OMax-Ofon

Gérard Assayag, Georges Bloch, Marc Chemillier

OMAX-OFON

G. Assayag
Ircam-Cnrs UMR Stms
gerard.assayag@ircam.fr

G. Bloch
Université de Strasbourg
gbloch@umb.u-strasbg.fr

M. Chemillier
Université de Caen
chemilli@free.fr

ABSTRACT

We describe an architecture for an improvisation oriented musician-machine interaction system. The virtual improvisation kernel is based on a statistical learning model. The working system involves a hybrid architecture using two popular composition/perfomance environments, Max and OpenMusic, that are put to work and communicate together. The Midi based OMAX system is described first, followed by the OFON system, its extension to audio.

1. INTRODUCTION

Machine improvisation and related style learning problems usually consider building representations of time-based media data, such as music or dance, either by explicit coding of rules or applying machine learning methods. Stylistic machines for dance emulation try to capture movement by training statistical models from a sequence of motion-capture sequences [6]. Stylistic learning of musical style use statistical models of melodies or polyphonies to recreate variants of musical examples [8]. These and additional researches indicate that emulation of particular behaviors is possible and that credible behavior could be produced by a computer for a specific domain.

In the field of music improvisation with computers there has been recently notable advances in statistical music modeling that allows capturing stylistic musical surface rules in a manner that allows musically meaningful interaction between humans and computers. We have experimented several of these models during the past years, and more recently implemented OMAX, an environment which benefits both from the power of OpenMusic [4] for modelling and high level programming, and MaxMSP [15] for performance and real time processing. This environment allows interaction with one or more human player, on-the-fly stylistic learning, virtual improvisation generation, metrical and harmonic alignment, stylistic model archiving and hybridation. It operates on two distinct time scales : the Max one, which is close to real time and involves fast decision/reaction, and the OpenMusic one, which has a deeper analysis/prediction span directed towards the past and the future. This two conceptions of time have to interact and synchronize through communication channels through which musical data as well as control signals circulate in both direction. A decisive advantage we have found in this hybrid environment experience is its double folded extendability. In the OM domain, it is easy, even while the system is running, to change the body of a lisp function and test incremental changes. Furthermore, it is easy to enrich the generation by connecting the system to a wide variety of compositional algorithms available in this environment. Same thing in the Max domain, with a fabulous collection of real-time generation and processing modules. As a proof of this flexibility, it took no more then 2 days, once OMAX was running, to extend it to OFON, its audio version, by a few tweaks on the OM side and the Max one as well. So we think of this environment more as an indefinitely modulable and extendible experimental environment for testing new ideas about interaction, than to a fixed, ready for distribution, application. A realist strategy would be, once we are happy with a learning/generating model experimented in OMAX, to fix it into a pure Max application by taking the time to recode the OM part in C.

2. STATISTICAL MODELLING

Statistical modeling of musical sequences has been experimented since the very beginnings of musical informatics [8]. The idea behind *context models* we use, is that events in a musical piece can be predicted from the sequence of preceding events. The operational property of such models is to provide the conditional probability distribution over an alphabet given a preceding sequence called a context. This distribution will be used for generating new sequences or for computing the probability of a given one. First experiments in context based modeling made intensive use of Markov chains, based on an idea that dates back to Shannon : complex sequences do not have obvious underlying source, however, they exhibit a property called *short memory property* by Ron and al; there exists a certain memory lengh L such that the conditional probability distribution on the next symbol does not change significantly if we condition it on *contextes* longer than L. In the case of Markov chains, L is the order. However, the size of Markov chains is $O(|\Sigma|^L)$, so only low order models have been actually experimented.

To cope with the model order problem, in earlier works [3, 10-13] we have proposed a method for building musical style analyzers and generators based on several algorithms for prediction of discrete sequences using Variable Markov Models (VMM). The class of these algorithms is large and we focused mainly on two variants of predictors - universal prediction based on Incremental Parsing (IP) and prediction based on Probabilistic Suffix Trees (PST).

