FROM *SOUND* SAMPLING TO *SONG* SAMPLING

Jean-Julien Aucouturier, Francois Pachet, Peter Hanappe
SONY CSL Paris
6, rue Amyot, 75005 Paris France.

ABSTRACT

This paper proposes to use the techniques of Music Information Retrieval in the context of Music Interaction. We describe a system, the SongSampler, inspired by the technology of audio sampling, which automatically samples a song to produce an instrument (typically using a MIDI keyboard) that plays sounds found in the original audio file. Playing with such an instrument creates an original situation in which listeners play their own music with the sounds of their favourite tunes, in a constant interaction with a music database. The paper describes the main technical issues at stake concerning the integration of music information retrieval in an interactive instrument, and reports on preliminary experiments.

1. INTRODUCTION

Music information retrieval research so far has not been much concerned with building interactive music systems. Systems which rely on a real-time music performance generally use it as an input front-end for a search, in a one-way approach which doesn't allow any subsequent interaction. In *Query by Humming* (QbH, [1]), the user sings a melody, and audio files containing that melody are retrieved. However, QbH does not exploit the resulting songs to respond to the original musical expression of the user, like e.g. improvising jazz musicians quoting or mimicking each others ([2]). Similar paradigms like *Query by Rhythm* ([3]) or *Query by Timbre* ([4]) share the same drawback.
Tzanetakis in [5] proposes an alternative browsing environment which offers a direct, continuous sonification of the user's actions. For instance, changing the target value of a query on tempo from 60 to 120 would morph the current song into a faster one, whereas traditional settings would require the user to press a "submit" button, which would stop the former song, and trigger the next one. While this is one step towards a seamless interaction with a music database, the system still offers no expressive control on the music as a music instrument would do. It remains a sophisticated jukebox.

Interactive music systems propose ways of transforming in real time musical input into musical output. Such responsiveness allows these systems to participate in live performances, either by transforming the actual input or by generating new material according to some analysis of the input. Musical interactive systems have been popular both in the experimental field [6] as well as in commercial applications, from one-touch chords of arranger systems to the recent and popular Korg Karma synthesizer [7]. While some interactive systems, referred to in [6] as "sequenced techniques", use pre-recorded music fragments in response to the user's input, these sequences are usually predetermined, and their mapping is predefined (triggered by e.g. dynamics, or specific notes, etc.). With the very large quantity of music available on personal computers, comes the fantasy of an interactive instrument able to explore any music database, responding to the user's input with automatically selected extracts or samples.
This paper describes a system, the SongSampler, which is an attempt at combining both worlds of music interaction and music information retrieval. Using techniques inspired by audio sampling, we propose to automatically produce a music instrument which is able to play the same sounds as an arbitrary music file. The SongSampler uses MIR techniques such as content descriptors or similarity measures in order to select the song(s) to sample in a music database. The resulting instrument influences the user's performance, which, in turn, is analyzed with MIR tools to produce queries and modify the sampler's setting. Playing with such an instrument creates an original situation in which listeners play their own music with the sounds of their favourite tunes, in a constant interaction with a music database.

2. AUTOMATIC SAMPLING

Audio Sampling ([8]) is the process of mapping arbitrary sounds (or samples) to a music instrument. Each note played on the instrument (typically a MIDI keyboard) triggers an audio sample corresponding to the pitch of the key (e.g. a C-60) and its loudness. Such digital samplers have been introduced in the 80s, and have been very popular thanks the realistic effect achieved by this technique: virtually any sound can be produced by a sampler, by definition. However, the creation of a particular setup for a sampler (e.g. a piano sound) is known to be a tedious and difficult task: samples must be first recorded (e.g. from existing, real music instruments), then assigned to the keys of a MIDI instrument. Details concerning the actual triggering of sounds must be carefully taken into account, such as the loop of the sustain part of the samples until the key is released. This process of specifying a sound for a sampler is usually done by hand [9].

The core of the system described here consists in producing automatically setups for software-based samplers, so that the sounds triggered correspond to the actual sounds present in a given music title. Consequently, a sampler setup produced from a given audio file will produce an instrument that plays the same sounds as the original audio file.

A popular format for what is referred to here as a "sampler setup" is the SoundFont® file format [10]. A typical SoundFont® file contains :

- Samples, which can be digital audio files, e.g. .wav, or loaded from ROM on the wavetable device. Samples have the option of being looped.
- Generators, which control properties of a sample such as initial pitch and volume as well as how these parameters are affected over time.
- Instruments, which use one or more samples combined with effects generators to create a sound producing device.

