Planning Human-Computer Improvisation

Jérôme Nika, Jose-Manuel Echeveste, Marc Chemillier, Jean-Louis Giavitto

Planning Human-Computer Improvisation

Jérôme Nika[1,2], **José Echeveste**[1,2,3], **Marc Chemillier**[4], **Jean-Louis Giavitto**[1,3]
[1] IRCAM, UMR STMS 9912 CNRS, [2] Sorbonne Universités UPMC,
[3] INRIA Rocquencourt, MuTAnt team, [4] Cams, Ecole des Hautes Etudes en Sciences Sociales
jnika@ircam.fr, echeveste@ircam.fr, chemilli@ehess.fr, giavitto@ircam.fr

ABSTRACT

Improvisation intrinsically carries a dialectic between spontaneity/reactivity and long-term planning/organization. This paper transposes this dialectic to interactive human-computer improvisation where the computer has to interleave various generative processes. They require different levels of prior knowledge, and follow a coarser *improvisation plan* driven by real-time interactions with the musicians. We propose a time-aware extensible architecture allowing the temporal coordination of different *improvisation strategies*. It integrates various generative strategies capitalizing on the system *Improtek* into the high-level structure provided by the language associated to the score follower *Antescofo*. The resulting framework manages the musical events, the triggering of generative processes at different time scales and the declarative specification of improvisation plans driven by the occurrence of complex events.

1. INTRODUCTION

This article proposes a new framework to conciliate the need of spontaneity and reactivity with long-term planning and temporal organization in human-computer improvisation. This framework enables the temporal coordination of different *improvisation strategies* within an *improvisation plan* driven by the occurrence of complex musical and logical events.

Most improvisation styles rely on prior knowledge of the temporal evolution of the music to produce. This temporal organization can be an *explicit sequence* as standard themes in be-bop improvisations or given melodies in some traditional folk music. When it is not, the temporal structure can be specified as a *sequential scenario* describing a sequence of constraints that must be satisfied successively by the improvisation to be played. A standard example is the harmonic progression used in most current western musical styles such as rock, blues, jazz or pop music. Otherwise, the temporal organization may not take the form of a sequential structure. In this case it can be best described as responses to complex events implying both musical events and logical conditions, as for instance in soundpainting. These temporal structures exist concurrently at different time scale, for example: at short-term, the synchronization of the notes of a generated sequence with current tempo; generation of musical sequences satisfying global constraints in mid-term; and at a higher level, switching from an improvisation part to another, defined by different sets of prior knowledge, memory, mechanisms and rules (such as switching from lead to follow, from free to idiomatic, *etc*.). The coordination of these temporal sequences and reactions constitute complex improvisation plans or *dynamic scores*.

In [1], Rowe outlines that designing interactive musical systems pushes up musical composition "to a meta-level captured in the processes executed by the computer". The framework proposed in this paper addresses this "meta-level" and is aimed at the definition and the implementation of the temporal structures used to guide or constrain music generation in reaction to an active listening of the live musical input. It couples two literatures, generative model with real-time recognition and reactive systems, usually considered separately but whose integration makes complete sense in interactive and improvised musical practices.

This architecture has been deployed with the development of an experimental prototype capitalizing on the *Improtek* [2] and the *Antescofo* [3] systems. The former provides structured and guided musical generation from an ordered and indexed online or offline memory, while the latter provides musical synchronization and the possibility to specify reactions to unordered complex events.

After presenting some background in interactive improvisation systems and sequencers in section 2, this paper gradually describes how improvisation strategies with different degrees of indeterminism can be employed within the same plan. Section 3 focuses on fixed reactions to a planned input. Section 4 presents the generation of musical sequences satisfying long-term constraints and section 5 how reactivity can be injected into these generative processes through dynamic calls and parametrization with the musical context. Finally, section 6 sketches programing patterns for writing reactions to unordered complex events.

