MUBU & FRIENDS - ASSEMBLING TOOLS FOR CONTENT BASED REAL-TIME INTERACTIVE AUDIO PROCESSING IN MAX/MSP

Norbert Schnell, Axel Röbel, Diemo Schwarz, Geoffroy Peeters, Riccardo Borghesi

IRCAM - Centre Pompidou
Paris, France

ABSTRACT

This article reports on developments conducted in the framework of the *SampleOrchestrator* project. We assembled for this project a set of tools allowing for the interactive real-time synthesis of automatically analysed and annotated audio files. Rather than a specific technique we present a set of components that support a variety of different interactive real-time audio processing approaches such as *beat-shuffling*, *sound morphing*, and *audio musaicing*.

We particularly insist on the design of the central element of these developments, an optimised data structure for the consistent storage of audio samples, descriptions, and annotations closely linked to the SDIF file standard.

1. INTRODUCTION

The tools provided by past and current research in music information retrieval provide manifold descriptions that permit the transformation of recorded sounds into interactive processes based on signal models, sound and music perception, musical representation, as well as further abstract mechanical and mathematical models. A variety of interesting approaches to interactive real-time processing are based on the transformation of certain aspects of a sound (e.g. its *pitch* or *rhythm*), while preserving others.

The *SampleOrchestrator* project[1], has allowed for the convergence of several music information retrieval and audio processing techniques into a set of complementary tools summarizing research carried out over the past ten years. While our work has been focused on experimentation with the use of audio descriptors and annotations for the interactive transformation and hybridization of recorded sounds, other parts of the project aimed at the development of novel techniques and tools for automatic orchestration as well as for the organization of large data bases of sounds.

Once given the availability of an adequate set of audio processing tools, the necessity to consistently represent audio data with audio descriptions and annotations within real-time processing environments such as Max/MSP appeared as a critical issue and led us to the development of the *MuBu* data structure.

2. THE *MUBU* MULTI-BUFFER

The *MuBu* data structure has been designed as a generic container for sampled sounds as well as any data that can be extracted from or associated to sampled sounds such as audio descriptors, segmentation markers, tags, music scores, and motion tracking data. *MuBu* is a multi-track container representing multiple temporally aligned (synchronized) homogeneous data streams similar to data streams represented by the SDIF standard [2] associating each element to a precise instant of time.

The implementation of *MuBu* overcomes restrictions of similar existing data structures provided in Max/MSP such as the *buffer~* and *SDIF-buffer* [9] modules as well as the data structures provided by the Jitter and FTM [3] libraries. Special attention has been paid to the optimization of shared RAM based storage and access in the context of concurrent multi-threaded real-time processing. All access to the data is thread-safe while the methods used in real-time signal processing methods have been implemented in lock free way.

A particular challenge for the design of *MuBu* has been the idea to provide a generic container that can represent and give access to any kind of data, but that also allows for the implementation of specific access and processing methods for tracks of particular data types. For example, an additive synthesis module connected to a *MuBu* container should be able to automatically find and connect to a track of partials or to a track of a fundamental frequency and another track of harmonic amplitudes. The synthesis module can be automatically configured depending on the additive representation loaded into one or multiple tracks of the container.

So far, *MuBu* implements import methods for audio files of various formats, SDIF files, MIDI standard files, and generic XML files.

We have developed a Max/MSP module integrating the *MuBu* library and implementing a set of messages to manage and access the data. In addition to the audio processing tools described below, we defined a set of Max/MSP modules referring to the *MuBu* module and accessing its data copying to and from Max/MSP data structures.

[1] `http://www.ircam.fr/306.html?&L=1`

2.1. Implementation Details

Each track of *MuBu* is implemented as a contiguous memory array containing the data over time and allowing for vectorised access. Similar to the SDIF standard, numerical data is represented as a sequence of two-dimensional matrices of floating-point values (often reduced to vectors and scalars). The time axis of a *MuBu* track can be represented either by a time-tag for each element or by a constant sampling rate defined for the whole track. Thus, uniformly sampled data streams of multiple rates and non-uniformly sampled data streams can be represented aligned to each other. In addition to a reference time-tag, a track can optionally define a start and end time for each of its elements giving the possibility to associate data not just to an instant but also to a period of time.

