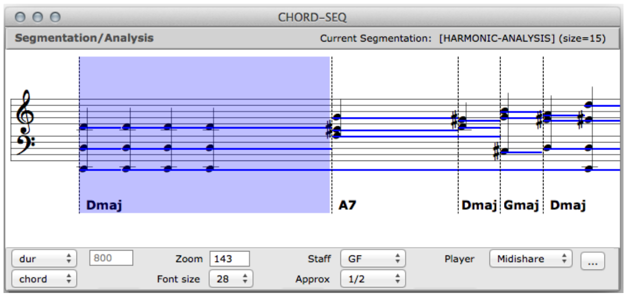


Current Segmentation: [HARMONIC-ANALYSIS] (size=15)
800 Zoom 143 Staff GF
Font size 28 Approx 1/2

Figure 6. Displaying the segments and analysis data in *harmonic-analysis*. Musical excerpt taken from the *String Quartet Op. 20 No. 4, I*, by Haydn.

A specific feature of the *harmonic-analysis* system is the left-to-right process of validation and correction of the segments analysis. As shown in Figure 6, *harmonic-analysis* actually complements the segment display by writing the proposed chord label, but also by colouring the segments. The "current segment" is the next segment to be validated in the sequence: it is coloured in blue.

Several callbacks of the score editor are caught by *harmonic-analysis*, and allow to modify its contents or properties before validating it. The segment length can be modified using the "click-and-drag" action, which allows to specify the new end of the current segment and position it at a different note in the sequence. This action forces the analysis server to recompute the weights for the current segment, as well as the complete analysis for the remaining part of the score (see Figure 7).



Current Segmentation: [HARMONIC-ANALYSIS] (size=15)
800 Zoom 143 Staff GF
Font size 28 Approx 1/2

Figure 7. New analysis after changing the first segment length.

The double-click action validates the segment (which will then be coloured in green) and sets the "current segment" pointer to the next one in the sequence (see Figure 8).

[6] This project is available for download and evaluation under an open source license at `http://grfia.dlsi.ua.es/cm/projects/drims/software.php`

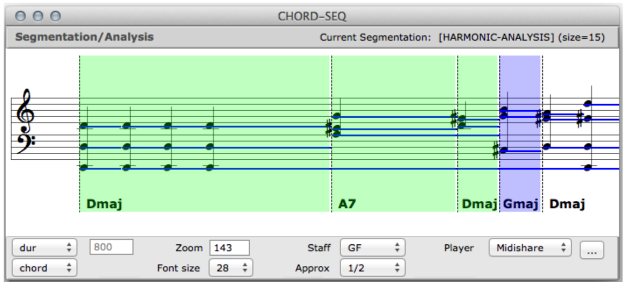


Figure 8. Analysis after validating the first three segments.

Figure 9 shows the segment data being changed using a contextual menu presenting to the user the list of secondary solutions found by the analysis system (also stored, though not visible to the user, in the segment's *analysis-data*), as well as additional items allowing to choose among the full set of possible chord labels.

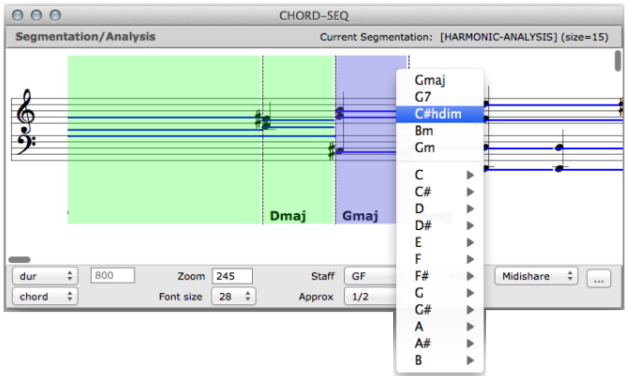


Figure 9. Modifing the chord label of a segment.

All the user modifications and actions result in notifications to the analysis server and eventual corresponding updates. In future works we plan to take into account these interactions to build an informed knowledge base allowing to improve subsequent analysis results.