From these early experiences we have drawn a series of prescriptions for an interactive music learning and generating method. In the following, we consider a learning algorithm, that builds the statistical model from musical samples, and a generation algorithm, that walks the model and generates a musical stream by predicting at each step the next musical unit from the already generated sequence. These prescription could be summed up as :

- learning must be incremental and fast in order to be compatible with real-time interaction, and be able to switch instantly to generation (real-time alternation of learning and generating can be seen as « machine improvisation » where the machine « reacts » to other musician playing).
- The generation of each musical unit must bounded in time for compatibility with a real time scheduler
- In order to cope with the variety of musical sources, it is interesting to be able to maintain several models and switch between them at generation time.
- Assuming the parametric complexity of music (multi-dimensionality and multi-scale structures) multi-attribute models must be searched for, or at least a mechanism must be provided for handling polyphony.

We have chosen for OMAX a model named *factor oracle* (FO) that conforms with points 1, 2 and 4. It is described in [1] and its application to music data is described in [2]. An exemple is given in figure 1.

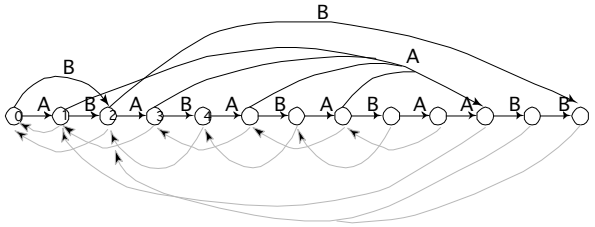

Figure 1. A factor oracle for the string ABABABABAABB.

FO are basically automata that capture all sub-phrases (factors) in a sequence of symbols, transformed into a linear chain of states by an efficient incremental algorithm. Through this transformation, the sequence structure is "learned" into the FO. The states are linked by two kind of arrows. Forward arrrows are called *factor links*. By following these at generation time, it is possible to generate factors of the original sequence, i.e. litterally repeat learned subsequences. Backward arrows are called suffix links. By following these, it is possible to switch to another subsequence sharing a common suffix with the current position. Such a recombination, is really a *context* based generation, the context being the common suffix. Although the probability model has not yet been defined, Fo's are conjectured to be VMM.

3. OMAX ARCHITECTURE

OMAX is distributed across two computer music environments : OpenMusic and Max. Obviously, the Max components are dedicated to real-time interaction, instrumental capture, Midi and audio control, while OpenMusic components are specialized in higher level operations such as building and browsing the statistical model. Communication channels between max and OM allow the transmission of streams of musical data as well as control information. In the primitive version of OMAX, Midishare [14] has been used as an inter-application communication system, but all the transmitted data had to be encoded in Midi. Of course this is not very convenient, specially when structured data has to be exchanged, so this component is being rewritten using OSC [17], a powerful message based communication protocol that is getting very popular in the computer music community.

On the OpenMusic side, a few lisp-based software components have been implemented :

- A *Midi listener*, that fetches the incoming midi events sent by Max and bufferizes them
- A *phrase segmenter* that prepares a phrase from the Midi stream based on special signals received from Max.
- A *polyphony expert module* that turns the raw Midi stream into a sequence of polyphonic units (see details below).
- A *harmonic expert module*
- A *learning/generating module* that implements the FO structure and knows 2 messages : *learn* and *improvize*, with a granularity of one phrase.
- A *communication module* that handles the data/control exchanges with Max.

On the Max side, the components are :

- A *Midi input module* that listens to a Midi port, prepares the data to be sent to OM, provides basic analysis in order to detect phrase boundaries. This module also provides clock information for the beat mode (see below).
- A *Midi sequencer A* where sequences can be launched in the case when the computer is supposed to play a fixed part such as a

such as a harmonic/rythmic background for improvisation.

- A *Midi sequencer B* that is fed with generated "improvisations" received from OM.
- A *communication module* that is responsible for data and control exchanges with OM.

Polyphony management :

The statistical modelling techniques we use suppose the data to be in the form of sequences of symbol taken from an alphabet. Of course music being multidimensional it has to be processed in some way in order for the model to be usable and meaningful. We detail in [11] a method for processing polyphonic streams in order to turn them into a sequence of symbols such that a sequence model can be built from which new generated strings can easily be turned into a polyphony similar in structure to the original. Such "super-symbols", output by the polyphony manager in OM, are "musical slices" associated with a certain pitch content and a duration.

