SOUND OBJECT CLASSIFICATION FOR SYMBOLIC AUDIO MOSAICING: A PROOF-OF-CONCEPT

Jordi Janer, Martin Haro, Gerard Roma
Universitat Pompeu Fabra
{firstname.lastname}@upf.edu

Takuya Fujishima
Yamaha Corporation
fujishim@beat.yamaha.co.jp

Naoaki Kojima
Media Artist
animatedmotion@gmail.com

ABSTRACT

Sample-based music composition often involves the task of manually searching appropriate samples from existing audio. Audio mosaicing can be regarded as a way to automatize this process by specifying the desired audio attributes, so that sound snippets that match these attributes are concatenated in a synthesis engine. These attributes are typically derived from a *target* audio sequence, which might limit the musical control of the user.

In our approach, we replace the target audio sequence by a symbolic sequence constructed with pre-defined sound object categories. These sound objects are extracted by means of automatic classification techniques. Three steps are involved in the sound object extraction process: supervised training, automatic classification and user-assisted selection. Two sound object categories are considered: *percussive* and *noisy*. We present an analysis/synthesis framework, where the user explores first a song collection using symbolic concepts to create a set of sound objects. Then, the selected sound objects are used in a performance environment based on a loop-sequencer paradigm.

1 INTRODUCTION

Corpus based sound synthesis encompasses a wide range of approaches that leverage the developments of audio description technologies for speech and music synthesis applications. Initially developed for text to speech applications, Concatenative Sound Synthesis [1] has been adapted to the musical domain, notably to singing voice and musical instrument synthesis. Musical mosaicing [2] introduced a more general approach from the point of view of unit selection by mapping the search of suitable sounds from the database to a constraint satisfaction problem. However, while traditionally closer to popular practices in the reuse of musical materials, applications of musical mosaicing have been limited by the dependence on an acoustic target that is analyzed to define the constraints. In this paper we present an initial step towards symbolic audio mosaicing based on machine learning of abstract sound categories. The system describes acoustic events in polyphonic music using human-understandable concepts (e.g. percussive, having prominence of singing voice, harmonic sound with constant pitch, etc.). Then it provides the user with a symbolic audio mosaicing interface to compose music by concatenating these "concepts". Sound descriptors from the symbolic sequence are used as target in an audio mosaicing system. One of the main bottlenecks we face when trying to develop such system is the need to automatically characterize music segments using perceptually meaningful sound object categories. Nowadays we have a lot of signal-level descriptors, but these descriptors are rarely linked with perceptually meaningful sound events in polyphonic music (with few exceptions like chroma features or some rhythmic descriptors). We make use of Music Information Retrieval (MIR) techniques like sound segmentation [3], instrument classification [4], and audio feature extraction [5] to classify polyphonic sound segments into pre-defined sound object categories.

Depending on the area of research we look from, there are several interpretations of the concept of sound object. If we look from a perceptually-oriented point of view we find concepts related to auditory streams and perceptual grouping of sound events [6]. Looking from a signal processing point of view we can rely on concepts like onsets or ADSR envelope. Traditional music theory deals the concepts of notes and chords. In *Musique Concrète*, Schaeffer proposed the idea of musical objects [7]. One simple working definition can be found in [8]: "Sound Object: a basic unit of musical structure, generalizing the traditional concept of note to include complex and mutating sound events on a time scale ranging from a fraction of a second to several seconds". Within this proof-of-concept paper we start by considering only two easily identifiable types of sound objects found in polyphonic music: *percussive* and *noisy* sounds. The former category includes sounds objects with a sharp attack followed by a decay (e.g. drums, pizzicato), while the latter includes *unpitched* sound objects with flat amplitude and spectral envelopes (e.g. stable white noise, highly distorted sounds).

The main characteristic of this project is the concept of semi-automatic sound object retrieval. The research tasks focus on detecting segments of a song with high probability of being member of a pre-defined sound category. At the same time, with the analyzed sounds, the user should be