2. RELATED WORK

Interactive improvisation systems can be categorized by their hard-coded inherent strategy to drive the music generation process. A first category let to an operator-musician the guidance of the generation process: OMax [4, 5] is controlled by a user steering the navigation through a representation extracted in real-time from the playing of a live musician. The user is also at the heart of Mimi4x [6] and is involved in the construction of the performance by choos-

ing the musical corpus and modifying the generation parameters. Other systems are driven by an analysis of the live inputs: SoMax [7] translates the musical stream coming from an improviser into constraints, for example described in terms of harmonic background, to filter the possible paths in a corpus whose internal logic can also be followed. VirtualBand [8], emphasizes interaction and reactivity and extracts multimodal observations from the musician's playing to retrieve the most appropriate musical segment in a memory in accordance to previously learned associations. In the same line, Reflexive Looper [9] uses in addition harmonic annotations in the search criteria. Finally, an upstream structure or description can run the generation process: PyOracle [10] proposes to create behavior rules or scripts for controlling the generation parameters of an improvisation session. The concept of control improvisation in [11] introduces a control structure to guide the improvisation by a reference sequence and satisfying given specifications. The generation process presented in section 4 also includes an inherent strategy based on long-term constraints. In the context of improvised mixed music, it needs to be integrated in an environment allowing the specification of temporal developments of musical processes in reaction to human performance or controls given to an operator-musician.

The system *Antescofo* is chosen for the high-level organization of the musician-computer interaction because it combines score following capacity with an expressive timed and reactive scripting language. In Max/MSP or PureData [12] which are dataflow graphical languages and where control and signal processing are statically determined, it is easy to construct static behaviors, but much harder to organize and control changing behaviors according to a complex scenario. *Antescofo*, compiled as a Max or PureData object, is used in these cases to interact with the external environment. Other dynamic languages encounter some success in the interactive music community such as SuperCollider [13] or Chuck [14]. These are textual languages facilitating the programming of audio and algorithmic processes. Their real-time properties make them ideal tools for "Live Coding" practices, often improvised, where the time of composition (in the program) coincides with that of performance. However, the semantics of these languages does not allow the direct specification of the behavior of the external environment. Furthermore, their models of time is not directly linked with that of the performer.

Compared to traditional sequencers such as LogicPro, ProTools or CuBase, *Antescofo* is dedicated to more dynamic situations. Ableton Live with Max4Live adds more possibilities of interaction compared to the sequencers cited above, but without providing a flexibility allowing to synchronize the electronic processes to the elastic time of the musician.

3. FIXED REACTION TO A PLANNED INPUT

An improvised music performance may refer to predefined melodies, scores, audio materials or more broadly sequences of actions with their own temporality. The synchronization with a musician's performance of heterogeneous electronic actions (playing an audio file, triggering of a synthesis sound, or the execution of some analysis processes, *etc*) is a common problem of interactive music systems. Many solutions have emerged to deal with this issue depending on musical purpose or available technologies, leading to the score following approach used in the environment described in this paper.

The most elementary solution is to launch a predefined electronic sequence recorded on a fixed support (magnetic band, classical sequencer). In this case, the musician's performance is totally constrained by the time of the concerned support. Another way to make the time of electronic actions progress is to use a cue-list logic. The electronic is here defined as a list of successive actions and a mechanism as a pedal controller activated by the musician at the right moment triggers corresponding actions to execute. This approach is more flexible than the previous one because the time of the performance is under the supervision of the musician. However it raises the issue of how to partition the electronic part of the score if it is continuous in the thought of the composer and during the performance. Furthermore, giving the control of the electronic time to the musician can hinder the expressivity during the performance.

Score following is defined as the real-time alignment of an audio stream played by one or more musicians into a symbolic musical score. It offers the possibility to automatically synchronize an accompaniment, and thus can be used for the association of an electronic part to a predefined instrumental in an improvised music context. *Antescofo* is a real-time system for interactive music authoring and performing. It focuses on high-level musical interaction between live musicians and a computer, where the temporal development of musical processes depends on active listening and complex synchronization strategies [3]. In [15] a novel architecture has been proposed that relies on the strong coupling of artificial machine listening and a domain-specific real-time programming language for compositional and performative purposes. The user creates an *augmented score* whose language integrates both programmed actions and musical events, allowing a unique and flexible temporal organization. The augmented score includes both the instrumental part to recognize and the electronic parts and the instructions for their real-time coordination during a performance.

The syntax for writing the instrumental part allows the description (pitches and durations) of events such as notes, chords, trills, glissandi and improvisation boxes. Actions are divided into *atomic actions*, performing an elementary computation, and *compound actions*. The atomic actions can be: messages sent to the external environment (for instance to drive a synthesis module), a variable assignment, or another specific internal command. The `group` construction describes several actions logically within a same block that share common properties of tempo, synchronization and errors handling strategies in order to create polyphonic phrases. Other constructions such as `loops` for iterated actions or `curve` for continuous specification

are also available.