Free / Beat scenarios :

Two different scenarios can be experimented in OMAX. In the so-called "free-mode", there is no notion of metrical structure. The human performer plays freely. From the OM point of view, phrases to be learned arrive with the appearance of sequences of polyphonic symbols with various durations. Generated phrases will share the same flexible allure.

In the "beat-mode", the musical session is anchored to a certain metrical / harmonic grid, which is generally played by the computer. In this mode, Max will intertwine clock messages in the stream of midi events it sends to OM. These clocks segment the time according to beat boundaries. The OM phrase segmenter and polyphony manager will build consecutive polyphonic symbols which duration are aligned with the beat boundaries. New improvisations generated by OM are now multiples of the beat duration. Upon receival of these sequences, Max will align them correctly with the metrical structure it is playing.

Handling the harmony is a bit more complicated and involves several round trips between Max and OM. This is due to the fact that we have mostly experimented in jazz harmony, including the jazz substitution mechanisms. For a given harmonic grid, stored in an OM data structure, the OM harmony expert module, operating one grid ahead with respect to the current time in Max, generates a new variant grid. The model used for this is detailed in [7]. OM expands the new grid into chords and sends the sequence of chords to Max which stores them in a midi sequencer, keeping them ready for the next grid occurence. At any time, when a human played phrase is sent to OM and therefore turned into a sequence of beat aligned polyphonic symbols, a harmonic label is sticked to each of these symbol. This is possible because the entering beats are numbered relatively to the periodic grid and OM knows the details of the grid currently played by Max.

The FO learning module now learns data packets that are contained within a beat and associated to a harmonic label.

At generation time, the harmony may have changed due to the perpetual substitution mechanism. So the FO is asked to generate under the constraint of the current harmony. Technically, this means that we follow the arrows (factor or suffix links) by giving a priority to the pathes that generate a sequence of harmonic labels either identical to the current ones or compatible under the substitution theory.

In the beat mode, the generated impros are thus aligned metrically and harmonically, although nothing restricts the harmony from changing at each grid occurrence. Of course it would be just a small change to this architecture if we wanted the harmony to be lively captured fro a human performer (e.g. a pianist), and there would be no need to refer to a particular style such as 'jazz' as long as we had a theory for harmonic label equivalences. Such an equivalence theory would be necessary in order to avoid too many failures at generation time. Such failures occur when the statistical process leads up to a state where no outgoing arrow is compatible with the current harmonic label.

Figure 2 gives an overview of the (simplified) Omax architecture.

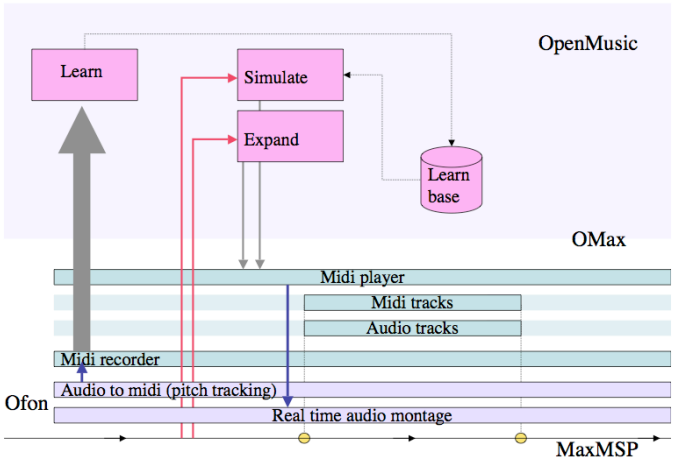

Figure 2. OMAX architecture

The bottom line is a time line where the small circles show grid boundaries. In the Max world (bottom half) the rectangles indicate Max processes running along time. The picture shows a situation where the beat mode, the more complex one, is used. The leftmost, thick arrow shows the stream of Midi information sent continuously to OM by the Midi recorder and being learned. The next, broken arrows are punctual signals sent to OM, telling it either to "expand" (generate data, either from compositional algorithm or from the harmony model), or to "simulate", that is asking an improvised phrase from the statistical learn base. These musical data, sent back to Max, will be reserved in the Midi sequencer and be ready for the next grid occurrence. Midi Tracks and audio tracks are for static data to be played by Max.
The two bottom Max processes implement the OFON extension and will be explained in the next section.

