Interacting with a Musical Learning System: The Continuator

ABSTRACT
The Continuator system is an attempt to bridge the gap between two classes of traditionally incompatible musical systems: 1) interactive musical systems, limited in their ability to generate stylistically consistent material, and 2) music composition systems, which are fundamentally not interactive. The purpose of Continuator is to extend the technical ability of musicians with stylistically consistent, automatically learnt musical material. This requires the ability for the system to build operational representations of musical styles in real time, and to adapt quickly to external musical information. The Continuator is based on a Markov model of musical styles augmented to account for efficient real time learning of musical styles and to arbitrary external bias. The paper describes the main technical issues at stake concerning the integration of an agnostic learning scheme in an interactive instrument, and reports on real-world experiments performed with various musicians.

KEYWORDS: Musical interaction, music, improvisation, collaborative music.

INTRODUCTION
"I don't play piano, I play pianist" used to say French Pop singer Claude Nougaro, pointing to his long time accompanist Maurice Vander. This joke summarizes well the goal the Continuator project. We seek to design musical instruments that provide exciting interactions through the ability to learn automatically arbitrary musical styles, and adapt these styles to the playing modes of the musician. The undertaking can be seen as a way to turn musical instruments from passive objects to active, autonomous systems, with which one can interact using high-level controls, much in the same way Claude Nougaro, through the blink of an eye, can control, or influence his pianist Maurice Vander.

Musical performance has been the object of numerous studies using all the software technologies at hand. In our context, we can divide these approaches in two categories: *interactive systems* and intelligent *music composition* systems. Interactive music systems propose ways of transforming in real time musical input into musical output. Musical interactive systems have been popular both in the experimental field [15] [17] as well as in commercial applications, from one-touch chords of arranger systems to the recent and popular Korg Karma synthesizer [9]. While a lot of work has been devoted to efficient controllers and interfaces for musical systems [4], [10], these systems all share a common drawback: they do not manage time, there is no memory of the past, and consequently the music generated is strongly correlated with musical input, but not or poorly with a consistent and realistic musical style.

On the other hand, music composition systems precisely aim at representing stylistic information, to generate music in various styles, from the Illiac suite [8] to the automatic compositions of [6]. More recently, constraint techniques have been used to produce stylistically consistent 4-part Baroque music (see [12] for a survey). In the domain of popular music, prototypes such as [2], [3], [14] have demonstrated the technical feasibility of simulating convincingly jazz styles by computer. In opposition to interactive music systems, the main drawback of these approaches is that they do not allow real musical interaction: they propose fully-fledged automata that may produce realistic music, but cannot be used as actual instruments. Moreover, these approaches require explicit, symbolic information to be fed to the system, such as human input for supervised learning, underlying harmonic structure, tempo, song structure, etc.

The system we present here is an attempt to combine both worlds: design real-time interactive musical instruments that are able to produce stylistically consistent music. More precisely, we propose a system in which musical styles are learned automatically, in an agnostic manner, and therefore do not require any symbolic information (style, harmonic grid, tempo). The system is seamlessly integrated in the playing mode of the musician, as opposed to traditional fully automatic or question/answer systems, and adapts quickly and without human intervention to unexpected changes in rhythm, harmony or style. Finally, the design of the system allows the sharing of stylistic patterns in real time and constitutes in this sense a novel form of collaborative musical instrument.

The remaining of the paper is structured as follows. First we describe the heart of the system, based on a Markov model of musical styles, augmented with a hierarchy of learning functions to adapt to imprecision in musical inputs. Then we discuss the issues related to turning the learning facility into an actual musical instrument. Finally we report on experiments with the system in various real world musical contexts.

INSIDE THE CONTINUATOR

In the standard mode, the system receives musical Midi input from one musician. The output of the system is itself sent to a Midi synthesizer and then to a sound reproduction system. The system acts basically as a sequence continuator: the note stream of the musician is continuously segmented into musical phrases. Each phrase is sent asynchronously to a phrase analyzer, which builds up a model of recurring patterns. In reaction to the played phrase, the system immediately generates a continuation, according to the database of patterns already learnt.