During performance, the runtime system evaluates the augmented score and controls processes synchronously with the musical environment, thanks to data received from the machine listening. The reactive system dynamically considers the tempo fluctuations and the values of external variables for the interpretation of accompaniment actions. The possibility of *dating the events and the actions relatively to the tempo*, as in a classical score, is one of the possibility offered by *Antescofo*. Within the augmented score language, the user can thus decide to associate actions to certain events with delays, to group actions together, to define timing behaviors, to structure groups hierarchically and to allow groups act in parallel.

Delays and durations are arbitrary expressions and can be expressed in relative time (in beats) or in physical time (in seconds). *Antescofo* provides a predefined dynamic tempo variable through the system variable `$RT_TEMPO`. This variable is extracted from the audio stream by the listening machine, relying on a cognitive model of the behavior of a musician [16]. Programmers may introduce their own frames of reference by specifying a local tempo for a `group` using a dedicated attribute. All the temporal expressions used in the actions within this group are then computed depending on this frame of reference. As for other attributes, a *local tempo* is inherited if groups are nested. A local tempo is an arbitrary expression involving any expressions and variables. This expression is evaluated continuously in time for computing dynamically the associated delays and durations.

4. IMPROVISATION GUIDED BY A SEQUENTIAL *SCENARIO*

4.1 Memory and scenario

When the prior knowledge on the structure of the improvisation is not as explicit as classical score, a melody or a theme, it may consist in a sequence of formalized constraints for the generation of the improvisation to create. Examples of such formalized structures can be a chord progression in blues, rock, or jazz improvisation; the harmonic progression given by the bass in the baroque basso continuo; or the precise description of the evolution of the improvisation in terms of melody, tempo or register in the indian raga. This section describes an improvisation model extending that of *ImproteK* [2] [1] , relying on such a sequence of constraints existing before the performance.

This model follows on the work initiated in [17, 18] on the navigation through a cartography of a musician's live playing, partly capturing his musical logic. The application of these principles in a real-time improvisation system led to OMax [4, 5], and long-term constraints and a priori knowledge were brought in the generation process with *ImproteK* by means of an abstract symbolism conveying different musical notions depending on the applications, like meter as regards rhythm or chord notation as regards harmony [19], joining previous works on the use of chord charts in improvisation [20, 21].

The improvisation process is here modeled as the articulation between a *scenario* to follow and a structured and indexed *memory* in which musical fragments are retrieved, transformed and reordered to create new improvisations:

- the *scenario* is a symbolic sequence guiding the improvisation and defined over an appropriate alphabet for the musical context,
- the *memory* is a sequence of contents labeled with a symbolic sequence defined over this same alphabet.

In this framework, "improvising" means going through the memory to concatenate some contiguous or disconnected blocks satisfying the sequence of temporal constraints given by the scenario. These blocks are chained in a way comparable to an improviser who is able to develop an improvisation by using motifs he eared or played himself in different contexts, described here with different scenarios. The improvisation process searches for the continuity of the musical discourse as well as the ability to grow apart from the original material. It relies on the indexation of some similar patterns in the scenario and the memory, and the self-similarities in both sequences.

4.2 Overview of the model

The scenario and the sequence describing the musical memory are represented as words defined over a same alphabet. This alphabet describes the equivalence classes chosen to compare the musical contents of the online or offline memory. After choosing a temporal unit for the segmentation, the letter at index T of the scenario S of length s is noted $S[T]$ and corresponds to the required equivalence class for the time T of the improvisation. In the same way, the letter $M[P]$ gives the equivalent class labeling the musical fragment corresponding to the date P in the memory M of length m. In what follows, each musical content in the memory will be assimilated to the letter $M[P]$ indexing it, and by extension the whole memory will be assimilated to the word M. The scenario gives access to a prior knowledge of the temporal structure of the improvisation to play. It enables to take into account the required classes for future dates $T+1$, $T+2$,... to generate improvisation at time T. The *current scenario*, noted S_T, corresponds to the suffix of the original scenario beginning at the letter at index T: $S[T]...S[s-1]$. At each time T, the improvisation goes on from the last state read in the memory at time $T-1$, searching to match the sequence of constraints given by the current scenario.

The proposed improvisation model undertakes successive *navigation phases* [2] through the musical memory, which rely on searches of *prefixes of the current scenario* in the memory. The length of these prefixes is one of the control parameters of the process. Figure 1 illustrates two consecutive generation phases.

