HAL
open science

A Modular Sound Descriptor Analysis Framework for Relaxed-real-time Applications

Diemo Schwarz, Norbert Schnell

To cite this version:

Diemo Schwarz, Norbert Schnell. A Modular Sound Descriptor Analysis Framework for Relaxed-real-time Applications. International Computer Music Conference (ICMC), Jun 2010, New York, United States. pp.1-1. hal-01161252

HAL Id: hal-01161252

https://hal.science/hal-01161252v1

Submitted on 8 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A MODULAR SOUND DESCRIPTOR ANALYSIS FRAMEWORK FOR RELAXED-REAL-TIME APPLICATIONS

Diemo Schwarz, Norbert Schnell

IRCAM–CNRS STMS
1 Place Igor Stravinsky

ABSTRACT

Audio descriptor analysis in real-time or in batch is increasingly important for advanced sound processing, synthesis, and research, often using a relaxed-real-time approach. Existing approaches mostly lack either modularity or flexibility, since the design of an efficient modular descriptor analysis framework for commonly used real-time environments is non-trivial. We first lay out the requirements for such a framework before describing our modular architecture that integrates instantaneous descriptors, segmentation modules, temporal modeling, converter modules, and external descriptors or meta-data. We present its proof of concept implementation in MAX/MSP and examples that show the simplicity with which new analysis modules can be written.

1. INTRODUCTION

Extraction of descriptors from audio signals is an essential technique in many fields of computer music research and production, and music information retrieval. It is becoming increasingly important for the next generation of real-time processing systems, such as content-based processing [1], corpus-based concatenative synthesis [13], and multimodal applications including gesture analysis. These techniques often employ a *relaxed real-time* approach where the incoming data streams are not immediately processed into output data streams. In this approach the incoming data and/or information extracted from the data is recorded and the output streams are generated from the recorded data. This approach allows for the integration of information over time and for applying advanced content based audio processing techniques such as temporal modeling, recognition, and classification. Applications using this approach often employ a segmental approach, i.e. the basic unit of information would be a segment of a sound signal with various descriptors characterising the segment as a whole.

An *audio descriptor* is a value that describes a certain quality of a sound, which can evolve over time or be constant. Although often used interchangeably with the term *audio feature*, the distinction can be phrased like this: *A descriptor defines the syntax and the semantics of one representation of a particular feature of audiovisual content. A feature is a distinctive characteristic of the data which is of significance to a user* [4].

There are a multitude of useful audio descriptors, and each can be calculated in a number of ways and variants, see e.g. Peeters [8] for an overview. This multitude means that a monolithic approach to descriptor analysis can never satisfy the needs for either comprehensiveness or flexibility in exploring different descriptors and versions.

The need for this flexibility, together with the desirable property of sharing the same code for the descriptor algorithms with off-line analysis programs, lead us to develop a modular descriptor analysis framework the architecture of which had to be carefully planned and is non-trivial.

1.1. Overview of Descriptor Extraction

The analysis of descriptors from audio data is also called *extraction*. In the general extraction process, the audio data is treated as a stream of overlapping windowed signal frames. For each frame, the *instantaneous descriptors* (abbreviated *inst*) are calculated, e.g. pitch, loudness, brilliance.

Some descriptors serve to take a decision on segmentation, or the stream of frames is divided into constant-size *texture window* segments. On the segments, a *temporal modeling* stage calculates segment descriptors that characterise the temporal evolution of the instantaneous descriptors over the duration of the segment, e.g. the mean value or the amount of variation, the slope, etc.. Some of these temporal modelings are applicable to any descriptor, some make only sense for certain descriptors (e.g. geometric mean for pitch). We call the former *universal*, and the latter *specific temporal modeling*. The resulting segment descriptors are called *const descriptors*. Segment descriptors can also be provided from the outside, e.g. from meta-data importer modules, or be global descriptors.

Any descriptor can also depend on one or several other descriptors, and a *converter* module can calculate a different representation (e.g. pitch in MIDI Note Number from pitch in Hz) or additional information from them (e.g. Chroma value from pitch).

Regarding the different levels of modularity, the spectrum ranges from completely monolithic systems, over libraries with static modularity, where modules can be config-

ured at run-time, but have to be known at compile-time, to dynamic modular systems like plugin or component frameworks, where new modules can be inserted at run-time.

