VIRTUALBAND: INTERACTING WITH STYLISTICALLY CONSISTENT AGENTS

Julian Moreira
Sony CSL
julian.moreira.fr@gmail.com

Pierre Roy
Sony CSL
roy@csl.sony.fr

François Pachet
Sony CSL
pachetcsl@gmail.com

ABSTRACT

VirtualBand is a multi-agent system dedicated to live computer-enhanced music performances. VirtualBand enables one or several musicians to interact in real-time with stylistically plausible virtual agents. The problem addressed is the generation of virtual agents, each representing the style of a given musician, while reacting to human players. We propose a generation framework that relies on *feature-based interaction*. Virtual agents exploit a style database, which consists of audio signals from which a set of MIR features are extracted. Musical interactions are represented by directed connections between agents through these features. The connections are themselves specified as mappings and database filters. We claim that such a connection framework allows to implement meaningful musical interactions and to produce stylistically consistent musical output. We illustrate this concept through several examples in jazz improvisation, beatboxing and interactive mash-ups.

1. INTRODUCTION

Collective improvisation is a group practice in which several musicians contribute their part to produce a coherent musical whole. Each musician typically brings in musical knowledge, taste, and technical skills, more generally a *style*, which makes him or her unique and recognizable. However, good improvisations are not only about putting together the competence of several individuals. Listening and interacting to each other is crucial, as it enables each musician to adapt to the global musical output in terms of rhythm, intensity, harmony, etc. The combination of individual styles with the definition of their interaction defines the quality of a music band. In short, group improvisation can be seen as principled interactions between *stylistically consistent* agents.

Many works have attempted to model and simulate the behavior of a real musician. A MIDI-based model of an improviser's personality is proposed in [6], to build a virtual trio system, but no explicit interactions between virtual agents and real musicians are proposed. Improtek [11] is a system for live improvisation that generates musical sequences built in real-time from a live source using concatenative synthesis. Improtek plays along with a musician improvising on harmonic and beat-based music, in the style of this musician, but interactions with him are based on feature similarity measures, thereby limiting the scope of man-machine interactions. The Jambot [4] infers in real-time various features from an audio signal and then produces percussive responses, following alternatively predefined behaviors, but the style of the Jambot itself is not clearly defined. Beatback [7] generates a MIDI signal based on rhythmic patterns and using Markov models. Besides the limitations of MIDI, interactions are limited to rhythmic elements only. This limitation also applies to the Chimera Architecture [3], a system that infers rhythmical information from an audio input, and generates percussive sounds depending on scenarios that fit with the current musical context. [9] presents a system that learns rhythmic patterns from drum audio recording and synthesizes musical variations from the learnt sequence. In addition to being dedicated to percussive sounds only, this system has no real-time live application. [16] presents an interactive music system driven by syncopation measurements, but also addresses rhythmical features only. [10] describes a system that interacts in real-time with a human musician, by adapting its behavior both on prior knowledge (database of musical situations) and parameters extracted during the current session, but this system forces the musician to manipulate a graphical interface during the improvisation, which makes the system hardly usable in a live performance for a solo musician.

In this paper we revisit the issue of musical interaction with stylistically consistent agents by taking a feature-based approach to the problem. We introduce VirtualBand (VB), a *reactive* music multi-agent system in which human musicians can control and interact with virtual musicians that retain their own style.

In VB, both human and virtual musicians are represented by agents that interact together through feature-based interactions. *Human agents* extract their features at every beat from the audio input of their corresponding musician. *Virtual agents* use features of audio *chunks* stored in a database. Interactions are modeled by *connections*, which are the core contribution of our approach. A connection is a directed link from a *master* (human or virtual) agent to a *slave* virtual agent. In reaction to a feature value provided by the master agent, the connection specifies to the slave agent which audio chunk to play from its database. VB can be seen as a reactive rather than deliberative multi-agent system [8]. Its potential lies in the seemingly infinite possibilities provided by feature interaction, as illustrated in this paper.

In Section 2, we describe the main components of VB. In Section 3, we illustrate feature interaction with several configurations of the system for jazz improvisation. Section 4 describes applications of VB in two other musical contexts: beatboxing and automatic mash-up.