[1] Links to video examples of live performances and work sessions can be found at http://repmus.ircam.fr/nika

[2] These navigation phases are successive steps in the algorithmic process producing the improvisation, but they do not correspond in general to distinct musical "phrases".

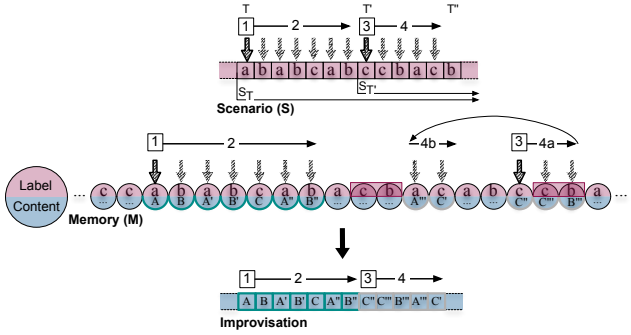

Figure 1. Improvising by articulating an indexed memory and a scenario: two navigation phases (1-2 and 3-4).

The first phase (steps 1-2) aims to generate the improvisation from the date T by satisfying the current scenario S_T=$S[T]...S[s-1]$, suffix of S. At the end of this first phase, the prefix $S[T]...S[T'-1]$ of the suffix S_T of S has been processed. A new research phase over the suffix $S_{T'}$=$S[T']...S[s-1]$ of S has to be launched at step 3 to complete the improvisation up to $T''-1$ (steps 3-4). Each of these phases through the memory is constituted by two consecutive *steps*:

1. Search for a prefix providing a *starting point* in the memory (steps 1 or 3 in figure 1), detailed in 4.3,
2. Follow a *linear or non-linear path matching the scenario* (steps 2 or 4 in figure 1), detailed in 4.4.

They respectively take advantage of the prior knowledge on the structure of the scenario and the memory, and address the concerns of musical continuity and transformation of the original material. Each of these two steps described in the following part of this section introduces a parameter whose value influences the musical result: c_f representing the *continuity regarding the future of the scenario*, and c_p quantifying the *continuity regarding the past of the memory*.

4.3 Step 1: searching for a section of the current scenario in the memory

The first step when going through the memory to produce the improvisation is to search for a pattern matching a section of the current scenario in the memory, that is to say a prefix $M[i]...M[i']$ of S_T=$S[T]...S[s-1]$ in M. The first element of this pattern, $M[i]$, gives a *starting point* for the navigation phase. The trivial solution is then to copy the whole pattern $M[i]...M[i']$ (the other options are described in the following paragraph 4.4). The set $SP(T)$ of possible starting points for the generation of the improvisation from a date T is defined by:

$$SP(T) = \{i \in \mathbb{N} \mid \exists c_f \geq 0, \\ M[i]...M[i+c_f] \in \text{Prefixes}(S_T)\}$$

where c_f measures the duration of the sequence in the memory starting at the chosen index i, matching a part of the current scenario S_T, and which can be literally cloned. The choice of an actual starting point among the elements in $SP(T)$ is made in accordance with the current state of the constraints imposed on it. For example, imposing the maximum value for c_f will lead to a maximal length prefix and will provide a homogeneous and continuous musical result throughout its duration. Conversely, a small value will lead to extractions of short segments in potentially different zones of the memory. Depending on the musical situations, one can successively prefer a state providing the coherence of the musical discourse in the long term and very close to a section of a learned sequence, or a more fragmented musical result. Constraints on others parameters also influence the choice of an element in $SP(T)$, among them the location of the pattern in the memory or the preference for a segment offering the best progression regarding the end of the segment produced by the previous phase. The control on these parameters can be given to an operator-musician and/or integrated in a higher level of the improvisation planning (see 6.5).

The prefix indexing algorithm used to obtain the set $SP(T)$ (detailed in an upcoming paper) uses the "failure function" of the Morris and Pratt algorithm [22] to process self-similarities in S_T in a preliminary analysis phase (as it is used in [23] to get the tables of prefixes and borders). This result is then used in the search phase to index the prefixes of the current scenario S_T in the memory M.