A track can define a name tag for each column of the contained data matrices or vectors that may refer to the type or semantic of the data in a particular column. The rows of the matrix are simply indexed and typically represent one or multiple instances of the structure defined by the columns. As an example, vectors of different descriptors stored in a track would be represented as elements of one row with column names referring to the different descriptors such as *loundness*, *pitch* and *brilliance*. Similarly, a MIDI note would be represented as a single row with columns corresponding to pitch, velocity and channel with the option of representing chords as multiple lines of notes within in the same element of the track. Following the same logic, spectral coefficients would be represented as a single column (e.g. named *MFCC* or *MEL*).

Since the matrix data is stored in a contiguous memory space, the capacity of a track in terms of the maximum number of elements as well as the matrix size for each track element has to be defined in advance whenever possible. This choice has been taken with respect to previous experiences with similar approaches showing that the vast majority of the envisaged applications can easily cope with this restriction while the implementation of the data structure and the access to the data can be enormously simplified and easily optimised in this way.

If in a particular data stream the elements of a track can have a variable number of rows (e.g. chords of notes or harmonics), the maximum number of rows must be defined and an additional vector is created within the track giving the actual number of rows for each element of the stream. For non-numerical data or for handling predefined data structures that can not be expressed with two-dimensional matrices, a track can define an additional optional vector of user data handles for each element requiring customised access methods.

A *MuBu* track can have a type tag such as *AudioSamples*, *Harmonics* or *MidiNotes*. This type tag can be used by an application or a particular set of software modules to coherently identify and process the stored data. We have started to provide a few predefined type tags and specified an API for user defined types. The type tag of the track also specifies the data type for the user data handles if used.

For the association of meta-data to a track, each track includes a name/value table. This table also allows to completely represent the most important file formats with one or multiple *MuBu* tracks and to enable the regeneration of identical or modified files from the stored data.

2.2. Display and Graphical Interaction

In parallel to the *SampleOrchestrator* project we have worked on a portable library for the display and editing of data streams such as those represented by *MuBu*. The library has been implemented based on the *Juce* framework[2] and provides a set of graphical displays and editors that remain independent from any data structure by providing interfaces for the access to the data. The first integration of these editors has been implemented for the FTM library in Max/MSP.

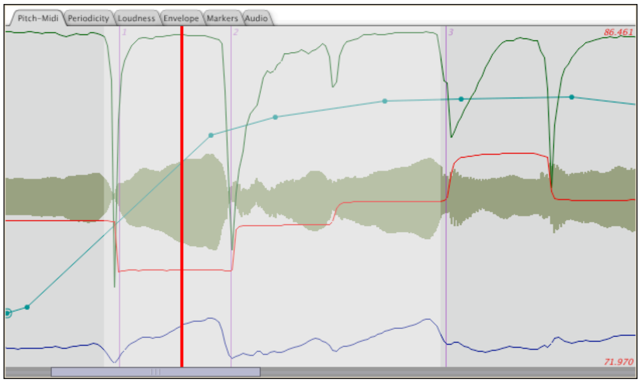

Figure 1. The *IMTR Editor* graphical component displaying multiple superposed data streams

In summary, the current version of the IMTR editor library provides the following features already fitting the needs of a graphical interface for *MuBu*:

- Editable graphical interfaces for time-tagged scalars (e.g. bpfs), arrays of sampled values (e.g. waveforms)
- Non-editable displays for sonogram and trace (e.g. harmonics with pitch and amplitude developing over time) representations
- Various display styles for each interface (e.g. *bpf*, *steps* or *peaks* for time-tagged values)
- Superposition or juxtapositon of horizontally aligned displays with controllable opacity
- Scrolling, zooming, selection and cursor tools

[2] http://www.rawmaterialsoftware.com/juce/

3. *MUBU*'S FRIENDS

Figure 2 shows the overall architecture of the components assembled around the *MuBu* data structure that are briefly described in the following sub-sections.