2. SYSTEM DESCRIPTION

The core of VB consists of 1) a clock that establishes the tempo and sends notifications at each beat to every agent - we consider fixed tempos in the current version, 2) a set of agents that receives these notifications and generate audio and 3) an audio playback engine.

2.1 Agents

There are two types of agents: human and virtual agents, representing respectively actual and virtual musicians. Human agents are responsible for extracting acoustic features from the audio signal of real musicians in real-time. These features are the main controlling device for virtual agents via connections, as explained below.

Virtual agents are designed to play *in the style of* the musicians they represent. To build a virtual agent, we record the musician in a musical situation that fits with the targeted performance context. The recorded audio is stored in a *style database*, organized in musically relevant *chunks* (usually beats and bars). The musician is asked to play so as to fully express his musical style, i.e., cover a large range of musical situations (e.g., playing with different intensities, in different moods, staccato or legato, using various patterns). Of course, it is difficult to express one's style exhaustively, so style databases are limited to specific musical *situations*, e.g., defined by musical genre such as Bossa Nova, Swing, etc. and a tempo. Note that the subject of *individual style capture* is still in its infancy and further studies should refine this concept.

In the database each chunk is associated to a set a feature values. A set of MIR features are automatically extracted from the audio signal of the chunk, e.g., RMS, numbers of onsets, spectral centroid, harmonic to noise ratio, or chroma. Contextual features are also associated to each chunk such as the harmony. Note that the harmony can either be extracted automatically or be imposed by a chord sequence. During performance, each virtual agent uses concatenative synthesis [15] to generate music streams as seamless concatenations of database chunks.

2.2 Connections as a Combination of Mappings

A connection between two agents models the *intention* underlying a musical interaction between two musicians. Connections are directed from a master agent A_M to a slave agent A_S, and relate a master feature F_M of A_M to a slave feature F_S of A_S.

At every beat, the connection selects the audio chunk that A_S is going to play, among the available chunks of its associated database D_S. To perform this task, the connection applies two successive mappings: 1) a *feature-to-feature* mapping from the domain X_M of F_M to the domain X_S of F_S, and 2) a *feature-to-chunk* mapping from X_S to D_S.

2.2.1 *The feature-to-feature (f2f) mapping*

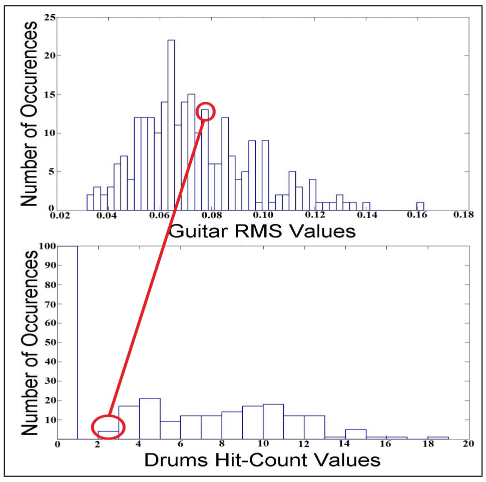


Figure 1. Illustration of a percentile mapping (in red) between two distributions. 1) RMS values extracted from a guitar track (top) and 2) hit-counts extracted from a drum track (bottom). Both tracks come from a recording of the jazz standard *Body and Soul*.

F_M and F_S may take their values in different domains (for instance integers or floating points) and with different value distributions. Normalizing the values of F_M and F_S allows to define a simple feature-to-feature (f2f) mapping, that maps any value x_M of F_M onto the same value x_M of F_S. However, with such a mapping, the distribution of the virtual agent's actual output doesn't match the distribution of the database, i.e., the virtual agent doesn't play consistently in the style of the musician it represents.

Instead we use a percentile-based f2f mapping, which preserves this distribution. If D is the domain of a variable x, the percentile function is defined by:

$$percentile(x, D) = \frac{|\{d \in D \mid d < x\}|}{|D|} \tag{1}$$

The f2f mapping is then defined as:

$$f2f(x_M) = percentile^{-1}(percentile(x_M, D_M), D_S) \tag{2}$$

Suppose a guitarist wants a virtual drummer to adapt its density to the guitar's energy. As a proxy of energy, we use master feature F_M = RMS for the guitar. Drums' density is represented by slave feature F_S = hit-count (i.e., number of onsets in the chunk). Fig. 1 shows the distribution of the RMS values of a typical guitar recording, and of the hit-count values of a typical drum recording. The distributions strongly differ, by their total number of values (257 values for the RMS, 157 for the hit-count), ranges (RMS values range from 0.03 to 0.16 whereas hit-count values range from 0 to 19) and number of bins (for the RMS, there are 50 different bins, and only 19 for the hit-count). The f2f mapping selects the hit-count value x_S with the same percentile as the RMS value x_M. A value $x_M = 0.076$, corresponds to a percentile of 0.067. The hit-count value with the same percentile is 2, as shown in Fig. 1.