5.2 Interface for Rhythmic Quantification

A second foreseen application for the presented framework is to improve the processes of rhythmic quantification performed in OpenMusic [26]. Rhythmic quantification (also sometimes referred to as "quantization") basically consist in converting a flow of temporal onsets (or inter-onset durations) into a rhythmic structure (including tempo, meter and pulse subdivisions). It is a major issue in computer music and computer-aided composition, for which efficient solutions are rare [27].

5.2.1 *Quantification as a Special Case of Analysis*

In a sense, rhythmic quantification can be considered as a form of analysis of a musical sequence, for which supervised or subjective inputs may enable more flexibility and will generally optimize the results. [7]

[7] Apart from the durations, we can also imagine that the rest of the score contents (that is, its other attributes such as pitches and velocities) can help the segmentation and analysis processes to deduce rhythmic information.

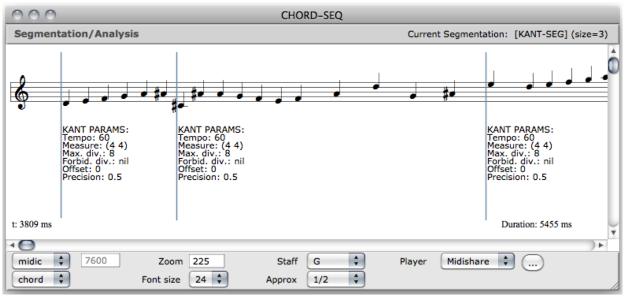
A typical problem in rhythmic quantification is the accumulation of delays and errors due to temporal shifts, non-constant tempo or approximations. From this perspective, for instance, a preliminary (and wise) segmentation of the sequence may ensure the rhythmic consistency inside successive blocks, and avoid possible errors and shifts to propagate in the score analysis.

The *OMKant* library for OpenMusic [8] allowed to perform such user-supervised segmentation as well as beat and meter detection in order to improve the quantification processes. This library has not been maintained and ported to the recent OpenMusic versions; however, we believe that the present segmentation framework may provide an interesting way to generalize and extend the principles of this project.

5.2.2 *Implementation*

We present here a preliminary implementation based on the current quantification tools available in OpenMusic. The *kant-seg* analysis class uses a new type of segment called *chord-marker*, which is attached to a chord and bounds implicitly the rest of the sequence until the next *chord-marker*. This type of segment, instantiated by the user on the same model as for the *chord-segments* in Section 4.1, allows to "slice" the whole sequence with no gaps nor overlapping intervals (every segment starts at a chord in the score, and its end is implicitly defined as the beginning of the next segment).

The analysis also uses another callback provided by our framework at initialization of the segments, and attaches them a specific structure, stored in the *segment-data*, containing all the parameters required to perform the quantification (estimated tempo, meter, constraints on pulse subdivisions, etc.) Figure 10 shows how these data are displayed in the score editor by the *kant-seg* analysis class.



KANT PARAMS:
Tempo: 60
Measure: (4 4)
Max. div.: 8
Forbid. div.: nil
Offset: 0
Precision: 0.5

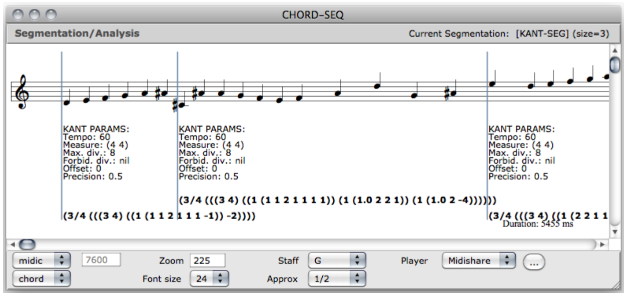
KANT PARAMS:
Tempo: 60
Measure: (4 4)
Max. div.: 8
Forbid. div.: nil
Offset: 0
Precision: 0.5

KANT PARAMS:
Tempo: 60
Measure: (4 4)
Max. div.: 8
Forbid. div.: nil
Offset: 0
Precision: 0.5

Figure 10. Segmentation for rhythmic quantification. With the *kant-seg* analysis class, every *chord-marker* segment is initialized with default quantification parameters.