4.4 Step 2: following a non-linear path matching the current scenario

For each generation phase, the improvisation process starts from the chosen starting point and runs through the memory collecting a sequence of states whose labels match the scenario, and this mechanism goes on until the research of a new starting point is necessary. After a starting point is chosen in the memory at the indexing step (4.3), the first solution is to literally clone the prefix beginning by this state (step 2, figure 1). Yet, in particular when S_T and M are very close, the set of possible paths has to offer more than a simple copy to create new material. The navigation process exploits therefore an analysis of the self-similarities in the structure of the memory to generate a continuous musical discourse while widening the scope.

The possible *progressions matching the scenario* from a given state $M[k_0]$ is thus defined as the set of the states in the memory sharing a common past with $M[k_0]$, and satisfying the next label $S[T]$ imposed by the scenario. The set $P(T, k_0)$ of indexes k of the states in the memory matching the time T of S and being a candidate for the next progression from the previous state $M[k_0]$ is defined by:

$$P(T, k_0) = \{k \in \mathbb{N} \mid \exists\, c_p \in [1, k], \\ M[k-c_p]...M[k-1] \in \text{Suffixes}(M[0]...M[k_0]), \text{ and} \\ M[k] = S[T]\}$$

The similar pattern in M and S_T can be linearly followed by choosing in $P(T, k_0)$ the consecutive state in the memory $M[k] = M[k_0+1]$. Assuming that jumps between two segments in the memory sharing a common past preserve a certain musical homogeneity, non-linear paths are

also considered, as in the step 4 (4a then 4b) in figure 1, to avoid a simple copy of the learnt sequence. Such jumps are authorized when the factors ending at the origin and at the destination of these jumps share a common suffix. The length of this common suffix c_p measures the length of the shared musical past and therefore quantifies the "quality" of these jumps.

The automaton structure chosen to learn the musical memory is the Factor Oracle [24, 25]. The progression process of this second step of the generation extends the mechanism for improvisation, harmonization and arrangement proposed in [2] based on the navigation [26, 27] in this automaton for musical applications. This automaton presents links locating repeated patterns within the sequence and providing the existence of a common suffix between the elements that they connect, giving then the successive sets $P(T, k_0)$. The common suffix is seen as a common musical past in the context of this application. The postulate at the heart of the musical models using the Factor Oracle [4, 5, 7, 10, 6, 11] is indeed that such non-linear paths in a musical memory thus mapped enable to create musical phrases proposing new evolutions while preserving the continuity of the musical discourse.

5. INTRODUCING REACTIVITY INTO THE IMPROVISATION MODEL

5.1 From static to dynamic generation

The improvisation model guided by a scenario can be used in live performance to generate autonomous sequences satisfying given specifications or in an offline process, for instance composition. This section introduces scheduling of dynamic calls to this model in order to bring reactivity and adapt to the generation process to the improvisation environment. This way, the generation can react to changes of control parameters (given to an operator-musician and/or mapped to the live musical input) or to dynamically modified scenarios while being coherent with the past and keeping its long-term horizon.

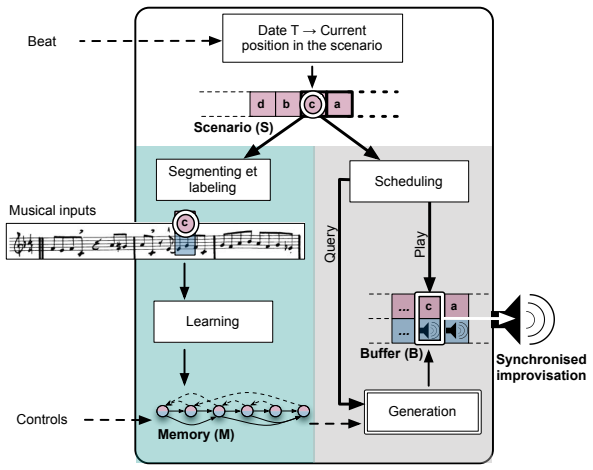

Figure 2. The generation process within a real-time architecture.

Listening to a source of beats gives the current date T, and also the associated label $S[T]$ in the scenario used in different concurrent processes. The three main aspects of the integration of the improvisation model into a real-time environment are presented in figure 2: 1) learning the musical material from the musicians playing with the system, 2) generating and 3) playing the new improvised phrases synchronously with the running improvisation session. For this end, a buffer is added to the two main elements in the model (scenario and memory). It contains the *short-term anticipated playing "intentions"*, which are refined over time.

5.2 Scheduling the navigation phases: anticipations and rewritings

The implementation of this architecture involves parallel processes listening and reacting to the environment, the elements produced by the model, and the instructions given by the operator or a higher scale improvisation plan.