The automatic extraction of a SoundFont®-like setup from an arbitrary audio file thus requires to :

- analyse (i.e. segment) the audio data to extract "meaningful" samples in the music
- extract high-level audio descriptors from the samples to select automatically the most appropriate samples to use for a given context. Notably, detect the pitch of each sample so it can be mapped to an instrument note
- detect parts of the segment that can be looped automatically (or, as it turns out, do more complex processing to time-stretch the samples)

Each of these 3 steps have received a vast number of technical solutions, which we will not review here. The SongSampler has a modular architecture, in which any suitable algorithm can fit.

However, we are dealing here with arbitrary music files, of arbitrary complexity, e.g. polyphonic, containing percussion instruments, effects, etc. In the next paragraphs, we propose a number of algorithms which we have designed specifically to fit this particular application context.

2.1. Multiscale segmentation

The aim of the segmentation algorithm is to extract samples that can act as well-defined musical events, i.e. which have a salient note or percussion played by some instrument(s) in the foreground, and a background based on the global sound of the sampled song. For instance, a typical sample from the song "Yesterday" by "The Beatles" ([11]) would be Paul McCartney singing "..day…", with the song's original background of acoustic guitar, bass and violin. The song is cut in the time domain, which means that each sample contains several instruments playing at the same time, and not separated individual tracks.

Typical segmentation algorithms ([12,13,14]) first computes a set of features from the signal cut into frames, and then detect the segment boundaries by looking for abrupt changes in the trajectory of features. In this work, we look for the energy variations of the signal. The signal is cut into frames (2048 points at 44100Hz), and for each frame, we compute the short–term spectrum. The spectrum itself is processed by a Mel filterbank of 20 bands. Each band's energy is weighted according to the frequency response of the human ear, as described e.g. in [14]. Finally, the energy is summed across all bands. Change detection is done by smoothing the energy profile by a zero-phase filtering by a Hanning window of size S_w, and looking for all the local maxima of the smooth version. The segment boundaries are the deepest valleys in the raw energy profile between 2 adjacent peaks in the smooth profile.

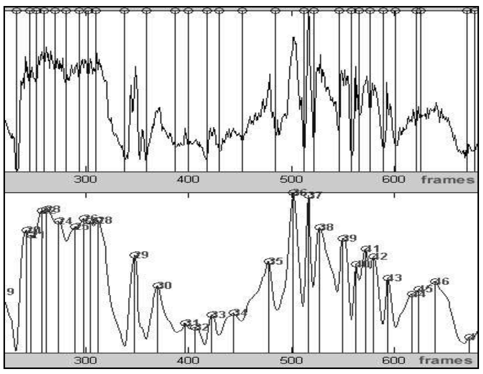

Figure 1A: Segmentation of an extract of "The Beatles - Yesterday". (Top) Segmented energy profile using a small Sw (150ms) : short events (right) get properly detected, while larger events (left) get oversegmented. (Bottom) Corresponding smoothed energy profile, used for peak detection.

While this scheme is effective for simple, percussive music, we observe that for non percussive, richer polyphonic music, the quality of the segmentation depends on the choice of S_w. In large events such as a sung note lasting for several seconds (e.g. the final "-day" in "Yesterday"), there may be several small peaks of energy corresponding to the other instruments playing in the background (e.g. a succession of chords played on the guitar). With a small S_w, all these peaks would be segmented, and the corresponding atomic event for the sampler application would be cut into several short identical notes (see Figure 1A). With a large S_w on the other hand, short meaningful events like isolated guitar chords get missed out (Figure 1B).

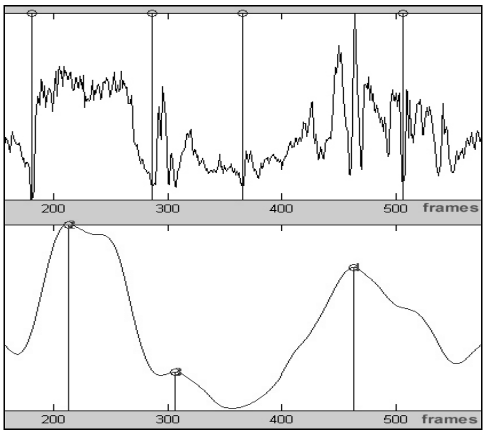

Figure 2B: Segmentation of an extract of “The Beatles - Yesterday”. (Top) Segmented energy profile using a large Sw (1s) : large events (left) are appropriately recognized, however smaller events (right) are missed out. (Bottom) Corresponding smoothed energy profile

Therefore we propose a multiscale segmentation algorithm, which adapts the size of the convolution window to the local shape of the energy profile. More precisely, we compute the STFT of the energy profile on a running 2-second window (with 90% overlap). As the energy profile is sampled using 50% overlapping, 2048 point frames (i.e. 43Hz), the FFT describes the frequency content between 0 and 20Hz, with a frequency resolution finer than 1Hz. We select the predominant local periodicity of the profile as the barycentre point (spectral centroid) of the spectral distribution within each frame :

$$SC = \frac{\sum_k kS(k)}{\sum_k S(k)}$$

where S is the magnitude spectrum of a frame. We then smooth the energy profile using a Hanning window size S_w equal to the inverse of the centroid of the corresponding FFT frame (to ensure continuity, Hanning window coefficients are normalized so they sum to one regardless of their length).