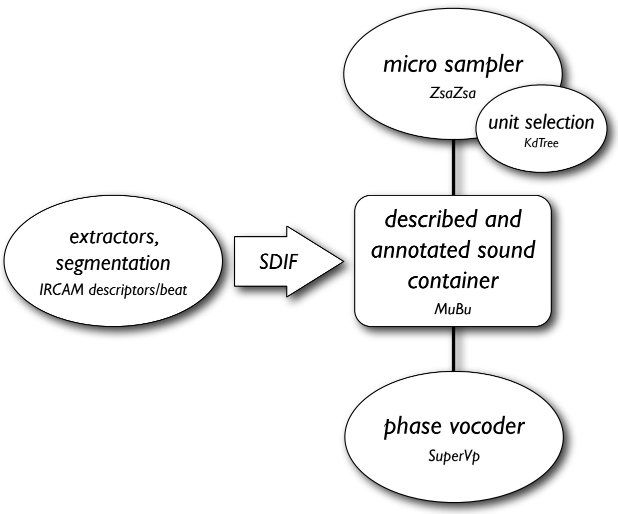

Figure 2. Overview of the assembled tools

3.1. Audio Analysis Tools

In the prototype applications that have guided our developments, the *IrcamDescriptors* and *IrcamBeat* command-line tools are used to perform off-line analysis of audio files written into SDIF files and loaded into *MuBu*. In addition, we have used the *AudioSculpt* application[1] to perform segmentation based on *SuperVP* also exported to SDIF files.

IrcamDescriptors has been partly developed within the *SampleOrchestrator* project based on the previously developed software and Matlab libraries. It basically implements the audio descriptors described in [5].

The *IrcamBeat* [6] command-line tool as well as *AudioSculpt* have been used to generate automatic segmentation for concatenative synthesis. While *IrcamBeat* performs an overall tempo analysis of rhythmic musical material to generate markers fitting a metric grid, the *SuperVP* engine of *AudioSculpt* generates markers at transitions based on spectral transitions without any further musical model. The two techniques are complementary and useful for different audio materials and concrete applications.

For the efficient implementation of a variety of interesting interactive real-time synthesis paradigms explored in the framework of the *SampleOrchestrator* project using similarity and distance measures, our collection of tools integrates the *KdTree* library [8]. The K-Nearest Neighbors (KNN) algorithm implemented by the library optimises the selection of segments or frames in a pre-analysed source audio stream based on the distance between numeric descriptor values. The *KdTree* library has been developed for the project and integrated with the other tools into a set of Max/MSP modules.

3.2. *SuperVP* Real-Time Modules

The *SuperVP* extended phase vocoder [7] has existed for some time as a command-line tool and as the audio processing engine behind *AudioSculpt* before being released as real-time modules that also have been integrated into Max/MSP as audio processing modules[3]. In the framework of the *SampleOrchestrator* project, these modules have been extended and currently integrate the following set of features:

- Time-stretching/compression and scrubbing with optional transient and waveform preservation
- Pitch transposition with optional transient, waveform and spectral envelope preservation
- Spectral envelope transformation (frequency and amplitude scaling)
- Remixing of spectral transient, sinusoidal and noise components
- Generalised cross-synthesis
- Source-filter cross-synthesis

In addition to the excellent sound transformation quality of the phase vocoder practically free of artefacts, the connection of *SuperVP* with *MuBu* provides the possibility of applying sound transformations as a function of previously extracted audio descriptors and segmentation. This permits to impose within certain limits an arbitrary evolution of temporality, monophonic pitch, loudness and spectral envelope to a given sound and create sounds by hybridisation of feature extracted from two or multiple sounds. In our prototype applications, we have used *SuperVP* to transform and hybridise strictly monophonic sounds recorded from solo instruments and voices.

3.3. *ZsaZsa*

The *ZsaZsa* library implementing a granular synthesis and concatenative synthesis engine has been developed as a second synthesis engine connected to *MuBu* complementary to *SuperVP*[4]. The design of *ZsaZsa* is based on the experiences with *Gabor* [4] in FTM & Co[5], a set of audio analysis and synthesis modules for Max/MSP allowing for highly modular programming of overlap-add audio processing. The *ZsaZsa* library even if integrated into Max/MSP as monolithic signal processing module, internally maintains a modular design separating the copying of the grains from the source sound, windowing, resampling and reinsertion of the

[3] The *SuperVP* Max/MSP modules are released in the IRCAM Forum. `http://forumnet.ircam.fr/708.html?L=1`

[4] We like to present *ZsaZsa* as *SuperVP's little sister*.

[5] `http://ftm.ircam.fr/`

grains into one or multiple output streams very similar to the design of typical *Gabor* applications. Other than *Gabor*, *ZsaZsa* is limited to asynchronous and synchronous granular synthesis and concatenative synthesis. The *ZsaZsa* library implements a completely generic engine that distinguishes these three synthesis modes only by the disposition and succession of synthesised segments (i.e. grains) and the chosen control parameters.