2.2.2 *The feature-to-chunk (f2c) mapping*

Eventually, the connection selects which chunk to play, given x_S. This is implemented by the feature-to-chunk (f2c) mapping, which represents the *musical intention* behind the connection. We specify this mapping *operationally*, by composing various *filters*. Given x_S and a subset $C \subseteq D_S$ of chunks, each filter selects a subset of C that satisfies a specific rule. Filters are defined once for all, as illustrated in the following examples:

- *ClosestChunks*$(C, x_S) \overset{def}{=} \underset{c \in C}{\text{argmin}} \mid F_S(c) - x_S|$
- *FarthestChunks*$(C, x_S) \overset{def}{=} \underset{c \in C}{\text{argmax}} \mid F_S(c) - x_S|$
- *MatchingChunks*$(C, x_S) \overset{def}{=} \{c \in C \mid F_S(c) = x_S\}$
- *UnmatchingChunks*$(C, x_S) \overset{def}{=} \{c \in C \mid F_S(c) \neq x_S\}$
- *ChordMatchingChunks*$(C, x_S) \overset{def}{=} \{c \in C \mid chord(c)$ is substitutable to $x_S\}$
- At a given beat b, *AdaptiveClosestChunks*(C, x_S)
$$\overset{def}{=} \begin{cases} \textit{ClosestChunks}(C, x_S) \textbf{ if } b \text{ is the first beat of a bar} \\ \{c\} \text{ where } c \text{ is the chunk that follows in } D_S \text{ the chunk currently playing } \textbf{otherwise} \end{cases}$$
- *RandomChunk*$(C) \overset{def}{=} random(C)$

The f2c mapping of a connection is a composition of filters, such as:

$$\textit{RandomChunk}(f_n(\ldots f_2(f_1(D_S, x_S), x_S) \ldots, x_S))$$

where $f_1, \ldots, f_n$ are filters.

In our example of the guitar controlling the drums, we use a *ClosestChunks* filter. A value of $x_M = 0.076$ for the guitar RMS will trigger a drum chunk with a hit-count value closest to 2.

When the composition of filters yields an empty subset, specific procedures are applied, such as using a default style database for the instrument.

2.2.3 *Multiple masters for one slave*

To better approximate the complexity of interaction occurring in a real band, a slave agent may be connected to several master agents. In this case, the f2f mappings are first computed independently and then intersected.

2.3 Representing Harmony as a Connection

In tonal music, such as jazz, musicians improvise on a chord sequence that is known to all beforehand. VB implements such shared harmonic information by a specific agent that represents the chord sequence.

The chord sequence agent A_C provides a unique feature whose value x_C is the name of the next chord in the sequence. A_C is typically connected to harmony-dependent agents, such as pitched instruments (e.g., piano, guitar). For such a virtual agent A, the connection is defined as follows: the master agent is A_C, the master feature value is x_C, and the slave agent is A.

Depending on the expected behavior of the virtual agent, with respect to harmony, various slave features and mappings may be used.

The most typical example is that of a piano comping agent A_P that is expected to play chords in the same harmony as x_C. Each audio chunk of the database D_P is associated to a feature value x_P which represents the corresponding chord name. In this case, the slave feature value is x_P, the f2f mapping is the identity: $x_P = x_C$, and the filter is *ChordMatchingChunks*. Note that the matching is not necessarily strict, but can use substitution rules, as explained in the next section.