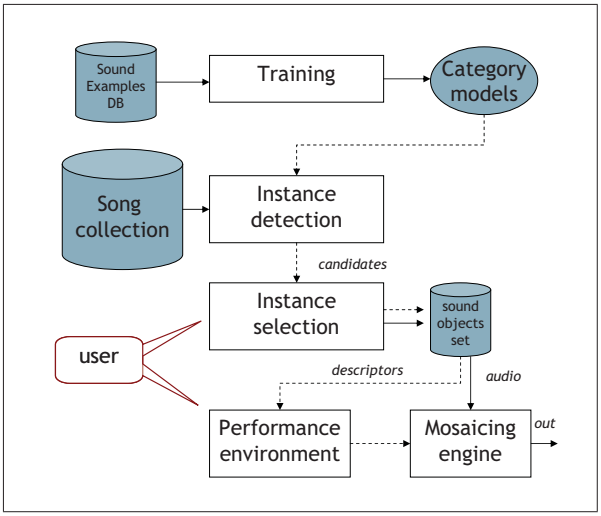

Figure 1. General overview of the implemented system. Note the user intervention in both exploration and composition processes.

able to create new audio material in the context of electronic music performances.

An overview of the process is depicted in figure 1. The steps involved are: *Instance Detection*, *Instance Selection* and *Performance Environment*. The first consists of two sub-processes: the model training, given a manual annotated ground-truth database; and the extraction of new sound object candidates from a song collection. In the instance selection process, the user selects among the extracted candidates in a GUI. Finally, the performance interface allows the user to specify different steps in a sequencer using sound categories. In the next sections we describe each step in detail.

2 AUTOMATIC INSTANCE DETECTION

In the proposed analysis framework, the user selects from a list of automatically detected sound objects. In order to detect these objects the system extracts acoustic descriptors from all songs in the collection, and creates a list of candidates. The instance detection process applies machine learning techniques to classify the list of sound candidates into pre-defined categories, outputting a class-label and a likelihood value for every proposed segment. Additional features, such as amplitude envelope and spectral content, are computed to further describe the sound objects. These features are used in the following stage, the sound object selection (see section 3).

2.1 Segment-based Sound Object Detection

Initially, one can foresee two different approaches to detect sound objects: frame-based and segment-based. In a frame-based approach sound objects can be identified by using a continuous descriptor computed out of frames of short duration. In a segment-based approach segments of longer duration are described and evaluated by an automatic classifier.

The problem of using a frame-based approach is that it ignores time-evolving aspects of the sound, such as the energy envelope. Therefore, our approach is segment-based, relying on supervised machine learning techniques. A manually annotated ground truth database is used to train models of sound object categories. From the audio file, we use an in-house sound analysis library to extract low-level acoustic features. In the next step we generate a list of segment candidates, which can overlap. For each candidate, a confidence measure is computed using automatic classification algorithms. We keep the non-overlapping segments with a confidence measure of more than 50%.

2.1.1 Training Database

In order to build a model for each sound category we need a ground truth database with labeled examples. We decide to construct two labeled databases (one per sound category) namely PercussionDB and NoiseDB.

PercussionDB: Concerning the *percussive* category, in order to obtain a more generalized model, we use a ground truth database built by processing the *ENST drums* database [9]. This is the largest publicly available drum database. It contains recordings from three different drummers and drum sets playing single hits, drum phrases and complete songs covering various styles. The authors provide two type of drum recording tracks namely dry (without sound effects) and wet tracks, along with the corresponding music accompaniments. In our case, since we want to detect percussive events in real world music, we use the wet tracks. In order to obtain "realistic" songs we mix the drums and their accompaniments tracks directly (without further changes of sound levels). Afterwards we segment 30 seconds of each song (and their labels) for a total of 64 songs. Since we want to detect all percussive sounds as belonging to one class we merge all the provided labels into one "parent" category named as "*percussive*". Finally we keep the sound events that are found by an onset detector [3], labelling intra-onset segments with a maximum length of 150ms. If the onset has at least one percussive label in its vicinity ($\pm$40ms) it is labeled as "*percussive*" otherwise is labeled as "*non-percussive*". At the end of this process we obtain 3.690 examples of percussive events (and 3.690 of non-percussive ones).

NoiseDB: Concerning the *noisy* category, we collected several songs containing potential "noisy" sounds, as well as isolated noise sound from sample collections. We manually

Amplitude-related object descriptors
Mean
Variance
Minimum
Maximum
Skewness
Kurtosis
Time-related object descriptors
Temporal Skewness
Temporal Kurtosis
Temporal Centroid
Maximum Normalized Position (MxNP)
Minimum Normalized Position
Slope: arc tangent of the slope of the linear regression of the data.
Normalized Attack (Decay): Slope form beginning (end) to the MxNP.
Attack (Decay): Slope from beginning (end) to Max position (in frames).