A HIERARCHICAL MARKOV MODEL OF MUSICAL SEQUENCES

Researchers in AI and information theory have long addressed the technical issue of learning automatically and in an agnostic manner a musical style. Shannon introduced in his 1948 seminal paper the concept of information based on probability of occurrence of messages. This notion was quickly used to model musical styles, and these experiments showed that it was possible to create pieces of music that would *sound like* given styles, by simply computing and exploiting probabilities of note transitions. More precisely, given a corpus of music material (typically music scores, or MIDI files), the basic idea is to represent in some way the local patterns found in the learnt corpus, by computing transition probabilities between successive notes. New music sequences are then generated using these probabilities, and these sequences will contain, by definition, the patterns identified in the learnt corpus.

One of the most spectacular applications of Markov chains to music is probably [6], although his musical productions are not entirely produced automatically. A good survey of state-of-the-art of Markov based techniques for music can be found in [16], and a recent development in [1].

These works show clearly two things: 1) Markov chain models and their extensions are powerful enough to represent efficiently musical patterns, but 2) their generative power is limited due to the absence of long-term information. In another words, these models can fool the listener on a short scale, but not for complete pieces.

Using Markov models for interaction purposes, and not for composing complete, fully-fledged musical pieces, allow us to benefit from 1) while avoiding the drawback of 2). The responsibility for organizing the piece, deciding its high-level structure is left to the musician. The system only "fills in the gaps", and the power of Markov chain can be exploited fully to this aim.

The Continuator system is yet another species in the world of musical Markov systems, although with novel features. In our context, we want to learn and imitate musical styles in a faithful and efficient manner, and make the resulting mechanism useable as an *actual music instrument*. This raised a number of technical issues, whose solutions were integrated in the Continuator. These issues are addressed in the following section.

HIERARCHICAL MARKOV MODELS

The learning module we propose systematically learns all phrases played by the musician, and builds progressively a database of patterns detected in the input sequences. We designed an indexing scheme which represents all the subsequences found in the corpus, in such a way that the computation of continuations is complete and as efficient as possible.

This technique consists in building a prefix tree by a simple, linear analysis of each input sequence. Each time a sequence is input to the system, it is stored in memory, and all subsequences encountered are systematically added to the tree. For reasons of space, we describe here only the most generic functions.

For instance, let us suppose the following input sequences:

Sequence #1: {A B C D}

and later:

Sequence 2#: {A B B C}

The system will build a tree containing, for all possible subsequences of each of these two sequences, the list of all continuations encountered in this learning corpus, and weighted by their number of occurrences. The scheme is called variable-order Markov chain because it contains the continuations for all subsequences of any length (up to a given maximum, typically 10). In our example, a subsequence such as {B} has the following possible continuations: C (from sequence #1), and B (from sequence #2).

A subsequence such as {A B} has continuations: C (from sequence #1) and B (from sequence #2).

A subsequence such as {B B C} has only as possible continuation D from sequence #1. Note that in this last case, there is no continuation for the whole subsequence {B B C}, so we get the continuation for the longest possible subsequence, here, {B C}. When several continuations are similar, they are all repeated. For instance, the continuations of {A} are {B, B} (from sequences #1 and #2 respectively).

In our context, the most important characteristic of the data structure we propose is that the sequence learned is not the input sequence itself. Indeed, Midi sequences have many parameters, all of which are not necessarily interesting to learn.

For instance, a note has attributes such as pitch, velocity, duration, start time, and possibly other information provided by continuous controllers (pitch bend, modulation,

etc.). A chord has attributes such as the pitch list, possibly its root key, etc. The sequence learned is therefore not the input sequence itself, but a sequence obtained by applying a *reduction function* to the original input sequence. The simplest function is the pitch. A more refined function is the combination of pitch and duration. [5] and [16] proposed different reduction functions (called viewpoints) for representing music in the context of musical analysis. Our experiments with real time interactive music led us to develop and implement such a library of reduction functions, including the ones mentioned in these works, as well as functions specially designed to take into account more realistic Jazz and classical styles. One of them is the *PitchRegion*, which is a simplification of pitch: instead of considering explicit pitches, we reduce pitches in regions, in practice by considering only *pitch / region_size*.