2. RELATED WORK

Real-time descriptor analysis has been a concern since the beginning of electro-acoustic music, and gained even more importance with content-based processing [1].

Many monolithic analysis modules for popular real-time environments exist, such as analyser~ [5] and many more. The first descriptor analysis frameworks that would allow the dynamic inclusion of external modules are either plugin frameworks such as the sadly defunct FEAPI [6], and the more lively VAMP,[1] or the patch-based ZSA [7] for MAX/MSP. The CATART system [14] for corpus-based concatenative synthesis in MAX/MSP introduced for the first time real-time and batch analysis, thanks to GABOR [11], but on a statically modular framework.

The powerful dedicated music information retrieval (MIR) framework MARSYAS is concerned with scheduling [3], as well as CLAM[2], but neither is a common environment for real-time sound and music applications.

The IRCAMDESCRIPTOR template library developed in the framework of the SampleOrchestrator project[3] is focused on descriptor extraction for classification in MIR. It models the dependency graph between analyser modules and important implementation aspects such as the scheduling and buffering policy by extensive use of C++ metaprogramming techniques. However, this means that new modules can not be integrated at run-time, and external segmentation sources are difficult to combine with the internal fixed-window-size oriented segmentation.

3. REQUIREMENTS

In this section, we will try to give an overview over the most important requirements for a modular descriptor analysis framework for real-time and off-line use that implements the general extraction process described in section 1.1. We first give requirements concerning functionality, then concerning efficient implementation in a real-time system.

3.1. Functionality Requirements

Scheduling Analysis should run either in batch on sound files and buffers, or on a live audio stream

Switch Analysis modules can be enabled/disabled at runtime

Temporal Modeling Any number of universal and specific temporal modeling algorithms can be integrated.

Segmentation Allow several streams of segmentations in parallel and overlapping segments, or an implicit segmentation, where segments are analysis frames, elementary wave forms, or whole sound files. Segmentation times can be given slightly in the past.

Descriptor Data Type Descriptors can be numeric scalars or vectors, or symbols

External Files Instantaneous and const descriptors, segment boundaries, and global meta-data should be imported from SDIF, text, XML or Matlab data files and merged with the flow of data.

3.2. Implementation Requirements

Efficient Modularisation The framework should allow an efficient implementation, notably by *sharing commonly used calculation results*, most of all the FFT representation, between modules, by *avoiding copying* and sending lists of data, instead writing them directly to its destination, and by *parallel processing*, e.g. by distributing work packets over different processor cores.

Multiple Graphs Different graphs of analysis modules should be able to work independently for different corpora in the same application.

External Segmentation External sources of segmentation, such as a human tapping on attacks, must be integratable into the data flow.

Additional Information More information should be attachable to descriptors, like descriptive text, unit, grouping, symbol list, distance matrix, distance function.

Reanalysis A subset of descriptors or only the segmentation and subsequent temporal modeling can be re-run with changed parameters.

4. ARCHITECTURE

This section describes our dynamically modular architecture for a descriptor analysis framework within a dataflow programming environment such as Max/MSP or PD. The framwork consists of a small number of fixed modules that exchange data with any number of analyser modules (descriptor analysis, segmentation, temporal modeling) provided by the patch programmer, by means of a protocol.

The fixed framework modules take care of the scheduling of the data exchange at the various rates (frame rate, segment rate, file rate), the intermediate storage of the calculated descriptor data, and the handling of dependencies between analyser modules.

At initialisation, each analyser module receives a discovery message, upon which it registers itself with the

[1] http://vamp-plugins.org
[2] http://clam-project.org
[3] http://www.ircam.fr/sor.html

framework, and declares its dependencies (descriptors it wants to receive as input). The framework then sends a reference to where the output data has to be written to each module.

At runtime, each analyser module receives the data it needs to analyse, together with timestamps, and the produced data is written directly to the output reference by the module.

4.1. Input Types

The different types of data that is input to analysis modules are given in the following, grouped by their rate:

At frame rate	At unit rate	At file rate
Signal sound vector **FFT** magn. vector **Inst** frame time, instantaneous descriptors vector	**Segment** start time, end time, instantaneous descriptors matrix **Const** start time, end time, const descr. vector	**File data** one file's segment descriptor matrix

4.2. Output Types

The different types of output data that is produced by analysis modules are:

Inst instantaneous descriptors vector
Const const descriptors vector (meta-data, global, etc.)
Segmentation seg. time, keep last segment flag

4.3. Module Types

The above data types serve to determine the useful distinctions of different types of modules by their input and output data types, summarised in table 1.