Figure 2-Bottom shows the SFFT of the energy profile used in Figure1. Large events correspond to low frequencies in the energy profile, i.e. small centroid frequencies in the spectrogram (order of 1Hz). Consequently, these zones get smoothed with large Hanning windows (order of 1 sec.). On the other hand, short events in the energy profile correspond to higher frequency content, higher centroids, and smaller windows size (order of 200ms). Figure 2-Top illustrates the corresponding multiscale segmentation, which preserves large, noisy events as well as short, high amplitude ones.

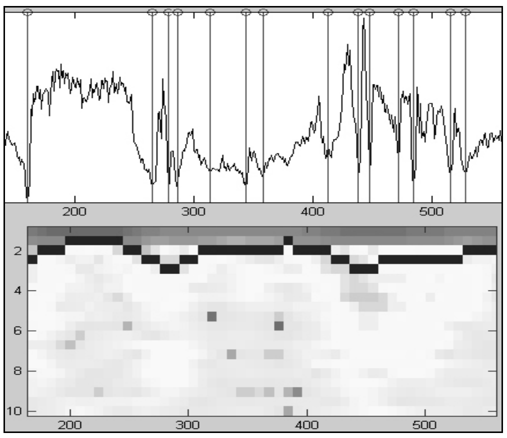

Figure 3: (Top) Multiscale segmentation of the same extract, using an adaptive convoluation window size : large windows on the left, and smaller windows on the right. (Bottom) Corresponding spectrogram of the energy profile, super-imposed (in black) with the spectral centroid of each frame, used to determine the windows size

2.2. Automatic Extraction of High-Level Descriptors

Each of the segments generated by the previous step constitutes a note, which will be mapped on the instrument. The mapping of the samples is based on a number of high-level descriptors automatically extracted for each sample. The SongSampler relies on the EDS (Extractor Discovery System) [16] to generate such descriptors. EDS is a system based on genetic programming which is able to automatically generate arbitrary perceptual descriptors, given only a test database and corresponding perceptive tests, by discovering and optimizing adapted features and machine learning models. In the current implementation of the SongSampler, EDS was used to generate a descriptor of harmonicity (see [16]). This descriptor is used in the SongSampler to filter out samples corresponding to non harmonic events (e.g. a snare-drum hit). However, there is an infinity of such descriptor/mapping possibilities, depending on the application context. For instance, if the mapping is done on a digital MIDI drum kit, we could use the EDS to generate a drum sound classifier (see e.g. [17]) in order to affect the percussive samples to the right pads. One key advantage of using the EDS is that the descriptors are by construction adapted to the type of samples used in the SongSampler. We expose further possible uses for the EDS in section 5.

2.3. Salient pitch detection

The SongSampler maps the samples to the keyboard key corresponding to their pitch, so a melody can be played. In the current implementation of the system, the pitch descriptor was not generated by the EDS, but rather was manually designed. We describe this specific algorithm in this section. Traditional monophonic, mono-instrument pitch detection algorithms are based on peak detection in the signal's spectrum. In more complex sound sources, state-of-art approaches look at spectral periodicities, either by a “comb-filter” sample-summing approach ([18,19]), or by a FFT approach ([20]). Such analyses done with a high frequency resolution have the advantage of yielding precise estimates of fundamental frequency, which e.g. can be used to study fine properties of instrument timbre. However, such high resolutions come at the expense of rather complex algorithmic provisions to cope with a number of signal complexities like inharmonicity or vibrato : inharmonicity factor, subband processing in [19], tracking of partial trajectories in [20].

In our context, we are only interested in a rough pitch estimate, with limited precision both in frequency (precise to the semi-tone, which is the MIDI resolution) and time (one salient pitch estimate for each sample). We apply a STFT to the signal, and converts the frequency scale into a midi pitch scale, using a bank of band pass filters, on per midi pitch from C0 to C7 with the width of one semitone. The remaining of the algorithm thus deals with a much simpler symbolic signal, a “pitchogram” which represents the energy of each potential midi pitch in the signal.

Figure 3f shows such a pitchogram (averaged over time) for a sample from “Yesterday”, for midi pitches ranging from 30 (F#1) to 90 (F#6). The pitchogram is then looked for local maxima, each of which constitute a pitch hypothesis.

Each pitch hypothesis receives a harmonic score according to the presence or not of another pitch hypothesis at the position of its harmonics :

- at octave (midi pitch+12)
- twelfth(octave+fifth) (midi pitch+19)
- fifteenth(two octaves) (midi pitch+24)
- major seventeenth (two octaves+major third) (midi pitch+28), etc.