Table 1. Object-level descriptors

annotate the noisy objects obtaining a total of 208 "*noisy*" sound objects.

2.1.2 Audio Descriptors

To characterize each sound object we first compute more than 90 frame-level audio descriptors. Then we compute several object-level descriptors to extract amplitude-related and time-related features from the time series of audio frames. For each frame-level descriptor, we compute the whole set of object-level features depicted in table 1. Thus, we obtain more than 1.400 descriptors for each sound object. A more detailed explanation of this process can be found in [10].

2.1.3 Classification Experiments

Once we have properly labeled databases (i.e. PercussionDB and NoiseDB) we use them to train several supervised classification algorithms.

For classification experiments we first perform a Correlation-based Feature Selection [11] (CFS) on each database to retain the most informative features (those with low intracorrelation and, at the same time, high class- correlation). Then we train both Support Vector Machines (SVM) and Logistic Regression [12] (LR) algorithms on each binary class problem (i.e percussive vs. non-percussive and noisy vs. non-noisy) using 10-fold cross validation in WEKA [1] .

Since we obtain quite similar results from SVM and LR, we adopted the later for simplicity. See table 2 for an overview on classification results. From the evaluation of the classification experiments, we can conclude that the model for the *percussive* category achieves good results (F-measure of 0.71) using a data set of more than 7.300 instances. We consider that this model is sufficiently general for classifying new *percussive* sound objects. Regarding the *noisy* model we also obtain good classification results (F-measure of 0.88). Although it would be interesting to have more examples to train the model we consider that this model is quite representative for *noisy* sound objects.

[1] `http://www.cs.waikato.ac.nz/ml/weka/`

Class	# instances	# descriptors	F-measure
P vs. N-P	7.380	74	0.71
N vs. N-N	416	38	0.88

Table 2. Classification results for percussive vs. non-percussive (P vs. N-P) and noisy vs. non-noisy (N vs. N-N) classes

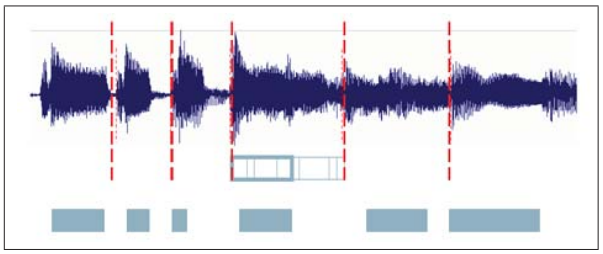

Figure 2. Audio onset segmentation (dashed lines), variable window length segmentation (empty squares), and sound object candidates (solid squares).

At the end of this process we have two LR models, one in charge of detecting *percussive* sound events and the other in charge of detecting *noisy* events. Since the LR algorithm outputs a class label and its corresponding probability measure, we store this probability to be used as a measure of confidence in the prediction.

2.1.4 Candidate Detection

In order to detect the *percussive* or *noisy* sound objects (and their confidence values) we need to evaluate all possible sound segments against our classification models. For every new song we compute its onsets and define sound segment boundaries as inter-onset intervals (see figure 2).

After the segmentation step we compute all possible candidate segments between two onsets, using a frame resolution of 512 samples. Additionally, we use category-specific rules in order to reduce the number of candidates and improve the performance of the classification. For the *percussive* sound category, the possible candidate segments must start at the onset position. Conversely, for the *noisy* category, candidates are not allowed to start at onset position in

order to avoid impulsive segments, and a minimum length is required so that they can be perceived as stable sounds.

At the end of the process, we obtain a list of sound object instance candidates and their confidence value. We also compute some additional object-level descriptors to be used in the assisted instance selection process described next. For *percussive* objects we compute their attack, decay and magnitude values and for the *noisy* category we compute spectral energy values for high, mid and low frequency bands.

3 ASSISTED INSTANCE SELECTION

3.1 Overview

The main feature of the selection prototype is to discover sound objects in a song collection. Given a song collection, we compute a list of candidate objects for each song. When a song is loaded in the main panel, the user can play the candidates, which are highlighted in different colors for *percussive* and *noisy* categories. Also, the user may restrict the found sound objects by setting a threshold, so that only those sounds with a confidence value above that threshold will be highlighted. Additional filtering can be done according to specific characteristics of each sound category. For example, for *percussive* sound objects the user might set values for attack, decay and magnitude and, according to the selected parameters, a triangular shape is drawn (see figure 3). When applying this filter, only those candidates whose attack, decay and magnitude values are within the ranges defined by the corresponding parameters and the tolerance slider are highlighted. *Noisy* sound objects may be filtered according to the distribution of energy in the spectrum. A dual slider allows specifying the relative proportion of high, mid and low frequency energy. A second slider specifying the desired total absolute level of energy. A rough representation of the target spectrum shape defined by these parameters is also displayed.