4. OMAX WITH SOUND : OFON EXTENSIONS

The Ofon system was developed in order to benefit from the Omax system while using acoustic instruments. It only works now with monophonic instruments.
The principle is simple : a pitch tracker extracts the pitch and intensity values while the sound of the instrument is recorded in a buffer.
Time stamps corresponding to the offset of sound events in the buffer are associated to the corresponding Midi events built at the ouput of the pitch tracker and sent to OM as well. Now the symbols learned in the FO contain a reference into a sound buffer. When OM performs the reconstruction of an improvisation using the oracle, the time stamps are sent back to Max along with the midi stream. An Ofon module, the audio montage, is then able to assemble the sound pieces extracted from the sound buffer thanks to the time stamps. So, the music is learned from an audio signal, and the virtual improvisations result in an audio signal reconstructed from thet original audio material.
It is worth noting that, since we can get several computer-generated improvisations going on at the same time, it becomes possible to create multiple clones of the live player and have him/her play with them. This results, of course, in an exciting interaction situation. For the moment, the system has been extensively used with saxophone players (Philippe Leclerc in France and Tracy McMullen in San Diego), and experimented with student actors of the Théâtre national de Strasbourg in a language and sound poetry environment under the direction of the stage director Jean-François Peyret.
There are three main problems in order to achieve such a system: a reliable pitch tracking, a real time editing of the sounds in the buffer, and a diffusion of the cloned instrument interesting enough to bear the immediate comparison with the real instrument.

Editing the buffer

Editing the buffer is a relatively trivial task, given the fact all the tools for sound editing are already implemented in Max.
The main problem of editing (whether in real time or not) is to correctly estimate the position of the editing point. This ends up being a pitch tracking problem: the tracking must determine the exact moment when the note starts (or with a constant offset), for every possible pitch.
An aspect specific to Omax is that, sometimes, editing is simply not necessary, when Omax recombines sound events that, in the original take, have been played in sequence. Therefore these occurrences must be detected to avoid unnecessary editing and just read continuously the corresponding part of the buffer in sequence.

Pitch tracking

One of the main problems is to get a reliable pitch tracking system. Everyone who has tempered with such a system knows how picky these kinds of system are, as soon as we are dealing with instruments with large timbre capacities, as it is often the case in real life. Having experimented with saxophone and voice, we started with quite tricky examples.
However, we need to know what we mean by "reliable". An important point, as we have seen, is to determine when the note begins whatever the pitch and regardless of the playing mode. If the time offset is constant on each pitch but differs according to the range, it can be

arranged by referring to a table of offsets, once the pitch is determined.

Extracting the pitch itself can be problematic, especially when the player uses sounds that do not correspond to the official labels of the Conservatoire de Paris or of the Julliard School. This is often the case when one works, for instance, with jazz players. One runs therefore into two kinds of problems: pitch detection errors and "wrong notes" added. Curiously, the errors in pitch detection are not that important, as long as they are consistent. The point of the oracle is to find similar passages, and consistency is enough to have it work well. Also, errors generally correspond to some aural logic (octave errors, which do not matter that much, and particularly strong harmonics being taken as the fundamental tone).

Much more important is the problem of added notes, since the Omax system may receive a lot of irrelevant information. These notes most often are more or less interesting pitch information on transient sounds (especially on instruments like saxophone, voice, but also bow and reed instruments in which the attack part can be relatively underlined for expressive purposes). It is important to get rid of those.

We are using the yin~ pitch algorithm [9] as implemented in Max by Norbert Schnell, along with some post processing of our own. The output of a saxophone looks like the plot in figure 3.