The analysis process in this case consists in computing a rhythm tree [28] starting from the durations in each individual segment, using the OM quantification function and the parameters stored in these different segments. The resulting tree is also stored in the segments' *segment-data* and eventually displayed in the score editor (see Figure 11).

[8] *OMKant* by Benoît Meudic, IRCAM.



Current Segmentation: [KANT-SEG] (size=3)

```
KANT PARAMS:
Tempo: 60
Measure: (4 4)
Max. div.: 8
Forbid. div.: nil
Offset: 0
Precision: 0.5
```

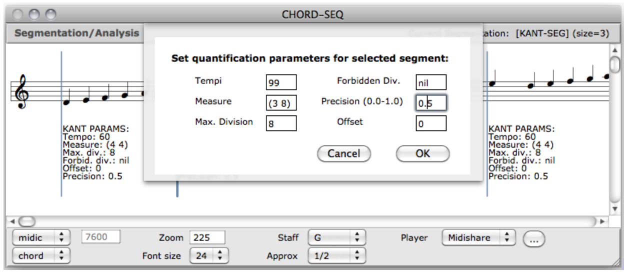
```
(3/4 (((3 4) ((1 (1 1 2 1 1 1 -1)) -2))))
(3/4 (((3 4) ((1 (1 1 2 1 1 1 1)) (1 (1.0 2 2 1)) (1 (1.0 2 -4))))))
(3/4 (((3 4) ((1 (2 2 1 1
Duration: 5455 ms
```

7600 · Zoom 225 · Font size 24 · Staff G · Approx 1/2 · Player Midishare

Figure 11. Analysis (quantification) of the *kant-seg* segments from Figure 10.

5.2.3 Interaction

The *kant-seg* analysis implements a special action at double-clicking a segment, which opens a dialog window (see Figure 12) allowing to set the quantification parameters for this segment before running the analysis.



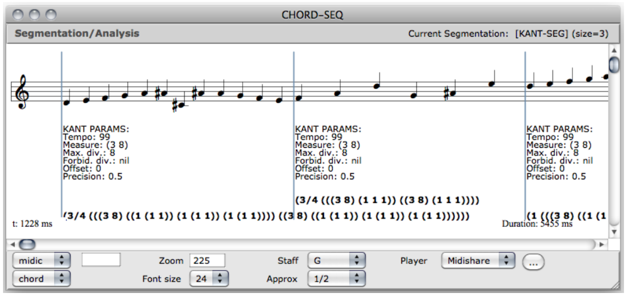
Set quantification parameters for selected segment:

Tempi	99	Forbidden Div.	nil
Measure	(3 8)	Precision (0.0-1.0)	0.5
Max. Division	8	Offset	0

```
KANT PARAMS:
Tempo: 60
Measure: (4 4)
Max. div.: 8
Forbid. div.: nil
Offset: 0
Precision: 0.5
```

Figure 12. Setting quantification parameters.

Figure 13 shows how setting segments and parameters appropriately allows to get better results in the quantification process (also note that the segment bounds have themselves also been modified as compared to Figure 11).



Current Segmentation: [KANT-SEG] (size=3)

```
KANT PARAMS:
Tempo: 99
Measure: (3 8)
Max. div.: 8
Forbid. div.: nil
Offset: 0
Precision: 0.5
```

```
(3/4 (((3 8) ((1 (1 1)) (1 (1 1)) (1 (1 1)))) ((3 8) ((1 (1 1)) (1 (1 1)) (1 (1 1))))))
(3/4 (((3 8) (1 1 1)) ((3 8) (1 1 1))))
(1 (((3 8) ((1 (1
t: 1228 ms
Duration: 5455 ms
```

Zoom 225 · Font size 24 · Staff G · Approx 1/2 · Player Midishare

Figure 13. Analysis (quantification) of the new *kant-seg* segments.