The harmonic score is computed as the weighted sum of the harmonics’ energy. Figure 3 shows the 5 best-score pitch hypotheses with their harmonics. Each of the harmonics’ energy is the energy of the corresponding pitch hypothesis as found in the pitchogram.

Additionally, we reinforce the scores of pitch hypothesis which uniquely explain one of their harmonics, e.g. the first harmonic in Figure 3-b. Note that the first harmonic of 3-b doesn’t become a pitch hypothesis and doesn’t appear as a new plot in Figure 3, because it has no harmonics, and thus receives a minimal score.

Finally, we pick the pitch hypothesis with the best score. In Figure 3, the best hypothesis is 3-c, which both includes a uniquely explained harmonic, and has an important harmonic score. The corresponding pitch is the midi pitch of its fundamental, i.e. 52 (E3). Note that others strategies are possible to cope with polyphony (e.g. choosing all hypotheses whose score exceeds a certain threshold)

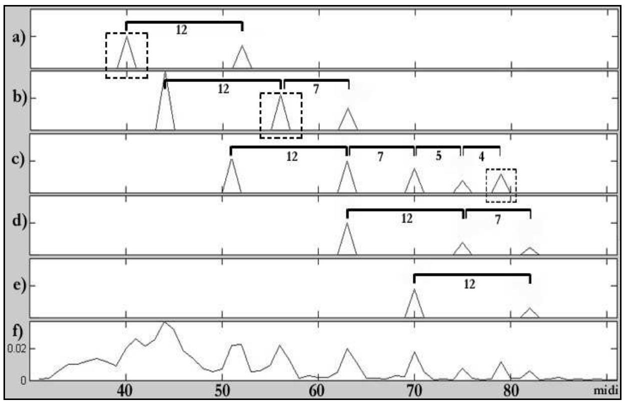

Figure 4: the 5 main pitch hypotheses (a-e) corresponding to the time-averaged pitchogram (f). Brackets show the harmonic relations between the partials of each hypothesis. Dotted-line squares highlight the partials which are explained by a unique pitch hypothesis.

We have conducted a small evaluation in order to fine-tune the weights involved in the computation of the harmonic score. The test database has 50 samples extracted automatically from 3 pop songs, and the target pitch were determined manually. We test 2 parameters : the number of harmonics to be considered (nh), and the weights of the harmonics, parameterized by exponential $\alpha - \beta \exp(\gamma.n)$ curves, with gamma ranging from –1 to 1. We observed that these parameters have little influence on the algorithm’s precision. The best precision (0.76) was obtained for nh=5 and gamma=0.4 (slightly decreasing weights). For better precision, harmonic weights could be adapted to specific instrument timbres.

2.4. Time stretching

The samples extracted by segmentation from the original song have the same duration as in the original song (e.g. 1.44 second for the above-mentioned “day” sample from “yesterday”, in the original recording in the Help album). However, when these samples are mapped on a music instrument, we want the duration to match the musician’s intention: notes can be shorter or longer than in the original song they were extracted from.

The process of modifying the duration of a note is called time stretching. Time stretching in traditional samplers is done by looping within the sustain part of the sample for the appropriate (longer) duration. This requires loop start and end points, which are usually found manually, and requires much trials-and-errors. Looping is well adapted for clean monophonic, mono instrumental samples. However, in our context of sampling arbitrary recording, with complex polyphony, background percussion, and “real-world” sound production like reverberation, this approach yields very poor results.

We time-stretch the samples using a technique know as phase-vocoder, which analyses the short term spectrum

of the signal and synthesizes extra frames to morph between the original signal's frames (e.g. adding an extra 50 millisecond every 50 milliseconds). The technique relies on a phase continuity algorithm called identity phase locking ([21]).
In our application context, many samples resulting from the segmentation algorithm described above are not ideally stable: while each sample is a coherent note (e.g. "day"), there are still minor events occurring in the background (e.g. softer notes of the guitar accompaniment), which creates discontinuities of timbre, energy, etc…

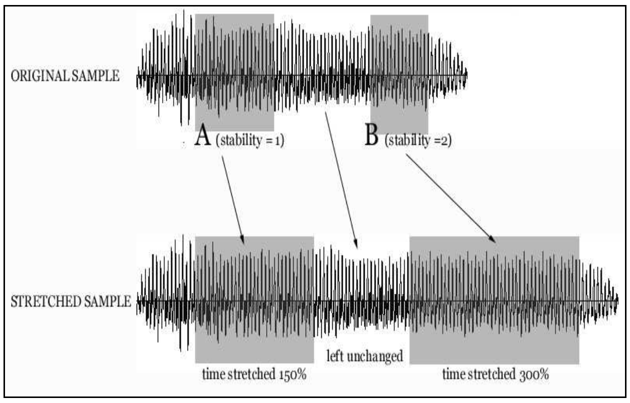

Figure 5: time stretching with stability zone weighting

If we apply the phase-vocoder to the whole sample, the algorithm would also stretch these transient events, leading to unrealistic, "slow-motion" sounds, known as transient smearing (e.g. guitar attacks lasting for too long). To avoid stretching zones of discontinuity in the signal, we first analyze each sample to find zones of spectral stability, using the EDS harmonicity descriptor (section 2.2). Each stable zone receives a stability score, and we only stretch these zones, by a stretch factor which is proportional to the zone's stability. (Figure 4)