GENERATION

The generation phase consists then, given an initial input sequence played by the musician, in computing successively continuations, step by step using a tiling process. First a note item is computed for the input sequence. Then the input sequence augmented by this item is considered, and the next item is computed, etc. At each step, a continuation for the subsequence of the maximum length is found, which results in optimum consistency with regards to the learnt corpus.

An important improvement on classical Markov-based generation mechanisms is behavior of our algorithm when it encounters a phrase for which no continuation is found.

Suppose a model trained to learn the arpeggio in Figure 1:

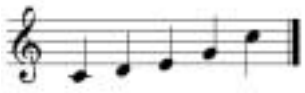

Figure 1. An arpeggio learnt by the Continuator.

Suppose that the reduction function is as precise as possible, say pitch, velocity and duration. Suppose now that the input sequence to continue is the following (Figure 2):

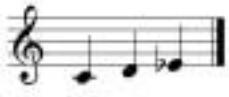

Figure 2. An input sequence which does not match exactly with a subsequence in learnt corpus.

It is clear that any Markov model will consider that there is no continuation for this sequence, because there is no continuation for the last note of the input sequence (here, E flat). The models proposed so far would then draw a new note at random, and actually start a new sequence.

However, it is also clear intuitively, that a better solution, in such a case, is to shift the viewpoint. In our context, this corresponds to using a less refined reduction function. Let us consider for instance pitch regions of three notes instead of pitches.

The learnt sequence of Figure 2 is then reduced to:

{PR1 PR1 PR2 PR3 PR5}

The input sequence is itself reduced to

{PR1 PR1 PR2}

In this new model, there is a continuation for the input sequence {PR1 PR1 PR2}, which is PR3.

Because our model keeps track of the original input sequences (and not only their reductions), we can generate the note corresponding to PR3 in the learnt corpus, in our case G. Once the continuation has been found, the process is started again with the new sequence, using the more refined reduction function.

More precisely, we introduce a *hierarchy* of reduction functions, to be used in cases of failure. This hierarchy can be defined by the user. A typical hierarchy is:

1 – pitch * duration * velocity

2 – small pitch region * velocity

3 – small pitch regions

4 – large pitch regions

where the numbering indicates the order in which the functions are to be considered in cases of failure in the matching process.

The approach we propose allows to take into account inexact inputs, with a minimum cost. The complexity for retrieving the continuations for a given input sequence is indeed very small as it involves only walking through trees, without any search.

FROM AN AUTOMATON TO A MUSICAL INSTRUMENT

The learning module described in the preceding section is able to learn and generate music sequences that sound like the sequences in the learnt corpus. As such, it provides a powerful musical automaton able to imitate faithfully styles, but not an interactive musical instrument. This section describes the main design concepts that can be used to turn this style generator into an interactive musical instrument. This is achieved through two related constructs: 1) a step-by step generation of the music sequences achieved through a real time implementation of the generator, and most importantly 2) an extension of the

Markovian generation process with a *fitness function* which takes into account characteristics of the input phrase.

REAL TIME GENERATION

The real time generation is a strictly technical issue but is an important aspect of the system since it is precisely what allows to take into account external information quickly, and ensure that the music generated follows accurately the input, and remains controllable by the user.

How Fast is Fast?

To give an estimation of our real time constraints, we have to know how fast a musician can play. We have considered an example by John McLaughlin, considered as one of the fastest guitarist in the world, in an example performed for a demo of a pitch to Midi converter (http://www.musicindustries.com/axon/archives/john.htm). An analysis of the fastest parts of the sample yields a mean duration of 66 milliseconds per note. Of course, this figure is not definitive, but can be taken as an estimate for a reasonable maximum speed. Our system should have a response time short enough so that it is impossible to perceive a break in the note streams, from the end of the player's phrase, to the beginning of the system's continuation: A good estimation of the maximum delay between two fast notes is about 50 milliseconds.

Thread Architecture

The real time aspect of the system is handled as follows. Incoming notes are detected by the system using the interruption polling process of MidiShare [11]: each time a note event is detected, it is added to a list of current note events. Of course, it is impossible to trigger the continuation process only when a note event is received. To detect phrase endings, we introduce a phrase detection thread which periodically wakes up and computes the time elapsed between the current time and the time of the last note played. This time delta is then compared with a *phraseThreshold*, which represents the maximum time delay within notes of a given phrase. If the time delta is less than the *phraseThreshold*, the process sleeps for *SleepTime* milliseconds. If not, a new phrase is detected and the continuation system is triggered, which will compute and schedule a continuation. The phrase detection process is represented in Figure 3.