The quantified data (or any contents of the analysis segments) can eventually be extracted from the score objects. In this case, a quantified score can be obtained by collecting and concatenating the voices corresponding to the successive segments' contents and quantification results (stored in *segment-data*—see Figure 14). In Figure 14 the voices resulting from our two previous analyses are extracted, and we can observe the difference in the quantification results depending on the segmentation and assigned parameters.

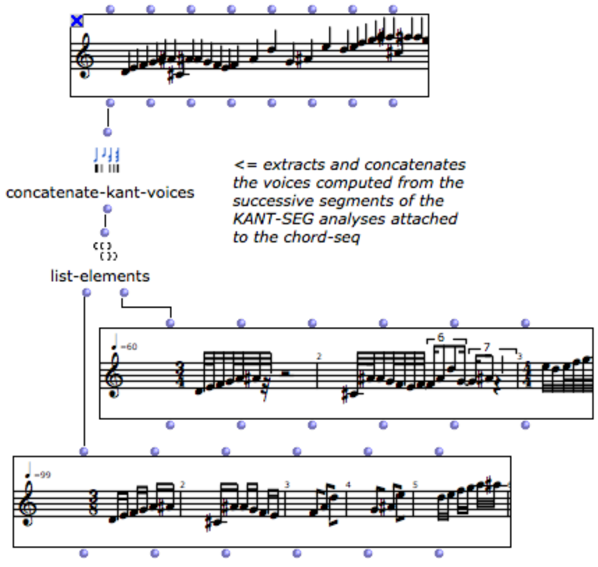

concatenate-kant-voices
<= extracts and concatenates the voices computed from the successive segments of the KANT-SEG analyses attached to the chord-seq
list-elements
=60
=99

Figure 14. Extracting analysis segments' data (here, the quantified *voices*) in OpenMusic visual programs. The quantified voices at the bottom correspond to the analyses in Figures 11 and 13.

6. CONCLUSION

We presented a new framework implemented in the OpenMusic environment allowing to connect and run user-defined musical analysis processes and representations in the score editors. [9]

The inbuilt analysis and segment classes in this framework are given as reference or examples: as we tried to demonstrate, it is relatively easy to define new types of analyses and benefit from the existing interaction and visualization tools provided by these predefined classes. The examples of the *harmonic-analysis* or *kant-seg* analyses presented in this article, conversely, show how it is possible to implement and redefine advanced interactions on top of the initial system.

As compared to related work in the computational analysis field, this framework tries not to propose any specific analysis approach. As far as possible, no assumption is made on the nature and contents of the analyses, so that any approach can be integrated and benefit from the graphical interfaces and additional processing possibilities on the musical material provided by the environment (be it for analytical or compositional purpose). This flexibility is however at the price of programming efforts (and corresponding skills), although future improvement shall complement the set of available segments and classes, and head toward minimizing the programming tasks required from the user.

Acknowledgments

The harmonic analysis project is supported by the University of Alicante project GRE09-32, the Spanish Ministry project TIN2009-14247-C02-02, and the program Consolider Ingenio 2010 (CSD2007-00018).

[9] The analysis framework shall be available in OM 6.6 and will contain a preliminary set of example analysis classes, including the ones presented in this paper.

7. REFERENCES

[1] A. Riotte and M. Mesnage, *Formalismes et modèles musicaux.* Editions Delatour France / IRCAM, 2006, (2 volumes).

[2] I. Xenakis, *Formalized Music: Thought and Mathematics in Composition.* Pendragon Press, 1992.

[3] M. Andreatta, "Méthodes algébriques dans la musique et la musicologie du XXème siècle : aspects théoriques, analytiques et compositionnels," Ph.D. dissertation, École de Hautes Études en Sciences Sociales, Paris, 2003.

[4] E. Cambouropoulos, "Towards a General Computational Theory of Musical Structure," Ph.D. dissertation, University of Edinburgh, 1998.

[5] L. Camilleri, "Computational Musicology. A Survey on Methodologies and Applications," *Revue Informatique et Statistique dans les Sciences Humaines*, vol. 29, 1993.