3. ARCHITECTURE

Figure 5 describes the architecture of the system. The core of the SongSampler is implemented in Java. It is composed of 2 concurrent interacting processes, a player, and a sampler. The interaction occurring between the Sampler and the Player is at the center of the application we propose in this work, and is described in details in the next section. In this section, we only describe the nature and medium of the communication between the components.
The SongSampler relies on 2 other components:

- MidiShare [22] is a real-time multi-task MIDI operating system. MidiShare applications are able to send and receive MIDI events, schedule tasks and communicate in real-time with other MidiShare applications.
- Fluidsynth [23] is a real-time software synthesizer based on the SoundFont 2 specification. The SongSampler relies on fluidsynth to efficiently play the samples analysed by the Sampler.

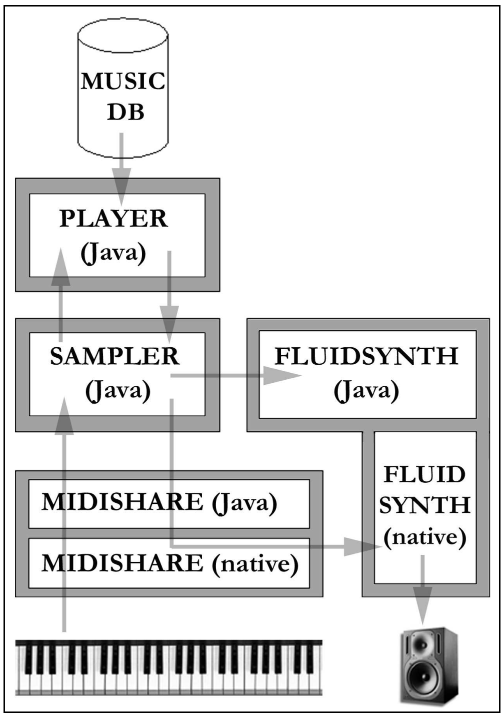

Figure 6: Architecture of the SongSampler, showing the 4 main components and their Java native interface.

Both Fluidsynth and the Sampler process are declared as MidiShare applications. MIDI messages coming from the user (e.g. through a MIDI keyboard) are routed via MidiShare to the Sampler.
After analysis, the Sampler forwards the midi messages to Fluidsynth, which triggers the corresponding samples and renders them to the audio device.
The Player communicates with a music database. It can autonomously query the db according to various criteria, which are inferred from the current state of the system and the user's actions. It can also play a song, and interact with the sampler by proposing new songs to sample.
The Sampler performs the analysis described in section 2 (possibly prepared in non-real time), assigns samples to Fluidsynth, reacts to midi messages coming from the user (e.g. midi program changes), and interacts with the player by forwarding the incoming user actions.
Fluidsynth is piloted in Java from the Sampler using Java native interfaces (JNI) which were developed for this project. Both Fluidsynth and the sampler communicate with MidiShare via JNI which were developed by Grame ([22]).

4. PRELIMINARY EXPERIMENTS

The SongSampler can be used in a variety of playing/listening modes, which results from the many possibilities of interaction between the Player and the Sampler process. In this section, we describe our preliminary experiments with the system.

4.1. Turn Taking

Figure 6 illustrates a first mode of interaction, where the user and the system's music player take turns.
In this setting, a song is chosen (e.g. our followed example, "Yesterday"), and analysed by the sampler : the song is segmented into meaningful notes and samples are analysed for pitch. The Sampler then maps the samples in fluidsynth in such a way that the samples are changed after each pressed key, and iterate in time order. For instance, pressing a key 3 times would trigger the 3 samples "yes-", "-ter-", "-day". The samples are matched to every note on the keyboard, but keep a relation to their pitch in the original signal. For instance, the "yes" sample in "yesterday" has an original detected pitch of "G3". If the user triggers this sample by pressing the "F3" key, fluidsynth automatically pitch-shifts[1] the sample to match the wanted pitch, i.e. the sample will be played one tone lower than in the original signal.
When the mode is started, the Player starts playing the song normally. At anytime, the user can press a note on the keyboard. When the note is received, the Player stops, and the Sampler seamlessly triggers the sample corresponding to the current position in the song. The user keeps the lead until he stops playing, i.e. a given time has passed since the last played note, at which point the Player starts playing the song again, at the position of the last triggered sample. Moreover, the new behavior of the Player depends very closely on the user's performance :

- bpm interaction: as the user plays with the Sampler, the bpm of its performance is tracked, using a real-time beat tracker ([24]). Upon restart of the Player, the song is time-stretched (using the technique exposed in section 2.3) to match the bpm of the user's performance.
- pitch interaction: the performed pitch of the last triggered sample is compared to the sample's original pitch, and upon restart, the Player pitch-shifts the song to match the transposition interval (using a phase-vocoder like for time-stretching).