The *ZsaZsa* synthesis engine has a slightly extended set of granular synthesis parameters and implements different segmentation schemes corresponding to the markers and segments provided by the *MuBu* container. Without segmentation, a grain is defined by an arbitrary onset time and duration in the source audio stream. When defining a segmentation (e.g. by referring to a *MuBu* track with time-tags and/or segments), a grain can be determined by one two or three markers, and a set of parameters illustrated in figure 3.

Especially for synthesis using segmentation, the player can be extended by a custom callback function. The callback function is implemented by the synthesis module integrating the *ZsaZsa* engine and is called each time when a new segment has to be synthesised taking into account the current *Period*/*Tempo* or *TempoFactor*. A callback function customised for a particular application determines the next segment to be synthesised represented by an index in the marker arrays. For example, a callback function could simply count forwards or backwards looping through the segments, randomly shuffle segments, or calculate the next segment regarding a set of rules, a statistical model and/or similarities calculated from a set of audio descriptors associated to the segments.

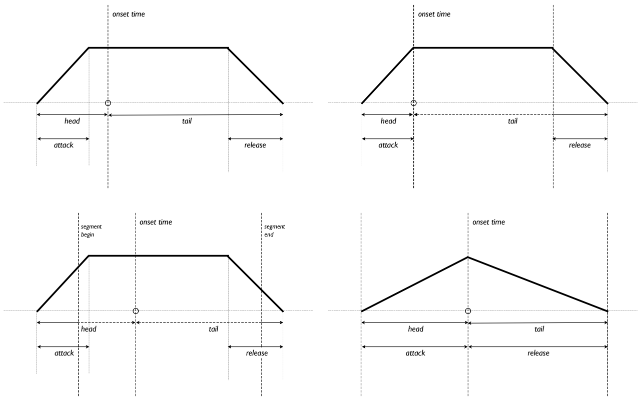

Figure 3. *ZsaZsa* windows defined by one, two or three markers and in the case of pitch synchronous synthesis.

In a first version, the *ZsaZsa* library has been integrated into three different Max/MSP modules, one performing granular synthesis (synchronous if a track of waveform markers is available in the referred *MuBu*), one performing descriptor based granular synthesis, and one for concatenative synthesis using a segmentation track.

4. CONCLUSIONS AND FUTURE WORK

We have assembled a complementary set of state-of-the-art audio processing tools integrated with Max/MSP that include a flexible data container enabling the experimentation with novel audio processing paradigms based on automatic analysis and annotation of recorded sounds.

Our current efforts are dedicated to the integration of further processing tools and paradigms, while keeping the developed framework simple and modular and trying to find an optimal balance between *generic* and *easy-to-use*.

5. ACKNOWLEDGEMENTS

The *SampleOrchestrator* coordinated by Hugues Vinet at IRCAM has been funded by the French *National Research Agency*. We especially would like to acknowledge our colleagues that have collaborated on different versions of the described developments: David Fenech, Carmine Emanuele Cella, Nicolas Sursock and Juan-Jose Burred.

Many thanks also to the team from *Universal Sound Bank* and especially Remy, Olivier and Alain, who have been (and still are) great partners in this project and with whom we hopefully continue soon on a new adventure.

6. REFERENCES

[1] N. Bogaards, “Analysis-Assisted Sound Processing with Audiosculpt,” in *DAFx*, Septembre 2005.

[2] J. J. Burred, C. E. Cella, G. Peeters, A. Röbel, and D. Schwarz, “Using the SDIF Sound Description Interchange Format for Audio Features,” in *ISMIR*, 2008.

[3] N. Schnell et al., “FTM — Complex data structures for Max,” in *ICMC*, Septembre 2005.

[4] ——, “Gabor, Multi-Representation Real-Time Analysis/Synthesis,” in *DAFx*, Septembre 2005.

[5] G. Peeters, “A large set of audio features for sound description (similarity and classification) in the CUIDADO project,” IRCAM, Tech. Rep., 2004.

[6] ——, “Template-based estimation of time-varying tempo,” *EURASIP*, 2006.

[7] A. Roebel, “A new approach to transient processing in the phase vocoder,” in *DAFx*, Septembre 2003.

[8] D. Schwarz, N. Schnell, and S. Gulluni, “Scalabilty in Content-Based Navigation of Sound Databases,” in *ICMC*, August 2009.

[9] M. Wright, R. Dudas, S. Khoury, R. Wang, and D. Zicarelli, “Supporting the Sound Description Interchange Format in the Max/MSP Environment,” in *ICMC*, October 1999.