[6] M. Mesnage, "Morphoscope, a Computer System for Music Analysis," *Interface (Journal of New Music Research)*, vol. 22, no. 2, 1993.

[7] D. Huron, *Humdrum*, Center for Computer Assisted Research in the Humanities, 1994, (resources and documentations available at http://humdrum.ccarh.org/).

[8] G. Milmeister, "The rubato composer music software: Component-based implementation of a functorial concept architecture," Ph.D. dissertation, University of Zurich, 2006.

[9] C. Pérez-Sancho, D. Rizo, J. M. Iñesta, P. J. Ponce de León, S. Kersten, and R. Ramirez, "Genre Classification of Music by Tonal Harmony," *Intelligent Data Analysis*, vol. 14, no. 5, 2010.

[10] E. Gómez, "Tonal Description of Music Audio Signals," Ph.D. dissertation, MTG, Universitat Pompeu Fabra, Barcelona, 2006.

[11] M. T. Pearce, D. Müllensiefen, and G. A. Wiggins, "A Comparison of Statistical and Rule-based Models of Melodic Segmentation," in *Proceedings of the International Conference on Music Information Retrieval*, Philadelphia, USA, 2008.

[12] P. R. Illescas, D. Rizo, and J. M. Iñesta, "Learning to Analyse Tonal Music," in *Proceedings of the International Workshop on Machine Learning and Music*, Helsinki, 2008.

[13] O. Lartillot-Nakamura, "Fondements d'un système d'analyse musicale computationnelle suivant une modélisation cognitiviste de l'écoute," Ph.D. dissertation, Université Pierre et Marie Curie, Paris, 2004.

[14] P. Couprie, "iAnalyse : un logiciel d'aide à l'analyse musicale," in *Actes des Journées d'Informatique Musicale*, Albi, 2008.

[15] M. Mesnage and A. Riotte, "Modelisation informatique de partitions, analyse et composition assistees," in *Les Cahiers de l'Ircam (3).* Paris: IRCAM, 1993.

[16] B. Meudic, "Détermination automatique de la pulsation, de la métrique et des motifs musicaux dans des interprétations à tempo variable d'œuvres polyphoniques," Ph.D. dissertation, Université Pierre et Marie Curie, Paris, 2004.

[17] Y.-K. Ahn, "L'analyse musicale computationnelle : rapport avec la composition, la segmentation et la représentation à l'aide des graphes," Ph.D. dissertation, Université Pierre et Marie Curie, Paris, 2009.

[18] M. Laurson, M. Kuuskankare, and K. Kuitunen, "Introduction to Computer-Assisted Music Analysis in PWGL," in *Proceedings of the Sound and Music Computing Conference*, Salerno, Italy, 2005.

[19] M. Kuuskankare and M. Laurson, "Survey of Music Analysis And Visualization Tools In PWGL," in *Proceedings of the International Computer Music Conference*, Belfast, N. Ireland, 2008.

[20] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic," *Computer Music Journal*, vol. 23, no. 3, 1999.

[21] D. Hanninen, "Orientations, Criteria, Segments: A General Theory of Segmentation for Music Analysis," *Journal of Music Theory*, vol. 45, no. 2, 2001.

[22] C. Hasty, "Segmentation and Process in Post-Tonal Music," *Music Theory Spectrum*, vol. 3, 1981.

[23] M. Mesnage, "Techniques de segmentation automatique en analyse musicale," *Musurgia*, vol. 1, no. 1, 1994.

[24] M. Andreatta and C. Agon, "Implementing Algebraic Methods in OpenMusic," in *Proceedings of the International Computer Music Conference*, Singapore, 2003.

[25] B. Pardo and W. P. Birmingham, "Algorithms for Chordal Analysis," *Computer Music Journal*, vol. 26, no. 2, 2002.

[26] C. Agon, G. Assayag, J. Fineberg, and C. Rueda, "Kant: a Critique of Pure Quantification," in *Proceedings of the International Computer Music Conference*, Aarhus, Danemark, 1994.