2.4 Reflexive Style Databases

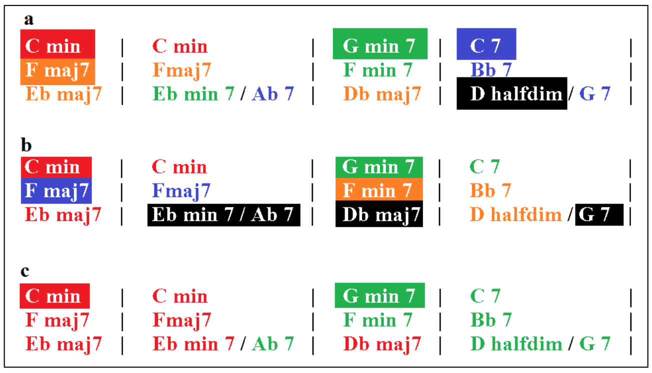
VB also features a *memoryless* mode, i.e., which doesn't require pre-recorded style databases. One motivation is to avoid "canned music" effects caused by the audience being unaware of the content of the style databases. In this mode, databases are built on-the-fly, typically like interactive systems such as Continuator [12] or Omax [11]. Reflexive style databases are used to generate various species of *self-accompaniments* such as duos or trios with oneself [13].

Conceptually, reflexive style databases do not differ from pre-recorded ones. Technically, they raise various real-time issues (segmentation and feature extraction must be performed in real-time without interfering with the main VB loop) that are not discussed in this paper.

More interestingly, reflexive style databases raise *sparsity* issues. Because a database contains only what the musician has played so far, its size will typically be much smaller than that of pre-recorded databases. As a consequence, the system has far fewer possibilities for generation. This is particularly annoying in contexts using predetermined chord grids. In principle, the system requires a long *feeding phase*, to accumulate at least one chunk for every chord of the sequence before it can start playing back relevant audio material. Such a feeding phase is usually boring for the musician as well as for the audience.

In order to reduce feeding, we propose to automatically expand style databases by using audio transformations, such as *transpositions*, implemented with pitch shifting algorithms [14]. In VB, the audio chunks of a database can be pitch-shifted so that the transposed bars can be used in new harmonic situations.

We also use so-called *substitution rules* to further expand the style database. Substitution rules consist in re-



a

C min	C min	G min 7	C 7
F maj7	Fmaj7	F min 7	Bb 7
Eb maj7	Eb min 7 / Ab 7	Db maj7	D halfdim / G 7

b

C min	C min	G min 7	C 7
F maj7	Fmaj7	F min 7	Bb 7
Eb maj7	Eb min 7 / Ab 7	Db maj7	D halfdim / G 7

c

C min	C min	G min 7	C 7
F maj7	Fmaj7	F min 7	Bb 7
Eb maj7	Eb min 7 / Ab 7	Db maj7	D halfdim / G 7

Figure 2. The feeding phase reduction using (a) transpositions, (b) substitutions, and (c) a combination of both.

placing a chord by another one with an equivalent harmonic function. This operation is widely used in jazz to bring variety in performance [1]. VB uses chord substitutions to play in a certain harmonic context audio that was recorded in another harmonic context, provided the two contexts may be substituted.

By combining transpositions and substitutions, the feeding phase is drastically reduced. The three examples of Fig. 2 show how to harmonize the song *Solar* from a limited number of input chords, using transpositions (a), substitutions (b), and a combination of both (c). The input chords (highlighted) generate the remaining chords of Solar (marked in the same color). Without substitutions or transpositions, 12 input chords would be required in the feeding phase. In contrast, (a) requires 5 input chords, (b) requires 8, and (c) only 2. For instance in (c), *G min 7* can be substituted for *C 7* using substitution "G min 7 : C 7", but also *F min 7* using a 1 tone downward transposition, and *Bb 7* with a combination of these two operations.

3. APPLICATION TO JAZZ IMPROVISATION

The following examples illustrate various configurations of VB with one human guitarist and one or two reflexive virtual agents. In the first and second examples, we present duos in which the virtual agent is controlled respectively by a spectral centroid feature and by a rhythm pattern feature extracted from the human guitarist. Then, we extend the example to a guitar trio configuration. An accompanying web site illustrates all these configurations with videos of the corresponding performances [1].

3.1 Spectral Centroid-Based Duo

Two jazz guitarists improvising together commonly use a question-answer interaction scheme: one of the guitarists plays a melody; as soon as he finishes, the other guitarist plays another melody that borrows elements from the first one. Typically the answering melody is in the same pitch range or shares similar rhythm patterns with the original melody. While a guitarist proposes a melody, the other one either stops playing or accompanies the first one with, e.g., chord comping.