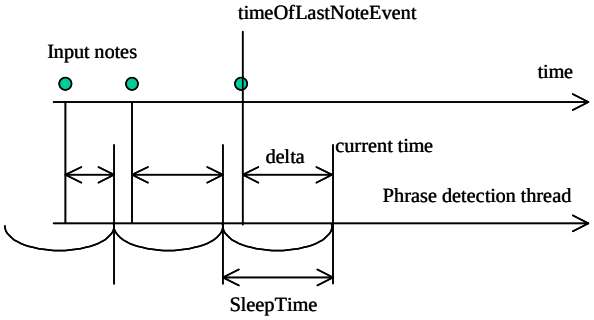

Figure 3. The input phrase detection process.

In other words, each time the phrase detection thread wakes up at time *t*, it computes the current time delay *delta*:

```
delta := currentTime – timeOfLastNoteEvent
```

It then compares this delay with the phrase threshold, decides or not to detect a phrase ending, and schedules itself to wake up at *t +SleepTime*:

```
If (delta >= phraseThreshold) then detectPhrase();

Sleep (SleepTime)
```

The real time constraint we have to implement is therefore that the continuation sequence produced and played by the system should be played with a maximum of 50 milliseconds after the last note event. The delay between the occurrence of the last note of a phrase and the detection of the end of the phrase is bounded by *SleepTime*.

In practice, we use a value of 20 milliseconds for *SleepTime*, and a *phraseThreshold* of 20 milliseconds. The amount of time spent to compute a continuation and schedule it is on average 20 milliseconds, so the total amount of time spend to play a continuation is in the worse case of 40 milliseconds, with an average value of 30 milliseconds, which fits in the scope of our real time constraint.

Step-by-Step Generation Process

The second important aspect of the real time architecture is that the generation of musical sequences is performed step-by step, in such a way that any external information can be used to influence the generation (see next section). The generation is performed by a specific thread (generation thread), which generates the sequence by chunks. The size of the chunks is parameterized, but can be as small as 1 note event. Once the chunk is generated, the thread sleeps and wakes up for handling the next chunk in time.

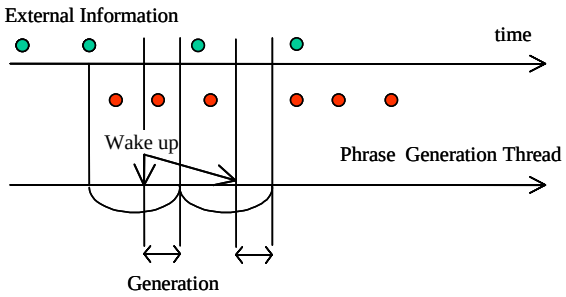

Figure 4. The step-by-step Generation Process allows to take into account external information continuously.

BIASING MARKOV GENERATION

The main idea to turn our automaton into an interactive system is to influence the Markovian generation by characteristics of the input. As we saw above, the very idea of Markov-based generation is to produce sequences in such a way that the probabilities of each item of the sequence are the probabilities of occurrences of the items in the learnt corpus.

In the context of musical interaction, this property is not always the right one, because many things can happen during the generation process. For instance, in tonal music, the harmony can change: in a Jazz trio for instance, the pianist will play chords which are not always the same, throughout the generation process. Because we target a real world performance context, these chords are not predictable, and cannot be learnt by the system prior to the performance. The system should be able somehow to take this external information into account during the generation, and twist its generated sequence in the corresponding direction.

The idea is to introduce a constraint facility in the generation phase. External information may be sent as additional input to the system. This information can be typically the last 8 notes (pitches) played by the pianist for instance, if we want the system to follow harmony. It can also be the velocity information of the whole band, if we want the system to follow the amplitude, or any information that can be used to influence the generation process. This external input is used to influence the generation process as follows: when a set of possible continuation nodes is computed (see section on generation), instead of choosing a node according to its Markovian probability, we weight the nodes according to how they match the external input. For instance, we can decide to prefer nodes whose pitch is in the set of external pitches, to favor branches of the tree having common notes with the piano accompaniment.