Using these simple mechanisms, the user can play new melodies with the sounds of the original song. In turn, the original song is modified according to the user's performance.

Example on Yesterday
Figure 6 is a transcription of a turn-taking interaction between a user and the song "Yesterday" by The Beatles. The example starts at the second line of the first verse. The Player plays the music corresponding to the melody {D,D,C,Bb,A,G} at a normal rate. At this point (#1), the user takes the lead, and press the {Bb} key, which is the original pitch of the next sample in queue (Paul McCartney singing "here"). Then the user starts deviating from the melody with an ascending pattern {C, C#,D}. This successively triggers the samples "to", "stay", "oh", at a different, increased pitch than in the original song. Simultaneously, the user increases the tempo from the original bpm of 100 quarter notes per minutes to an "allegro" tempo of 140 bpm. After triggering the "oh" sample, the user stops playing. The Player now takes the lead (#2), and restarts the original song at the position of the next sample ("I"). As the last triggered sample is pitched a perfect fifth higher than the original pitch, the original song is pitch shifted by a fifth, which creates a feeling of continuity with the user's phrase. At the same time, as the user bpm is higher than the bpm of the original song, the song is time-stretched to match the new tempo.

[1] In the current version of fluidsynth, pitch-shifting is done by resampling.

4.2. Collaborating

Figure 7 illustrates another mode of interaction, in which the Player loops on a section of the original file (the introduction of Yesterday). In the mean time, the Sampler processes another section of the song (Paul's voice on the first verse and chorus). The user is then able to improvise a phrase with the accompaniment of the original song. This mode is an interesting exercise for musicians, as they have to make the best of the offered accompaniment, e.g. in Figure 7, although the song is in the key of F major, the guitar vamping on the introduction does not play the third degree (A). This leaves an ambiguity on the major/minor character of the song[2], which is exploited by the user (alternation of minor third Ab and major third A).

4.3. Exploring the database

The Sampler and the Player need not process the same song. For instance, in the previous mode, the Sampler may query a song in the database according to any metadata, e.g. instrument = piano. Consequently, the user would play on top of Yesterday's guitar comping with e.g. the piano sound of a Beethoven sonata. Moreover, the Sampler thread can listen to the end of each of the user's phrases, and change the synth's settings with another piano song so that the user explores all piano songs/sounds in the database.

5. FURTHER WORK

Many standard techniques of Music Information Retrieval can be integrated in the interactive system describe above. This section lists some of the possibilities we envision :

[2] This ambiguity on the song's key is actually largely exploited in the melody of original song, as analysed e.g. in [25]

- Editorial metadata on the songs : play only with samples from the Beatles, or only “Brasilian sounds”
- High-Level Descriptors on the samples : Perceived Energy ([16]) (samples with high energy are triggered when keys are pressed with a high velocity, while softer samples are triggered for lower velocities), Instrument ([27]) (play only samples of acoustic guitar). As described in section 2, the SongSampler relies on the EDS to automatically generate such descriptors.
- Query by Melody ([1]) The user plays the melody of “Michelle” with the Samples of “Yesterday”, and the Player replies with “Michelle”
- Query by Harmony ([28]): The player selects a song whose harmony matches the phrase being played
- Query by Timbre ([4]): the SongSampler may interact with the user by proposing (i.e. either play or sample) songs which sound similar to the song currently being played/listened to.
- Structural Analysis of the songs ([29]) : sections to sample or to loops (see 4.2.) may be automatically detected.

6. CONCLUSION

This paper describes a system, The SongSampler, which automatically samples a music file in order to produce an instrument that plays the same sounds as the original audio file. This is an attempt at mutually enriching both worlds of Music Information Retrieval and Music Interaction. The process of interacting with a music collection creates a novel immersive browsing experience, in which queries are not necessarily formulated by the user, but are rather inferred from the user’s actions. On the other hand, playing with such a MIR-enabled interactive instrument enhances the feeling of appropriation by letting listeners play their own music with the sounds of their favorite tunes.

7. ACKNOWLEDGEMENTS

The authors wish to thank Tristan Jehan for interesting discussions on the topic of segmentation. The scores in Figure 6 and 7 were obtained with Guido NoteViewer (http://www.noteserver.org).