Algorithm 1. Scheduling the reactions to the environment

Initial state:
Buffer (storing the musical elements to play) = $\emptyset$
E (index of the first empty position in the buffer) = 0
CurrentTimePlayed = false

1 **Whenever** T updated **do**
2 Learn inputs from [T-1,T[labeled by S[T-1] in M
3 CurrentTimePlayed ← false
4 **if** *Buffer*[T] **then**
5 Play(*Buffer*[T])
6 CurrentTimePlayed ← true

7 **Whenever** E - T < minimum imposed anticipation **do**
8 T' ← max(T, E)
9 Generate(T', $S_{T'}$)

10 **Whenever** modif. of parameters or S affecting date T' ≥ T **do**
11 Generate(T', $S_{T'}$)

12 **Whenever** *RecvdElem* = *(Idx,Content)* received **do**
13 **if** (*Idx* = T) & (¬ CurrentTimePlayed) **then**
14 Delay ← Date(update T) - Date(*RecvdElem*)
15 Play(*Content*, Delay)
16 CurrentTimePlayed ← true
17 *Buffer*[*Idx*] ← *Content*
18 E ← max(*Idx*+1, E)

These processes correspond to the three blocks building the generic dynamic score given in algorithm 1. It contains the scheduling of the calls to the model described in 4, and the sequence triggering:

1. Listening to update of current date orchestrates labeling and learning of musical material, and playing of anticipated events stored in the buffer (lines 1-6 in algorithm 1).

2. When a new element generated by the model is received (lines 12-18 in algorithm 1), it is stored in the buffer or immediately played managing potential delays.

3. A query to generate a segment of improvisation starting at date $T' \geq T$ associated to suffix a $S_{T'}$ of the scenario can be sent if it is required.

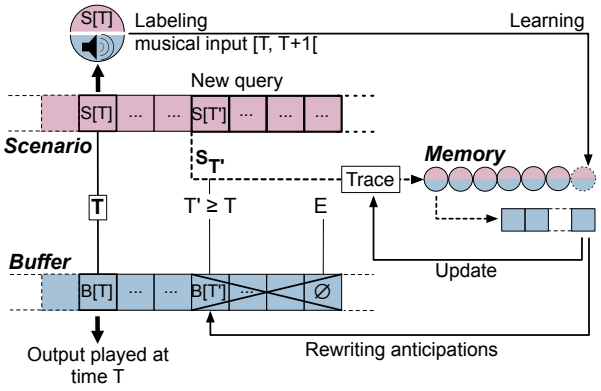

Figure 3. Processes launched at time T of the performance, reaction to a modification affecting time $T' \geq T$.

The model is called to generate the improvisation phase (see 4.2) after phase, anticipating the performance time. A query launching a new generation phase is thus sent if the minimum anticipation delay between the current time T and the first date with no element previously generated and stored in the buffer E is reached (lines 7-9 in algorithm 1). Figure 3 gives an overview of different processes running at a given time T of performance. In reaction to modification of a parameter or an update of scenario affecting a date $T' \geq T$, a query launching a new generation phase is sent (lines 10-11 in algorithm 1). If this query overlaps with a previous one, the associated anticipated part of the improvisation is rewritten. This scheduling architecture combines anticipation and reactive controls. The *short-term intentions* stored in the buffer evolve and are refined as the performance goes. A *trace* records history of paths in the memory and constraints of last navigation phases so that coherence between successive phases associated to overlapping suffixes of the scenario is maintained. This way for example, constraints can be imposed in real-time, and short-term memory can immediately be injected while keeping up with the scenario. A "reaction" is not seen here as a spontaneous instant response, but as a revision of the short-term intentions matching the scenario in the light of new events from the environment.

6. CHAINING REACTIONS TO UNORDERED COMPLEX EVENTS

6.1 From scheduling to logical planning

The generative model introduced in section 4 was implemented as a Common Lisp library using the OpenMusic environment [28]. The memory and the navigation methods are called in this interaction context through a producer-consumer system which involves parallel processes sharing accesses to the memory and the information received from the dynamic score (algorithm 1). This structure containing the scheduling of the calls to the model, the buffer, and the triggering of the playing of the computed sequences is implemented using *Antescofo* language. The corresponding program is generic enough to be independent of improvisation situations and of types of scenario. The improvisations generated by the model are then played in synchrony with the musical environment, following the fluctuation of the tempo. The synchronization strategies to manage the delays (lines 13-16 algorithm 1) associated to anticipation are used to maintain musical coherence despite real-time modifications of generation parameters.