[27] P. Desain and H. Honing, "The Quantization Problem: Traditional and Connectionist Approaches," in *Understanding Music with AI: Perspectives on Music Cognition*, M. Balaban, K. Ebcioglu, and O. Laske, Eds. Cambridge: MIT Press, 1992.

[28] C. Agon, K. Haddad, and G. Assayag, "Representation and Rendering of Rhythm Structures," in *Proceedings of the Second International Conference on Web Delivering of Music – WedelMusic'02*, Darmstadt, Germany, 2002.

Appendix: Example of an Analysis Class (Source code in Common Lisp)

```lisp
;==================================
; CLASS DEFINITION
;==================================

(defclass! my-analysis (om::abstract-analysis) ())

;==================================
; SETUP FOR DEFAULT BEHAVIOURS
;==================================

; MY-ANALYSIS is only compatible with objects of type 'chord-seq'
(defmethod compatible-analysis-p ((analyse my-analysis) (object om::chord-seq)) t)
(defmethod compatible-analysis-p ((analyse my-analysis) (object t)) nil)

; MY-ANALYSIS is able to perform segmentation, analysis of the segments,
; ... and both in the same run
(defmethod compute-segments-p ((self my-analysis)) t)
(defmethod analyse-segments-p ((self my-analysis)) t)
(defmethod compute+analyse-segments-p ((self my-analysis)) t)

; The default segment type for MY-ANALYSIS is 'chord-segement'
(defmethod default-segment-class ((self my-analysis)) 'chord-segment)

;==================================
; DEFINING THE SEGMENTATION PROCESS
;==================================

; Automatic segmentation option: Each chord in the score makes a new segment
(defmethod compute-analysis-segments ((self my-analysis) (object t))
  (loop for c in (om::get-real-chords object)
        for i = 0 then (+ i 1) collect
        (make-instance 'chord-segment :chord-ids (list i)
                       :color (om-random-color))
      )))

;==================================
; DEFINING THE ANALYSIS PROCESS
;==================================

; Segment analysis:
; Compute an object of type 'n-cercle' (circular pitch class representation)
; from all the pitches found in the segment.
(defmethod analyse-one-segment ((self my-analysis) (seg chord-segment) (object t))
   (let* (
         ; Get all chords from segment's chord-IDs
         (chords (loop for n in (chord-ids seg) collect (nth n (om::get-real-chords object))))
         ; Gather all pitches (<lmidic>) from collected chords
         (pitches (remove-duplicates (apply 'append (mapcar 'om::lmidic chords))))
         ; Build a new global chord and convert to 'n-cercle' (chord2c)
         (circle (om::chord2c (make-instance 'om::chord :lmidic pitches) 2)))
    ; Set the segment data
    (setf (segment-data seg) circle)))

; Segment display by MY-ANALYSIS (see Figure 5)
(defmethod draw-segment-data ((self my-analysis) segment view)
  (let* (
         ; Segment bounds in pixels
         (x1 (time-to-pixels view (segment-begin segment)))
         (x2 (time-to-pixels view (segment-end segment)))
         ; Segment center in pixels
         (mid (round (+ x1 x2) 2)))
    (when (segment-data segment)
      ; Segment-data is an instance of 'N-CERCLE' (see analyse-one-segment)
      (om::draw-cercle (segment-data segment) view mid (- (h view) 120) 40 2 3 t 0)
      (om-with-font *om-default-font1*
        (om-draw-string (- mid 20) (- (h view) 60)
                        (format nil "~A" (car (om::puntos (segment-data segment))))))
      )))

;==================================
; DEFINING ADDITIONAL ACTIONS
;==================================

; Analysis "cleanup": removes segments of size < 2
(defmethod clean-analysis ((self my-analysis))
   (setf (analysis-segments self)
         (remove-if
              #'(lambda (seg) (< (length (chord-ids seg)) 2))
              (analysis-segments self)))
   )

; Corresponding hook in the editor contextual menu (see Figure 3)
(defmethod get-analysis-menu-items ((self my-analysis))
  (list (om-new-leafmenu "Clean Analysis" #'(lambda () (clean-analysis self)))
 ))
```