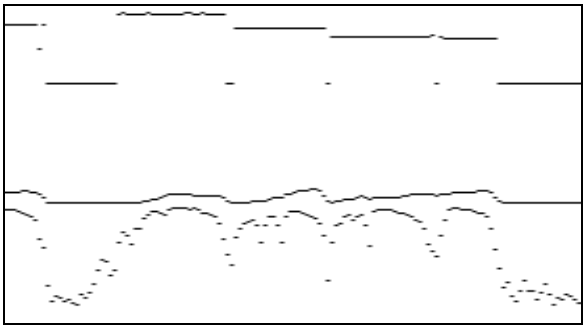

figure 3. An example of the yin~ output. Pitch is on top, amplitude in the middle and quality at the bottom. Time is right to left.

The most interesting feature of the yin~ algorithm is its quality parameter. In the picture above, we can notice that, if the quality is not sufficient, the pitch is routed to a default low value (this is the case, for instance, on the right part of the graph, in the beginning of the excerpt, since the graph scrolls from right to left). This only means that the instrument is not playing. The quality factor is useful in an instrument in which the "purity" of the sound is extremely variable: we can accept pitches even with a low quality factor. In the middle of the graph, there are moments when the quality falters: how to decide if the pitch is good enough? On the other hand, when reading the graph, the solution is obvious: the melody is (from right to left) D# E G# D# B (last note on the graph, cut). Obviously, the three (or perhaps four) first notes are slurred, the last one is detached, and there is vibrato on the top D#.

The extensions to yin tried so far consists of :

- using a window of a certain length (e.g., 60 milliseconds) for each new pitch value detected
- using the quality factor in order to allow several types of sound, to generate note-off and to consider low quality as a special pitch
- achieving statistics on each window: that is, each new pitch value opens the window and, for the duration of the window, the following values are analyzed; if the original value represents more than a certain proportion of all found values (50% for example), a note-on is generated.

It results in a delay (the window length) before evaluating the pitch; this delay is relatively short, except if a large window is chosen to suppress the vibrato (not always a desirable feature). If two notes follow each other (slurred playing), the corresponding note-off is automatically generated (since we are dealing with a monophonic instrument). If notes are detached, the insufficient quality factor (insufficient for the duration of the window) will generate a note-off between them.

The result is satisfying, especially since it gives a really correct time estimation for editing. It requires of course a careful tune-up of the three parameters: quality factor, window length and statistical mean to keep a pitch value (the two last parameters depend on the DSP vector size, of course). With a very large vector size, there can be just one value in each window, and the averaging is done by the yin~ algorithm itself, a hardly satisfying solution.

Towards a realistic diffusion of the cloned instrument

The other problem of Ofon (and actually of Omax in certain circumstances) is a very general and quite ancient problem: the coexistence of acoustic instruments with loudspeakers. The difficulty is even greater in Ofon, since the virtual instrument is supposed to sound so similar to the "real" one.

The main difference between a loudspeaker and an acoustic instrument is that the directivity of the acoustic instrument varies, according to many parameters, mostly pitch and loudness (not only, by the way: on a violin, many pitches can be played on several strings and have therefore distinct directivities).

That is the reason why a variable directivity system called "La Timée" has been developed at Ircam [16] in Olivier Warusfel's team. The connection of "La Timée" to Ofon has proved easy, thanks to the help by Olivier Warusfell, since the MIDI data used with Omax and resulting from the pitch tracking give the necessary

information of pitch and loudness allowing to command the directivity parameters (which are pitch-loudness dependent). These data are transmitted by Max to La Timée control system through OSC. It results in a very good rendering of the virtual saxophones. The description of this system is beyond the scope of this paper However, it is possible to get a decent virtual instrument without such a complex system.