Beyond scheduling aspects detailed in the previous paragraph, this section presents *Antescofo* features allowing specification of high-level temporal structures of this model and also more generally of different kind of processes that can be involved in an improvised interactive music performance. Recent developments of the language integrate handling of dynamic duration, complex events specification and dynamic processes. This generalizes the notion of score following beyond triggering of an action or recognition of causal events. The score is no longer subject to linear rigidity of classical scores. It can be seen more as an interactive system where events, actions, durations, tempi and all the temporal structures can change dynamically. Such features make it an adequate environment for the temporal coordination of the processes that can be involved in Human-Computer Improvisation.

6.2 The `whenever` statement

The `whenever` statement launches actions conditionally on the occurrence of a signal.

```
whenever (predicate)
{
  actions-list
} until (expression)
```

`predicate` is an arbitrary expression. Each time the variables of `predicate` are updated, the expression is re-evaluated. The `whenever` statement is a way to reduce and simplify the specification of the score particularly when actions have to be executed each time an event is detected. It also escapes the sequential nature of traditional scores. Actions of a whenever statement are not statically associated to an event of the performer but to the dynamic satisfaction of some predicate. They can be triggered as a result of a complex calculation, launched by external events, or any combinations of the above.

6.3 Patterns

The `whenever` structure is relevant when the user wants to define a reaction conditionally to the occurrence of an event. A logical event is specified thanks to a combination of variables. Complex events corresponding to a combination of atomic events with particular temporal constraints are however tedious to specify. *Antescofo* patterns make the definition of this kind of events concise and easy. A pattern is made of punctual events (`EVENT`) and of events with some duration (`STATE`). The example below shows how to define the complex event `pattern::P`, matching the following configuration: the variable `$x` is greater than

a threshold equal to 10 during 0.5s. Then the variable is updated to 1 before one second is elapsed. This pattern can then be used in the specification of a `whenever` condition to assign a reaction to this event.

```
@pattern_def pattern::P
{
  STATE $x where($x>10) during 0.5s
  before 1s
  EVENT $y where($y=1)
}
whenever(pattern::P)
{
 actions...
}
```

6.4 Processes

Processes are groups of actions dynamically instantiated. Unlike the other actions, the runtime structure associated to a process is not created during the loading of the score but at the time of the call, in accordance with its definition. Then, all the expressions involved in the process (durations, command names, attributes, etc.) may depend on parameters of the performance.

```
@proc_def ::delay($pitch, $d)
{
 $d/3 loop $d/3
 {
  play $pitch
 }during [2#]
}
NOTE C4 2.
  ::delay("C4", 2.)
NOTE D3 1
  ::delay("D3", 1)
```

In the previous example, the process `::delay` repeats twice the note that triggered it during the duration of this note. Processes are first-class values: for example, a process can be passed as an argument to a function or an other process. It can be recursively defined and various instances of a same process can be executed in parallel. Processes are quite adapted to the context of improvised music, and can be used for example as a library of parametrized musical phrases that are instantiated following the musical context.

6.5 Writing improvisation *plans*

The set of tools presented in this section enables to write *improvisation plans* defining different kinds of interactions. The schematic example in figure 4 shows an generic example of such a plan. In this context, the score of the musician is not completely defined and the inputs of the reactive module are not only extracted from *Antescofo* listening machine but can also be provided by external modules.

Each state corresponds to an interaction mode between the performer and the system. Satisfaction of temporal patterns `p1`, `p2` or `p3` allows to switch between the different states `s0`, `s1`, `s2` and `s3`. These patterns can for example be defined as temporal evolutions of some audio descriptors. `s0` is associated to a classical phase of interaction of a score following system with electronic actions adapting to a sequence of predefined events. Reaching the end of the sequence leads to the beginning of the next part (`s1`) where the musician improvises with the generation model guided by a scenario chosen as a given harmonic