In this case, the harmonic information is provided implicitly, in real time, by one of the musician (possibly the user himself), without having to explicitly enter the harmonic grid or any symbolic information in the system.

More precisely, we consider a function *Fitness(x, Context*) with value in [0, 1] which represents how well item *x* fits with the current context. For instance, a Fitness function can represent how harmonically close is the continuation with respect to external information. If we suppose that *piano* contains the last 8 notes played by the pianists for (and input to the system), *Fitness* can be defined as:

$$Fitness(p, piano) = \frac{\text{nb notes common top and piano}}{\text{nb notes in piano}}$$

This fitness scheme is of course independent of the Markovian probability defined above. We therefore introduce a new weighting scheme which allows to parameterize the importance of the external input, via a parameter *S* (between 0 and 1):

$$Prob(x) = S * Markov\_Prob(x) + (1 - S) * Fitness(x, Context)$$

By setting *S* to extreme values we eventually get two extreme behaviors:

- S = 1, we get a musical automaton insensitive to the musical context,
- S = 0, we get a reactive system which generates the closest musical elements to the external input it finds in the database.

Of course, interesting values are intermediary: when the system generates musical material which is both stylistically consistent, and sensitive to the input. Experiments in these various modes are described below in the Experiment Section.

CONTROL AND HIGH-LEVEL STRUCTURE

Playing "interesting" phrases is an important ingredient of musical improvisation, but it is not the only one. High-level structure is as important to produce a full-fledged piece: it is not always desirable to have the system continue systematically all phrases played. Typically, the musician can start a piece by playing, e.g. a musical theme, and then let the system play progressively longer and longer continuations until the end, when the musician plays back the theme, without the system continuation.

To allow the user to switch between these different modes in an intimate and non-intrusive way, we have identified a set of parameters that are easy to trigger in real time, without the help of a graphical interface. The most important parameter is the S parameter defined above, which controls the "attachment" of the system to the external input. The other parameters allow the musician to switch on or off basic functions such as the learning process or the continuation process.

By default, the systems stops playing when the user does, to avoid superposition of improvisations. With minimum training, this mode can be used to produce a unified stream of notes, thereby producing an impression of seamlessness between the sequence actually played by the musician and the one generated by the system.

Additionally a set of parameters can be adjusted from the screen, such as the number of notes to be generated by the system (as a multiplicative factor of the number of notes in the input sequence), and the tempo of the generated sequence (as a multiplicative factor of the tempo of the incoming sequence). One important control parameter allows the musician to force the system to remain in some regions, deemed interesting or particularly well suited to

the moment of the piece. This parameter is usually associated with a Midi control change such as the breath control. By pushing the control up, the system will remain in the region, and instantaneously become a sort of arpeggiator, creating a periodic rhythm from the local Markovian region selected. Another trigger of the same control restores the system back to the usual Markovian generation process, which results in forcing the system to explore another musical region.

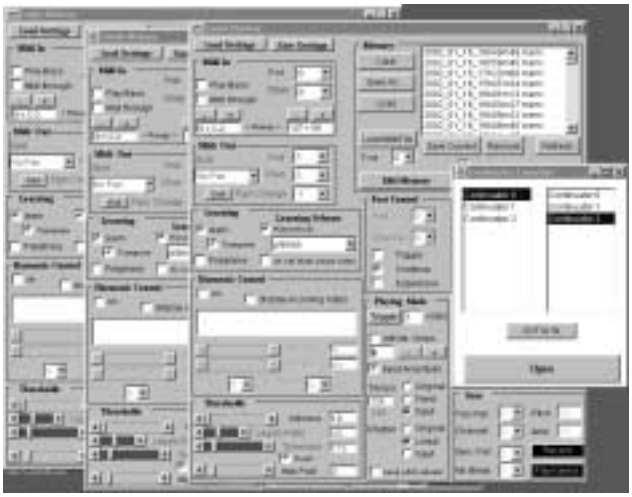

Figure 5. Multiple copies of the Continuator in action

The system described above contains many parameters, but and is in some sense autonomous. There are musical situations in which it is interesting to use several, independent versions of the system, each with its own inputs and outputs. We have designed a scheme which able to launch different continuators at the same time, possibly synchronizing them (see Figure 5).