4.4. Dataflow Schema

Figure 1 shows the dataflow between the 5 fixed framework modules (in ellipses) and the different types of analyser modules (in rectangles). To the right, the underlying components Descriptor Schema and Descriptor Container take care of the definition and registration of information about the attached analyser modules and their output descriptors, and the intermediate and final storage of results.

in \ out	inst	segment	const
signal	analyser (signal)	segmenter (signal)	-
fft	analyser (fft)	segmenter (fft)	-
inst	converter	segmenter (descr)	-
segment	-	-	temporal modeling
const or file	-	-	converter, meta-data

Table 1. Module types by input and output data type.

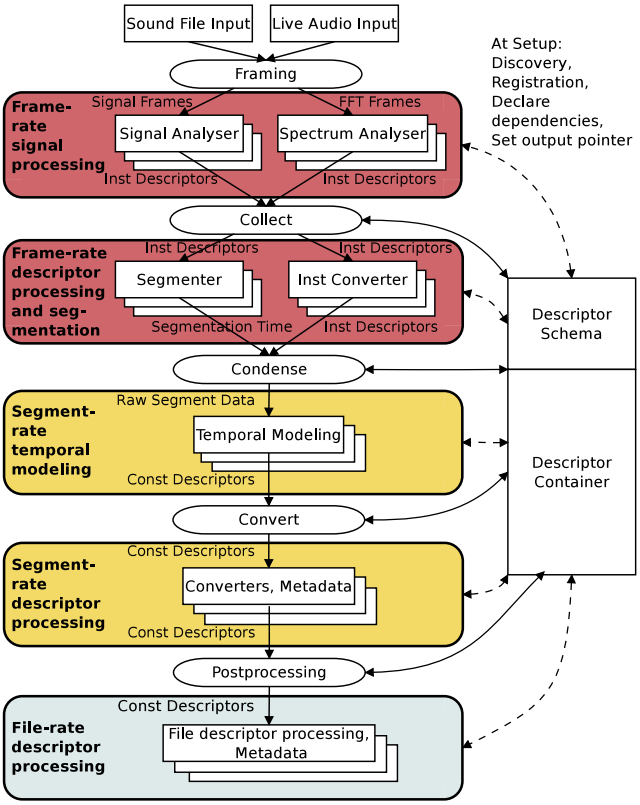

Figure 1. Dataflow schema of the modular descriptor analysis framework, showing the 5 different scheduling steps.

The fixed framework modules control the scheduling of processing. The Framing module splits audio from files or a live stream into signal frames and outputs these to signal analysers. If any spectrum analysers are attached, it also calculates and outputs an FFT. The Collect module gathers the instantaneous descriptors calculated above and redistributes them to converter and segmentation modules. The Condense module is triggered by a segmentation message and outputs the gathered instantaneous descriptors for the current segment to the attached temporal modeling modules. The Convert module provides the segment descriptors to converter or external meta-data modules. Finally, the Postprocessing module is called once per analysed file or live session to calculate segment descriptors that need the whole file's segments and descriptors to be present.

5. PROOF OF CONCEPT IMPLEMENTATION

The architecture described above was first implemented in the CataRT system [4] for real-time interactive corpus-based concatenative synthesis[12, 13, 14], based on the FTM&Co. extension library [9] at [5] for Max/MSP, taking advantage of FTM's real-time optimised data structures and

[4] http://imtr.ircam.fr/imtr/CataRT [5] http://ftm.ircam.fr

powerful sound and matrix processing tools [11, 2]. A second implementation based on the MUBU container [10] is in progress and shares the framework (with a different implementation of the fixed modules) and, most importantly, the analysis modules.

An analysis patch looks very similar to figure 1. The analysis modules are simply connected to the corresponding framework module. They can be very simple, as figure 2 (left) shows, since all details of the protocol are handled by a stub module imtr.analysis.stub.