8. REFERENCES

[1] Hu, N. and Dannenberg, R. “A Comparison of Melodic Database Retrieval Techniques Using Sung Queries”' in *Joint Conference on Digital Libraries*, New York: ACM Press, (2002), pp. 301-307.

[2] Monson, I. *Saying Something: Jazz Improvisation and Interaction*, Chicago Studies in Ethnomusicology, The University of Chicago Press, 1996

[3] Chen, J. and Chen, A. “Query by Rhythm: An Approach for Song Retrieval in Music Databases”, *Proceedings Eighth International Workshop on Research Issues in Data Engineering, Continuous-Media Databases and Applications*, (1998), pp. 139--146.

[4] Aucouturier, J.-J. and Pachet F., “Improving Timbre Similarity: How high is the sky?”. *Journal of Negative Results in Speech and Audio Sciences*, 1(1), 2004.

[5] Tzanetakis, George, Ermolinskyi, Andreye, and Cook, Perry "Beyond the Query-by-Example Paradigm: New Query Interfaces for Music Information Retrieval" In *Proc. Int. Computer Music Conference (ICMC)*, Gothenburg, Sweden September 2002

[6] Robert Rowe, *Interactive Music Systems*, MIT Press Cambridge, Massachusetts 1993.

[7] Karma music workstation, Basic guide. Korg Inc. Available : www.korg.com/downloads/, 2001.

[8] Roads, C. “Sampling Synthesis” in *The Computer Music Tutorial*, pp.117-124, MIT Press, Cambridge, Massachusetts 1995.

[9] Duffel, D. *The Sampling Handbook*, Backbeat Books, May 2004.

[10] SoundFont® is a registered trademark by E-MU. Specifications available : www.soundfont.com/documents/sfspec21.pdf

[11] “Yesterday”, John Lennon/ Paul McCartney, in “Help”, Parlophone, 1965

[12] Tzanetakis, G., and Cook, P., “Multifeature Audio Segmentation for Browsing and Annotation”, *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, Oct 1999.

[13] Rossignol, S. et al. “Feature Extraction and Temporal Segmentation of Acoustic Signals”, in *Proceedings of the International Computer Music Conference*, 1998.

[14] Pampalk, E., Dixon, S. and Widmer, G. “Exploring music collections by browsing different views”, in *Proceedings of the International Symposium on Music Information Retrieval* (ISMIR), Paris, France 2003

[15] Foote, J. and Cooper, M. “Media segmentation using self-similarity decomposition”, *in Proc. SPIE Storage and Retrieval for Multimedia Databases*, Vo. 5021, January 2003.

[16] Pachet, F. and Zils, A. “Evolving Automatically High-Level Music Descriptors From Acoustic Signals.” *Springer Verlag LNCS*, 2771, 2003.

[17] Gouyon, F., Pachet, F. and Delerue, O., “On the use of zero-crossing rate for an application of classification of percussive sounds”, in *Proceedings of the Digital Audio Effects (DAFx'00) Conference*, Verona, Italy 2000

[18] Klapuri, A. et al. “Robust multipitch estimation for the analysis and manipulation of polyphonic musical signals”, in *Proceedings of the Digital*

Audio Effects (DAFx'00) Conference, Verona, Italy 2000

[19] Tadokoro, Y., Matsumoto, W. and Yamaguchi, M. “Pitch detection of musical sounds using adaptive comb filters controlled by time delay”, in *proc. ICME*, Lausanne, Switzerland, 2002..

[20] Marchand, S. “An Efficient Pitch-Tracking Algorithm Using a Combination of Fourier Transforms.” In *Proceedings of the Digital Audio Effects (DAFx'01) Conference*, pages 170-174, Limerick, Ireland, December 2001.

[21] Jean Laroche and Mark Dolson "New Phase Vocoder Technique for Pitch-Shifting, Harmonizing and Other Exotic Effects". *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. Mohonk, New Paltz, NY. 1999.

[22] Orlarey, Y. and Lequay, H. MidiShare: a Real Time multitasks software module for MIDI applications, in *Proceedings of the International Computer Music Conference*, Computer Music Association, San Francisco, pp. 234-237, 1989.

[23] Fluidsynth website http://www.fluidsynth.org

[24] Scheirer, E. D. (1998). “Tempo and beat analysis of acoustic musical signals,” *Journal of the Acoustical Society of America*, 103(1), 588–601.

[25] Pollack, Alan W. (1993), Notes on "Yesterday". *Notes on ... Series* no. 74, 1993. Available : http://www.recmusicbeatles.com

[26] Perfecto Herrera, Geoffroy Peeters, Shlomo Dubnov “Automatic Classification of Musical Sounds” *Journal of New Musical Research* 2003, Vol. 32, No. 1, pp 3-21

[27] Pickens, J. et al. "Polyphonic Score Retrieval Using Polyphonic Audio Queries: A Harmonic Modeling Approach," *Journal of New Music Research* 32(2):223-236, June 2003.