EXPERIMENTATIONS

We have conducted many experiments with the system, in various modes and configurations to validate our claims. We report results and lessons learned in the following sections.

INDISTINGUISHABILITY

It is difficult to describe music by words, and rate its quality, especially jazz improvisation. However, we can easily rate how the system differs from the human input. We have conducted tests to check whether listeners could tell when the system is playing or not. In most of the cases, if not all, the music produced is undistinguishable from the user's input. This is typically true for quick and fast Jazz solos. An audio example available at our web site gives an example of a Jazz tune ("On Green Dolphin Street", by Kaper & Washington), where the musician (here, the author) plays a guitar solo which is continued by the system, interrupts several time the system to launch another phrase, and finally concludes the improvisation. The reader/listener can assess the difficulty in distinguishing these different phases as the whole improvisation is seamless. Other examples can be found at our web site, in which the system generates long and often impressive jazzy phrases in the styles of guitarists such as Pat Martino, John McLaughlin, or Alan Holdsworth.

ATTACHMENT

The attachment mechanism we have introduced is particularly spectacular when used in conjunction with a fixed metrical structure. In this mode, the system can play an accompaniment in a given tempo which tries to satisfy two conflicting constraints: 1) stylistic consistency and 2) consistency with the external input. Audio Examples will be presented in which the system plays a chord sequence (from previously learnt material), and tries in real time to "follow" harmonically the input by a real musician (the author again). The chords generated by the system fit naturally and quickly to harmonic changes. Occasional unexpected harmonic progressions are also generated, but which all fit the two constraints of stylistic consistency and fitness with external input.

Many experiments in the various styles of the Karma music workstation were also recorded and will be made available at our web site. In these experiments, we have connected the Continuator to the Korg Karma workstation, both in input and output. The Continuator is used as an additional layer to the Karma effect engine. The Continuator is able to generate infinite variations from simple recordings of music, in virtually all the styles proposed by the Karma (over 700).

SUBJECTIVE IMPRESSIONS: THE AHA EFFECT

Besides the evaluation of the musical quality of the music produced by the system, we have noticed a strong subjective impression on the musician playing. We have conducted a series of experiments and concerts with famous Jazz musicians, and the reactions of musicians playing with the system were always extremely positive. The most striking effect, noticed systematically on all musicians experimenting with the system, can be best described as a *Aha* effect, triggered by the sudden realization that the system is starting to play exactly in the same style as oneself, or suddenly pops up patterns played a few minutes earlier.

The accompanying Video shows a series of such effects on different musicians, styles and instruments (Bernard Lubat, Alan Silva, Claude Barthélémy), with sudden and characteristic bursts of laughter or astonishment. Some of tem are illustrated in Figures 5 and 6.

Figure 5. Jazz musician Alan Silva playing with Continuator.

Figure 6. Bernard Lubat playing with the Continuator.

NEW MUSICAL COLLABORATIVE MODES

An interesting consequence of the design of the system is that it leads to several new playing modes with other musicians. Traditionally, improvised music has consisted in quite limited types of interaction, mostly based around question/answer systems [2] [3]. With the Continuator, new musical modes can be envisaged:

- *Single autarcy*. One musician plays with the system after having fed the system with a database of improvisations by a famous musician, as Midi files. We have experimented in particular with a database of midi choruses from Pat Martino, provided by [7], and a database of Bernard Lubat's piano style. An extension of this mode consists in using several versions of the same system, with the same inputs, but generating simultaneously different outputs. Used in the linear rhythmic mode, this configuration results in a multiple voice arpeggiator which that produces continuously variations.

- *Multiple autarcy*: each musician has its own version of the system, with its own database. This provides a traditional setting in which each musician plays with his/her own style. Additionally, we experimented concerts in which one musician (Gÿorgy Kurtag) had several copies of the system linked to different midi keyboards. The result for the listener is a dramatic increase in musical density.