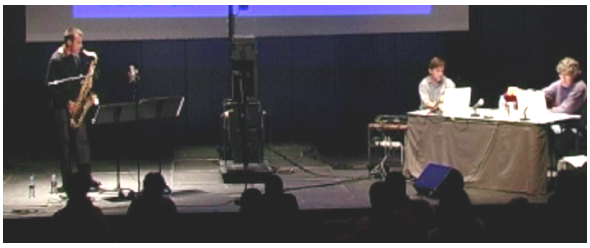

figure 4. An Ofon concert at Ircam (SMC'05, Workshop on Improvisation with the Computer) : left, Philippe Leclerc, saxophone. In the center, "la Timée. Right, Olivier Warusfel and Georges Boch.

A "cheap imitation" of a variable directivity system already gives a much more realistic effect than a single loudspeaker. It can be realized by a set of loudspeakers centered on one point. Each speaker points in one direction and can have its proper frequency response. The balance between the speakers is controlled by the pitch and loudness data, in reference (or not) to the directivity features of the emulated instrument.

5. OFONVIDEO

Ofonvideo was developed by three students of University of Strasbourg II (Anne-Sophie Joubert, Vincent Robischung and Émilie Rossez) under the direction of Georges Bloch. It adds a further Layer to Ofon, and therefore to Omax.

The principle is the same as in Ofon. The performer is filmed, the learning is performed on the music, and the filmed extracts are re-edited in real-time, according to the Omax virtual improvisation sequences augmented with time stamps. So, recombined improvisations of the music are doubled by a consistent recombination of video sequences.

Ofonvideo has been programmed with jitter, an extension of the Max programming language aiming primarily at video control.

The technical complexity of Ofonvideo depends on several features: the video has to be recorded on disk (and not in RAM), the machine must be powerful (actually, a single machine must be dedicated to the video processing) and the delay between the recording and playback of film and sound files must be carefully adjusted. Finally, the system is greatly enhanced if too short edits are avoided (in sound, one can edit a very short note, shorter even than a frame, but it is useless – and power expensive – in video).

Image and sound in Ofonvideo

In the principle, it looks like the sound in Ofonvideo will always double the image, a feature which can become tedious. This is false. It is always possible to call archived images, at any moment, and send them to the screen, regardless of the sound going on. And there is another possibility. In Ofon, there are five possible sound tracks at the same time, therefore five possible films. If the Ofon mixer mutes the sound tracks instead of stopping them, the images are still going on. Furthermore, Ofon can be set in "loop" mode while the film is running further on.

One of the main interests of Ofonvideo is to allow improbable encounter between musicians. For example, the saxophonist Philippe Leclerc can meet Thelonius Monk, improvise with him, and each musician influences the improvisation of the other. The filming ends up in a bottomless mirror, in which Leclerc watches Leclerc who whatches Monk who watches Leclerc… see figure 5.

figure 5. A picture of an Ofonvideo session, with Philippe Leclerc, saxophone, watching himself watching Monk. This very picture may appear, seconds later, on the screen!

6. FURTHER DEVELOPMENTS

Omax-Ofon-OfonVideo open new perspectives in computer assisted improvisation and in the study of style in improvised music, in the path of the interactive systems imagined by Joel Chadabe at the end of the sixties and developed later by many performers-researchers such as Georges Lewis, David Wessel etc. The representational and compositional capacities of OpenMusic enrich this system further and evokes the

idea of "composed improvisation" systems [5]. The coexistence of Open Music and Max gives another dimension to this idea: the presence of a sophisticated computer assisted composition software allows the interaction processes to act on more global features, features that only musical representation can grab in an abstract manner.

Let us claim this paradox: most interactive processes do require very little real time. If a decision is taken at a compositional level and on musical parameters that extend on a large scope (in other terms, if we view interaction as something else than a ornamentation of an otherwise fixed compositional process), we generally have some time to process the computation. What does matter is that, at a given moment, some proposition (solution) is available. For this reason, the coexistence of a very fast response and tracking (max) with a very powerful and flexible music representation program (OM) seems the obvious solution, until the ideal time when unified environments would exist.

This is not exactly the way Omax works at the present moment: some aspects (notably the acquisition of sequences) are directly made in Open Music. Not only is it not practical (the MIDI or sound or video sequences should be acquired in Max and their parameters globally sent to Open Music) but also detrimental to the modularity of the system. This is a historical consequence of the development of the Omax system, accomplished by several people over a large span of time. This defects of youth are being corrected, notably by grouping the whole sequencing aspect in the FTM environment developed by Norbert Schnell at Ircam. Once implemented, a complete interactive architecture to which any Open Music composition-improvisation process could be patched will be available.