[1] http://francoispachet.fr/virtualband/virtualband.html

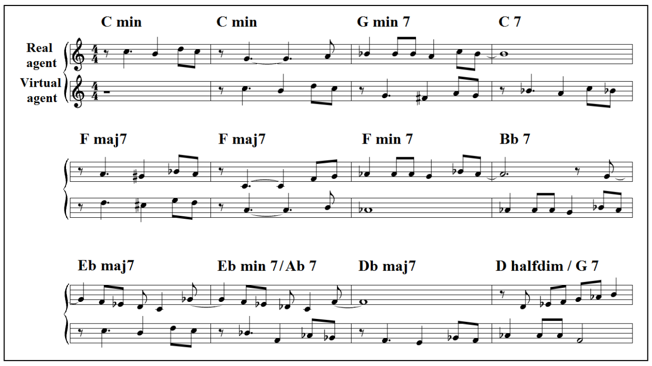


Figure 3. Score of a spectral-centroid based duo: a human agent (A_H) plays the melody of Solar. A virtual agent A_V matches the spectral centroid of A_H, with a one-bar delay. At bar 11 (*Db maj7*) A_V uses a transposition of bar 5 (*F maj7*) of A_H to match the spectral centroid of bar 10 of A_H. At bar 8 (*Bb 7*), substitution "F min 7 : Bb 7" allows A_V to play bar 7 of A_H (*F min 7*).

We simulate these scenarios with various configurations of VB based on pitch features of audio content. A reflexive style database records the incoming audio of a human guitarist A_H and extracts the spectral centroid of each bar. We chose the spectral centroid as an approximation of pitch, but more refined descriptors could be used interchangeably. A virtual agent A_V, associated to the database, is connected to A_H. At each bar, this connection is specified by the following elements:

- master feature F_H = spectral centroid of A_H; slave feature F_V = spectral centroid of A_V;
- f2f = percentile from a value F_H to a value of F_V (details in Section 2.2.1);
- f2c = *ClosestChunks* (details in Section 2.2.2).

Note that in the meantime, another connection ensures that A_V also plays according to the harmonic constraints (see Section 2.3 for details).

In this configuration, the system behaves as a self harmonizer: A_V follows the spectral centroid of A_H with a one-bar delay (see Fig. 3), using music material that sounds like what the guitarist just played.

Replacing the filter in the configuration above by:

- f2c = *FarthestChunks*

implements another musical intention: the agent plays audio that is far away pitch-wise to the human's input. In this configuration, the two outputs (human and virtual guitar) are clearly distinct.

3.2 Rhythm-Based Duo

Question-answer musical dialogues can also be based on rhythmical similarities. A guitarist plays a rhythm pattern for a few bars, and the other one responds by playing a similar pattern. We introduce a feature that represents the rhythm pattern: the *RMS profile*. This profile is obtained by computing the RMS 12 times per beat over one bar. The agent plays back chunks with a similar profile, with a systematic one-bar delay. Technically, the connection is specified by:

- master feature F_H = RMS profile of A_H; slave feature F_V = RMS profile of A_V;
- f2f = identity, i.e., for a value x_H of F_H and a value x_V of F_V: $x_V = x_H$;
- f2c = *ClosestChunks*. The distance between two RMS profiles is the scalar product of the two vectors.

This mode is fun and lively as the interaction is easily perceived by the musician and the audience. However, it requires larger databases, so it should be used when the system has accumulated enough rhythm samples to play back interesting variations.

As before, changing the filter to:

- f2c = *FarthestChunks*.

implements a very different musical interaction: the virtual agent plays chunks that are dissimilar to the input of the musician, which is more difficult to anticipate for a human than similar patterns.

3.3 Trio

In a typical jazz guitar trio, one of the guitarists improvises melodies on a harmonic grid, while the other two provide respectively chordal and bass accompaniments. These roles (melody, chords, and bass) represent different guitar *playing modes*. During an improvisation, guitarists typically shift roles in turn. When a musician takes the lead (solo), the other guitarists adapt their behavior so that each mode is always played by someone. With VB, we represent this configuration so that a guitarist can be self-accompanied by two reflexive virtual agents, sharing the same reflexive database.