Sound object candidates may be added to a list using a context menu. The list represents the selection of sound objects that the user can export to the performance environment (section 4).

Additionally, a list of similar objects in the whole song collection is computed in advance for each candidate, using Euclidean distance. A second list shows the most similar objects for the currently selected candidate.

4 PERFORMANCE ENVIRONMENT

The performance tool consists in a mosaicing system that concatenates sound objects previously selected using the exploration prototype. It is composed of two separate modules: the graphical user interface and the audio engine. The interface displays information about the current status of the synthesis process, and sends sound object descriptors to the audio engine through Open Sound Control [13]. The audio engine receives the list of descriptors for each step of the sequence and uses them to select the appropriate sound object for that step.

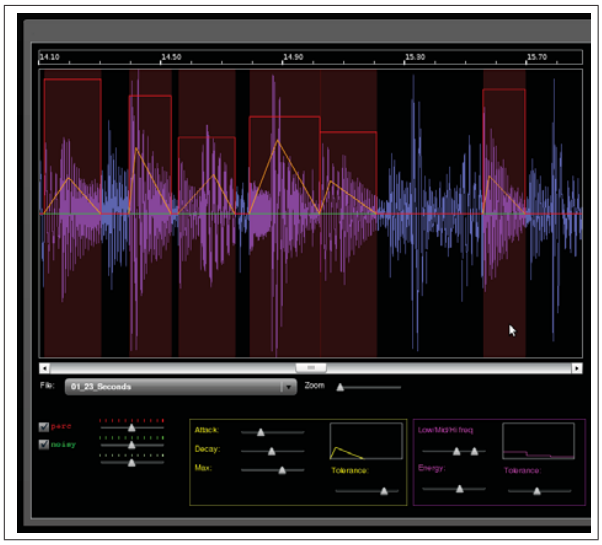

Figure 3. Interface of the exploration prototype. Three sound objects are identified. Amplitude envelope for the percussive objects(red) are represented with triangles.

4.1 Interface

The interface consists of three hierarchical levels: *Composer*, *Sequencer* and *Browser*. Each level can be also related to a different temporal scale: song / performance (*Composer*), loop (*Sequencer*) and sound object (*Browser*). Each *sequencer* has a variable number steps (e.g. sixteen steps in figure 4), and for each step the user can browse and select sound objects of a given category to compose a loop.

In the *Browser*, the user can listen to the sound objects of a given category in a specific sound objects set (previously stored with the selection tool). Typically, the number of objects for one category will be under 100 instances. Sound objects can be scattered according to some signal descriptor (e.g. mapping energy to distance to the center). The user can pre-listen to one sound object at any time before selecting it.

The *Sequencer* is a classical step sequencer. It has a circular shape, showing in real-time the current playing position. The number of steps is specific to a given loop and can be set by the user. Each *Sequencer* can also be rotated in order to modify its relative phase. Each step of the *Sequencer* can be filled with a sound object of one of the defined categories or left empty. The desired target object is selected through the *Browser*. A preset management system allows the user to store, load and unload sequences.

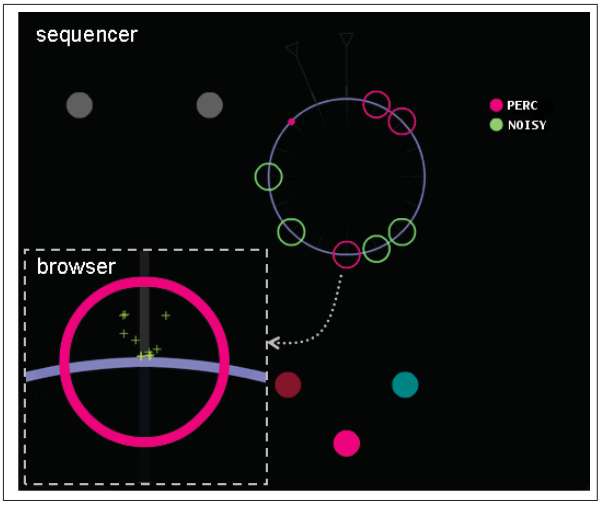

Figure 4. Synthesis interface, showing a *Sequencer* example, and a detailed view of the sound objects *Browser*. The *Composer* level (not shown in the figure) is the top level and can contain multiple *Sequencers*.