7. REFERENCES

[1] Allauzen C., Crochemore M., Raffinot M., Factor oracle: a new structure for pattern matching, *Proceedings of SOFSEM'99, Theory and Practice of Informatics, J. Pavelka, G. Tel and M. Bartosek ed., Milovy, Lecture Notes in Computer Science 1725, pp 291-306, Springer-Verlag, Berlin,* 1999.

[2] Assayag, G., Dubnov, S., Using Factor Oracles for Machine Improvisation, G. *Assayag, V. Cafagna, M. Chemillier (eds.), Formal Systems and Music special issue, Soft Computing 8, pp. 1432-7643, September 2004.*

[3] Assayag, G., Dubnov, S., Delerue, O., "Guessing the Composer's Mind: Applying Universal Prediction to Musical Style," *Proc. Int'l Computer Music Conf., Int'l Computer Music Assoc., pp. 496-499, 1999.*

[4] Assayag A., Rueda C., Laurson, M., Agon C., Delerue, O. "Computer Assisted Composition at Ircam: PatchWork and OpenMusic," *The Computer Music J., vol. 23, no. 3, 1999, pp. 59-72.*

[5] Bloch, G., Chabot X., Dannenberg, R., «A Workstation in Live Performance: Composed Improvisation», *Proceedings of International Computer Music Conference*, The Hague, Netherlands, 1986.

[6] Brand, M., and Hertzmann, A, 2000, Style Machines, In Proceedings of SIGGRAPH 2000, New Orleans, Louisiana, USA

[7] Chemillier, M., "Toward a formal study of jazz chord sequences generated by Steedman's grammar," *G. Assayag, V. Cafagna, M. Chemillier (eds.), Formal Systems and Music special issue, Soft Computing 8, pp. 617–622, 2004*

[8] Conklin, D. « Music Generation from Statistical Models », *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences, Aberystwyth, Wales, 30– 35, 2003.*

[9] de Cheveigné, A., Kawahara, H. "YIN, a fundamental frequency estimator for speech and music", *J. Acoust. Soc. Am. 111, 1917-1930, 2002.*

[10] Dubnov, S. Assayag, G., "Improvisation Planning and Jam Session Design using concepts of Sequence Variation and Flow Experience", *Proceedings of Sound and Music Computing '05, Salerno, Italy, 2005.*

[11] Dubnov, S., Assayag, G., Lartillot, O., Bejerano, G., "Using Machine-Learning Methods for Musical Style Modeling," *IEEE Computer, Vol. 10, n° 38, p.73-80, October 2003.*

[12] Dubnov, S., Assayag, G. « Universal Prediction Applied to Stylistic Music Generation » *in Mathematics and Music, A Diderot Mathematical Forum, Assayag, G.; Feichtinger, H.G.; Rodrigues, J.F. (Eds.), pp.147-160, Springer-Verlag, Berlin, 2002.*

[13] Dubnov, S., Assayag, G., El-Yaniv, R. "Universal Classification Applied to Musical Sequences," *Proc. Int'l Computer Music Conf., Int'l Computer Music Assoc., 1998, pp. 332-340.*

[14] Orlarey, Y., Lequay " MidiShare : a Real Time multi-tasks software module for Midi applications", *Proceedings of the International Computer Music Conference 1989, Computer Music Association, San Francisco, 1989.*

[15] Puckette, M. "Combining Event and Signal Processing in the MAX Graphical Programming Environment." *Computer Music Journal 15(3): 68-77, 1991.*

[16] Warusfel, O., Misdariis N., "Sound Source Radiation Synthesis: From Stage Performance to Domestic Rendering" in *Proceedings of 116th AES Convention, 2004*.

[17] Wright M., Freed, A., Momeni A.: "OpenSound Control: State of the Art 2003.", *Proceedings of NIME 2003: 153-159, 2003.*