In addition to the RMS profile, we extract the *playing mode* from each recorded beat, among four possible modes: melody, chord, bass and silence [2]. Each virtual agent is associated to a unique playing mode (one to the bass, one to the chords), and agents follow a mutually exclusive rule, i.e., they play only if the human guitarist is not playing in the same mode, following [13]. Given a virtual agent A_V (virtual bass or virtual chords), such a rule is easily modeled by a connection:

- master feature F_H = playing mode of A_H; slave feature F_V = playing mode of A_V;
- f2f = identity;
- f2c = *UnmatchingChunks*.

Furthermore, like in the previous example, each agent follows the rhythmical patterns of the guitarist. For instance, a walking bass, or chord comping (a lot of notes per bar, regularly spaced) can be triggered by playing a fast and regular melody. This configuration provides the feeling of a standard jazz guitar trio to a solo musician.

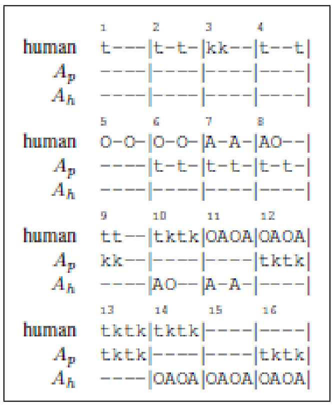
4. OTHER APPLICATIONS

In this section, we describe applications of VB in two other musical contexts: beatboxing and mash-ups; also illustrated by videos on our web site.

4.1 Reflexive Beatboxing

Beatboxing is a music style where musicians use their mouth to simulate percussion. Beatboxing also involves humming, speech or singing (see [17]). Common beatboxers typically alternate between modes, but some great beatboxers are able to play two modes at the same time (e.g., percussion and humming).

Beatboxing with VB aims at augmenting the performance of a moderately good beatboxer by allowing him or her to play, via reflexive virtual agents, several modes at the same time. Using the same connection settings as in Section 3.3, the system records and stores in a reflexive database the incoming audio of the real beatboxer. From this audio, the database distinguishes between two playing modes: the percussion mode and the humming mode. Then two virtual agents are connected to this database, one representing the percussion mode and the other the humming mode. The classification is performed following a similar scheme to [2], except for the set of features selected by the classifier (here harmonic-to-noise ratio, spectral centroid and Yin). Following a mutually exclusive principle, virtual agents play alternatively, depending on the mode of the human beatboxer, and following his or her rhythmic patterns. Table 4 illustrates a typical session with such an augmented beatboxer.



```
         1    2    3    4
human    t---|t-t-|kk--|t--t|
  A_p    ----|----|----|----|
  A_h    ----|----|----|----|
         5    6    7    8
human    O-O-|O-O-|A-A-|AO--|
  A_p    ----|t-t-|t-t-|t-t-|
  A_h    ----|----|----|----|
         9    10   11   12
human    tt--|tktk|OAOA|OAOA|
  A_p    kk--|----|----|tktk|
  A_h    ----|AO--|A-A-|----|
         13   14   15   16
human    tktk|tktk|----|----|
  A_p    tktk|----|----|tktk|
  A_h    ----|OAOA|OAOA|OAOA|
```

Figure 4. A 16-bar performance of the beatboxing system: `t` and `k` are percussive sounds, `O` and `A` are hummed vowels. A_P is the percussive agents, and A_H is the humming agent. Agents follow a mutually exclusive principle and try to match the human's rhythm with a one-bar delay.

4.2 Mash-up

A mash-up is created by blending two or more songs, usually by overlaying a track of one song over the tracks of another [5]. Mash-ups exploit multi-track songs whose tracks are available as separated audio files. A straightforward way to implement mash-up with VB is to represent each track of each song as a virtual agent. The database of each agent consists of all the audio chunks obtained by segmenting the corresponding track. The mash-up is obtained by muting and replacing one agent by a track agent representing the same instrument of another song. The virtual agent representing the replacing track is connected to the muted agent of the original song.

For instance, one can replace the drums in song *Roxanne* (*The Police*) by another drummer. The muted agent that represents the original drum track controls the substitute drummer through rhythm patterns. Technically, the settings of the connection are the same as presented in the first example of Section 3.2. On the accompanying web site we provide mash-ups of *Roxanne* with the drummer of 1) *Smells Like Teen Spirit* (*Nirvana*), 2) *Hey* (*Pixies*), 3) Bossa Nova and 4) Funk drums played by Jeff Boudreaux.