The top level is the *Composer*, which might contain one or several *Sequencers* synchronized to a global user-defined tempo. The user can create a composition by manipulating the *Sequencers* (changing location, muting, soloing) on the fly. When having several *Sequencers* in one *Composer* panel, the descriptors sent to the audio engine will depend on the position of the *Sequencers*. For each step, the sent descriptors will be a linear combination of the descriptors of the individual sound objects.

4.2 Audio mosacing engine

The audio mosaicing engine is responsible for the actual sound generation. It selects and concatenates samples from the internal sound bank obtained in the selection process. Each step in a loop is described by the vector of audio descriptors sent by the the interface module. The audio engine seeks the sound objects that best match the incoming descriptors.

One of the main motivations of using audio mosaicing is the flexibility in modifying the synthesized sounds by changing the content of the audio engine's internal sound bank. In this case, the synthesized output will mimic the structure and timbre characteristics of the sound objects used in the *Composer*, but using different sounds. Also, more complex interactions based on the mosaicing paradigm are possible using multiple *Sequencers* as described.

5 CONCLUSIONS AND FUTURE WORK

We have presented and implemented a proof-of-concept prototype for symbolic audio mosaicing. The main idea behind this system is to combine MIR and corpus-based synthesis techniques to obtain a new analysis/synthesis framework for music creation. The proposed application replaces, within the mosaicing paradigm, the target audio sequence by a symbolic sequence constructed with pre-defined sound categories.

We have implemented a fully working prototype considering two sound categories (i.e. *percussive* and *noisy* sounds) automatically detected by machine learning techniques. We believe that this user-assisted application is an engaging interface for audio mosaicing. This application gives the users an alternative to pre-defined sample banks by exploring their own music collections using high level categories.

The next logical step is to add some more categories and perform a user study in order to evaluate the concept. Ultimately, it should be possible for users to define their own sound categories. Concerning the segmentation step, we plan to extend the research by using other descriptors than onset detection. This approach might improve the generation of candidates for some categories.

6 ACKNOWLEDGMENTS

This work is partially funded by Yamaha Corp., Japan and the EU IST project Salero FP6-027122. The authors would like to thank Lucas Kuzma, Perfecto Herrera, Bee Suan Ong and Sebastian Streich.

7 REFERENCES

[1] Schwarz, D. (2004). 'Data-Driven Concatenative Sound Synthesis'. PhD Thesis. Université Paris 6 - Pierre et Marie Curie.

[2] Zils, A. and Pachet, F. (2001). 'Musical Mosaicing'. In Proc. Of the COST-G6 Workshop on Digital Audio Effects (DAFx-01), Limerick.

[3] Brossier, P. (2007) 'Automatic Annotation of Musical Audio for Interactive Applications,'Centre for Digital Music, Queen Mary University of London 2007

[4] Herrera, P. Klapuri, A. Davy, M. (2006) 'Automatic classification of pitched musical instrument sounds' in Signal processing methods for music transcription, Springer, 2006

[5] Peeters, G. (2003). 'A large set of audio features for sound description (similarity and classification) in the Cuidado project'. IRCAM.

[6] Bregman, A. S. (1990) 'Auditory scene analysis: the perceptual organization of sound', Cambridge, Mass. : MIT Press, 773.

[7] Schaeffer, P. (1966), 'Traité des objets Musicaux'. Paris. : Seuil.

[8] Roads, C. (2001), 'Microsound'. Cambridge, Mass. : MIT Press, 409.

[9] Gillet, O. and Richard, G. (2006) 'ENST-Drums: an extensive audio-visual database for drum' ISMIR, 156-159.

[10] Haro, M. (2008) 'Detecting and Describing Percussive Events in Polyphonic Music', Master Thesis, UPF, Barcelona.

[11] Hall, M. (2000) 'Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning', Proc. 17th International Conf. on Machine Learning, 359–366.

[12] le Cessie, S. and van Houwelingen, J (1992) 'Ridge Estimators in Logistic Regression', Applied Statistics, 41(1):191-201.

[13] Kuzma, L. (2008) 'An Interface for Sequencing with Concatenative Sound Synthesis', Master Thesis, UPF, Barcelona.