[28] Geoffroy Peeters, Amaury La Burthe, Xavier Rodet “Toward Automatic Music Audio Summary Generation from Signal Analysis”, in proc. International Conference on Music Information Retrieval (ISMIR), Paris (France) October 2002

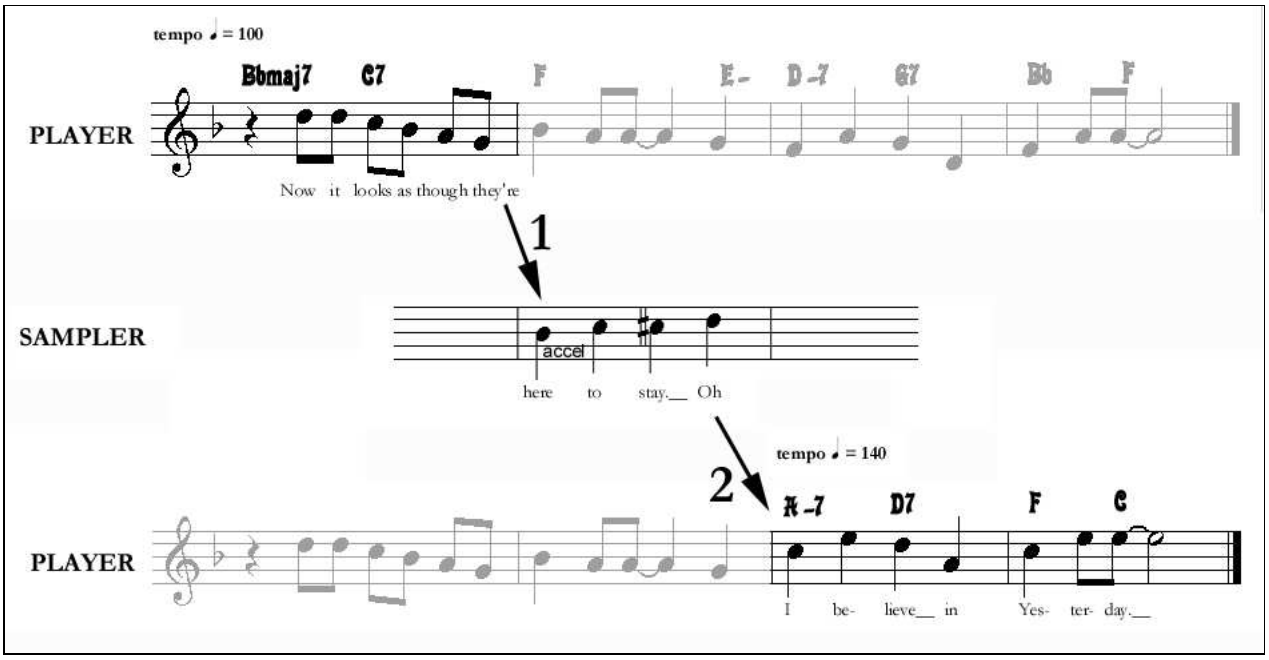

Figure 7: Turn Taking on “Yesterday – The Beatles”.

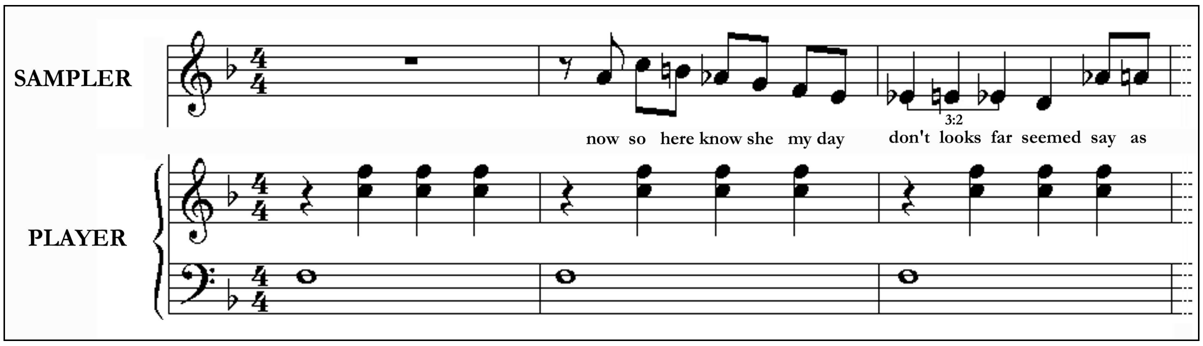

Figure 8: Playing on top of the introduction of “Yesterday”, with samples from the verse and chorus