We can hear in the provided example each drummer playing in his style while seemingly following the song's structure by, e.g., playing breaks at the right time, or playing more intensively on choruses and bridges.

5. CONCLUSION

We revisit the problem of interacting with stylistically consistent agents from a MIR viewpoint. In VirtualBand, interactions are specified using *features pairs*, taken from the vast library of features developed in MIR. VB agents are reactive but not deliberative, i.e., they do not attempt to exhibit autonomy, make goals, or plan ahead. But the examples show that even with *only* reactive agents, rich and complex interactions can take place, by exploiting the complex correlations that typically occur between pairs of features computed on human audio signals.

However, VB only scratches the surface of the new field of *individual style modeling*. Current work focuses on issues like how to "saturate" a style database, or how to predict the emergence of long-term structure from low-level feature interaction.

6. ACKNOWLEDGEMENT

This research is conducted within the Flow Machines project which received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 291156.

7. REFERENCES

[1] J. Coker. *Elements of the Jazz Language for the Developing Improvisor*. Alfred Music Publishing, 1997.

[2] R. Foulon, F. Pachet, and P. Roy. Automatic classification of guitar playing modes. *Proc. of the CMMR Symposium*, 2013.

[3] T. Gifford and A.R. Brown. Do androids dream of electric chimera? In *Proc. of the ACMC*, pages 56–63, 2009.

[4] T. Gifford and A.R. Brown. Beyond reflexivity: Mediating between imitative and intelligent action in an interactive music system. In *Proc. of the HCI Conference*, 2011.

[5] J. Grobelny. Mashups, sampling, and authorship: A mashupsampliography. *Music Reference Services Quarterly*, 11(3-4):229–239, 2008.

[6] M. Hamanaka, M. Goto, H. Asoh, and N. Otsu. A learning-based jam session system that imitates a player's personality model. In *Proc. of the IJCAI*, volume 18, pages 51–58, 2003.

[7] A. Hawryshkewich, P. Pasquier, and A. Eigenfeldt. Beatback: A real-time interactive percussion system for rhythmic practise and exploration. In *Proc. of the NIME Conference*, pages 100–105, 2011.

[8] L. Iocchi, D. Nardi, and M. Salerno. Reactivity and deliberation: A survey on multi-robot systems. In M. Hannebauer, J. Wendler, and E. Pagello, editors, *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, volume 2103 of *Lecture Notes in Computer Science*, pages 9–34. Springer, 2000.

[9] M. Marchini and H. Purwins. Unsupervised generation of percussion sound sequences from a sound example. In *Proc. of the SMC Conference*, 2010.

[10] A. Martin, A. McEwan, C.T. Jin, and W. L. Martens. A similarity algorithm for interactive style imitation. In *Proc. of the ICMC*, pages 571–574, 2011.

[11] J. Nika and M. Chemillier. Improtek: integrating harmonic controls into improvisation in the filiation of OMax. In *Proc. of the ICMC*, pages 180–187, 2012.

[12] F. Pachet. The continuator: Musical interaction with style. *Journal of New Music Research*, 32(3):333–341, 2003.

[13] F. Pachet, P. Roy, J. Moreira, and M. d'Inverno. Reflexive loopers for solo musical improvisation. In *Proc. of the SIGCHI Conference*, CHI '13, pages 2205–2208. ACM, 2013. Best paper honorable mention award.

[14] C. Schörkhuber and A. Klapuri. Pitch shifting of audio signals using the constant-q transform. In *Proc. of the DAFx Conference*, 2012.

[15] D. Schwarz. Current research in concatenative sound synthesis. In *Proc. of the ICMC*, pages 9–12, 2005.

[16] G. Sioros, A. Holzapfel, and C. Guedes. On measuring syncopation to drive an interactive music system. In *Proc. of the ISMIR Conference*, pages 283–288, 2012.

[17] D. Stowell and M. D. Plumbley. Characteristics of the beatboxing vocal style. *Dept. of Electronic Engineering, Queen Mary, University of London, Technical Report, Centre for Digital Music C4DMTR-08-01*, 2008.