6. CONCLUSION AND FUTURE WORK

The modular descriptor analysis framework architecture and the implementation we presented in this article provides several advantages over the state of the art of existing frameworks:

- great flexibility in choosing only the needed modules for a given task
- adding existing modules is as simple as making one connection; they will be discovered and initialised automatically, can be presented in a menu, etc.
- writing new modules is very simple, since the stub handles all of the protocol (separation of concerns)
- intermediate results are available in real-time at the output of the modules, for tight real-time processing such as envelope following or visualisation

One limitation is that unlike IRCAMDESCRIPTOR with its fixed-size segment approach that can be extended over multiple scales, the time resolution hierarchy in our framework is limited to two levels (frame and segment rate), for reasons of scheduling. Further scale levels could only be added by introducing new scheduling modules.

One important point we learned during design and implementation is the importance of the message protocol between the modules, that is more stable than the framework implementation itself, which will be adapted in the future re-implementation using Mubu [10].

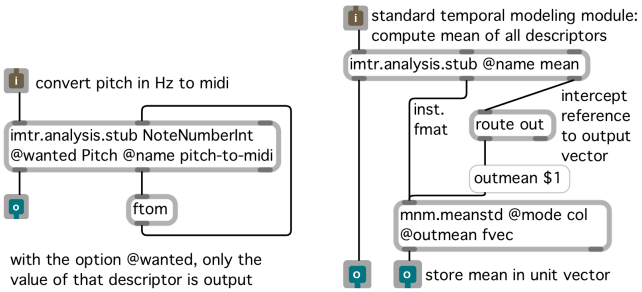

Figure 2. Example of a converter module (left) and a temporal modeling module (right), using the stub to declare the module name and the produced descriptor.

7. ACKNOWLEDGEMENTS

The authors would like to thank Carmine Cella and the anonymous reviewers for their detailed comments.

8. REFERENCES

[1] X. Amatriain, J. Bonada, A. Loscos, J. Arcos, and V. Verfaille, "Content-based transformations," *J. of New Music Research*, vol. 32, no. 1, pp. 95–114, 2003.

[2] F. Bevilacqua, R. Muller, and N. Schnell, "MnM: a Max/MSP mapping toolbox," in *New Interfaces for Musical Expression*, Vancouver, 2005, pp. 85–88.

[3] N. Burroughs, A. Parkin, and G. Tzanetakis, "Flexible scheduling for dataflow audio processing," in *Proc. ICMC*, New Orleans, USA, 2006.

[4] J. Hunter, "MPEG7 Behind the Scenes," *D-Lib Magazine*, vol. 5, no. 9, 1999, http://www.dlib.org/.

[5] T. Jehan, "Musical signal parameter estimation," Master's thesis, IFSIC, Univ. Rennes, and CNMAT, Univ. California, 1997.

[6] A. Lerch, G. Eisenberg, and K. Tanghe, "FEAPI: A low level feature extraction plugin API," in *Digital Audio Effects (DAFx)*, Madrid, 2005.

[7] M. Malt and E. Jourdan, "Zsa. Descriptors: a library for real-time descriptors analysis," in *Sound and Music Computing (SMC)*, Berlin, Germany, 2008.

[8] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the Cuidado project," Ircam – Centre Pompidou, Tech. Rep., 2004.

[9] N. Schnell, R. Borghesi, D. Schwarz, F. Bevilacqua, and R. Müller, "FTM—Complex Data Structures for Max," in *Proc. ICMC*, Barcelona, 2005.

[10] N. Schnell, A. Röbel, D. Schwarz, G. Peeters, and R. Borghesi, "MuBu & friends – assembling tools for content based real-time interactive audio processing in Max/MSP," in *Proc. ICMC*, Montreal, 2009.

[11] N. Schnell and D. Schwarz, "Gabor, Multi-Representation Real-Time Analysis/Synthesis," in *DAFx*, Madrid, 2005.

[12] D. Schwarz, "Concatenative sound synthesis: The early years," *JNMR*, vol. 35, no. 1, Mar. 2006.

[13] ——, "Corpus-based concatenative synthesis," *IEEE Sig. Proc. Mag.*, vol. 24, no. 2, Mar. 2007.

[14] D. Schwarz, R. Cahen, and S. Britton, "Principles and applications of interactive corpus-based concatenative synthesis," in *JIM*, GMEA, Albi, France, Mar. 2008.