- *Master/Slave*: one musician uses the system in its basic form, another (e.g. pianist) provides the external data to influence the generation. This is typically useful for extending a player's solo ability while following the harmonic context provided by another musician.

- *Cumulative*: all musicians share the same pattern database. This setting was experimented during a Jazz festival (Uzeste, France), where two musicians played with the same (Bernard Lubat) database,

- *Sharing*: each musician plays with the pattern database of the other (e.g.; piano with guitar, etc.). This creates exciting -new possibilities as a musician can experience playing with unusual patterns.

CONCLUSION

We have described a music generation system which is able to produce music learnt in a agnostic manner, while remaining intimately controllable. This is made possible by introducing several improvements to the basic Markovian generation, and by implementing the generation as a real time, step-by-step process. The resulting system is able to produce musical continuations of any user – including beginners - according to previously learnt, arbitrary styles.

The experiments and concerts performed with professional artists show that not only the music generated is of very high quality (as good as the music learnt by the system), but, more importantly, that such a learning facility can be turned into an actual music instrument, easily and seamlessly integrated in the playing mode of the musician. Current works focus on the design of an audio version to expand the possibility of musical input (voice in particular). This version will use the same kernel described here, augmented with audio descriptors extracted in real time. These descriptors are made possible by ongoing work on musical metadata (in particular in the Cuidado project, see Pachet, 2002). The resulting system, besides extending the possibility to audio, will also provide a link between the domain of musical performance and musical listening.

REFERENCES

1. Assayag, G. Dubnov, S. Delerue, O. Guessing the Composer's Mind: Applying Universal Prediction to Musical Style, Proc. ICMC 99, Beijing, China, I.C.M.A., San-Francisco, 1999.

2. Baggi, D. L. NeurSwing: An Intelligent Workbench for the Investigation of Swing in Jazz, in Readings in Computer Generated Music, IEEE Computer Society Press, 1992.

3. Biles, John A. Interactive GenJam: Integrating Real-Time Performance with a Genetic Algorithm, Proc. ICMC 98, Ann Arbor, Michigan, 1998.

4. Jan Borchers, Designing Interactive Musical Systems: a Pattern Approach, HCI International '99. 8th International Conference on Human-Computer

Interaction, Munich, Germany, from 22-27 August, 1999.

5. Conklin, D. and Witten, Ian H. Multiple Viewpoint Systems for Music Prediction, JNMR, 24:1, 51-73, 1995.

6. Cope, David. Experiments in Musical Intelligence. Madison, WI: A-R Editions, 1996.

7. Heuser, Jorg, Pat Martino – His contributions and influence to the history of modern Jazz guitar. Ph.D thesis, University of Mainz (Ge), 1994.

8. Hiller, L. and Isaacson, A., *Experimental Music*, New York: McGraw-Hill, 1959.

9. Karma music workstation, Basic guide. Korg Inc. Available at: http://www.korg.com/downloads/pdf/KARMA_BG.pdf, 2001.

10. New Interfaces for Musical Expression (NIME'01), http://www.csl.sony.co.jp/person/poup/research/chi2000wshp/, 2000.

11. Orlarey, Y. Lequay, H. MidiShare: a Real Time multi-tasks software module for Midi applications Proceedings of the International Computer Music Conference, Computer Music Association, San Francisco, pp. 234-237, 1989.

12. Pachet, F. Roy, P. "Automatic Harmonization: a Survey", Constraints Journal, Kluwer, 6:1, 2001.

13. Pachet, F. “Content-Based Management for Electronic Music Distribution”, Communications of the ACM, to appear, 2002.

14. Ramalho G., Ganascia J.-G. Simulating Creativity in Jazz Performance. Proceedings of the National Conference in Artificial Intelligence, pp. 108-113, AAAI-94, Seattle, AAAI Press, 1994.

15. Robert Rowe, *Interactive Music Systems* (1993).

16. J. L. Triviño-Rodriguez; R. Morales-Bueno, Using Multiattribute Prediction Suffix Graphs to Predict and Generate Music, Computer Music Journal 25 (3) pp. 62-79, 2001.

17. William F. Walker A Computer Participant in Musical Improvisation, Proc. Of CHI 1997. Atlanta, ACM Press, 1997.