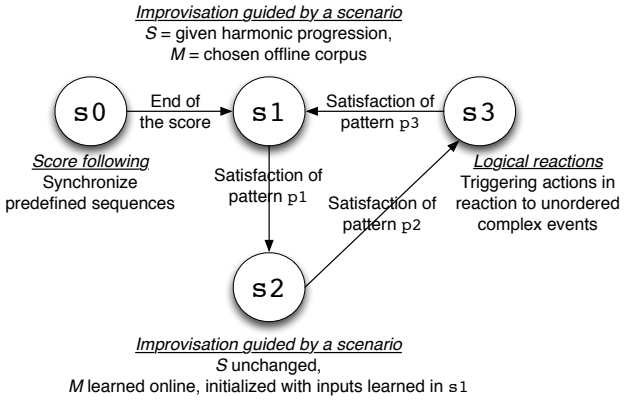

Figure 4. Schematic example of an improvisation plan.

progression, and a musical memory initialized with a chosen corpus. The part corresponding to `s2` continues with the same scenario using the memory learned from the musician's performance during `s1`. Finally, `s3` is a combination of predefined interactive mechanisms associating electronic reactions to unordered events.

7. CONCLUSION

This paper sketches an environment allowing the temporal management of human-computer improvisation. In this approach, improvisation plans with different kinds of interactions at multiple time-scales were considered ranging from completely determined to unordered events and reactions. The proposed environment integrates a model of generation for the improvisation in the high-level structure provided by a language for the specification and coordination of electronic actions depending on defined events. This coupling of high-level dynamic language with an improvisation model (generative processes), which is at the core of the prototype presented in this paper, enhances the coupling between interactive computer music and improvised practices. Researches on those lines could address complex problems with simple and elegant solutions useful for both music planning and improvised live performance.

Acknowledgments

The authors would like to thank Jean Bresson, Arshia Cont and Gérard Assayag for fruitful discussions. This work is supported in part by ANR projects: IMPROTECH ANR-09-SSOC-068 and INEDIT ANR-2012-CORD-009-03.

8. REFERENCES

[1] R. Rowe, "The aesthetics of interactive music systems," *Contemporary music review*, vol. 18, no. 3, pp. 83–87, 1999.

[2] J. Nika and M. Chemillier, "ImproteK, integrating harmonic controls into improvisation in the filiation of OMax," in *Proc. of the International Computer Music Conference*, Ljubljana, 2012, pp. 180–187.

[3] A. Cont, "On the creative use of score following and its impact on research," in *Proc. International Conference on Sound and Music Computing*, Padova, July 2011.

[4] G. Assayag, G. Bloch, M. Chemillier, A. Cont, and S. Dubnov, "OMax brothers: a dynamic topology of agents for improvization learning," in *Proc. of the 1st ACM workshop on Audio and music computing multimedia*, ACM, Santa Barbara, California, 2006, pp. 125–132.

[5] B. Lévy, G. Bloch, and G. Assayag, "OMaxist dialectics," in *Proc. of the International Conference on New Interfaces for Musical Expression*, 2012, pp. 137–140.

[6] A. R. François, I. Schankler, and E. Chew, "Mimi4x: an interactive audio–visual installation for high–level structural improvisation," *International Journal of Arts and Technology*, vol. 6, no. 2, pp. 138–151, 2013.

[7] L. Bonnasse-Gahot, "Online arrangement through augmented musical rendering," *Ircam STMS Lab Internal Report - ANR Sample Orchestrator 2, ANR-10-CORD-0018*, 2013.

[8] J. Moreira, P. Roy, and F. Pachet, "Virtualband: Interacting with stylistically consistent agents," in *Proc. of International Society for Music Information Retrieval Conference*, Curitiba, 2013, pp. 341–346.

[9] F. Pachet, P. Roy, J. Moreira, and M. d'Inverno, "Reflexive loopers for solo musical improvisation," in *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, Paris, 2013, pp. 2205–2208.

[10] G. Surges and S. Dubnov, "Feature selection and composition using PyOracle," in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, Boston, Massachusetts, 2013.

[11] A. Donze, S. Libkind, S. A. Seshia, and D. Wessel, "Control improvisation with application to music," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-183, Nov 2013.

[12] M. Puckette, "Combining event and signal processing in the max graphical programming environment," *Computer Music Journal*, vol. 15, pp. 68–77, 1991.

[13] J. McCartney, "Supercollider: a new real-time synthesis language," in *Proc. of the International Computer Music Conference*, Hong Kong, 1996.

[14] G. Wang, "The chuck audio programming language." a strongly-timed and on-the-fly environ/mentality"," Ph.D. dissertation, Princeton University, 2009.

[15] A. Cont, "Antescofo: Anticipatory synchronization and control of interactive parameters in computer music," in *Proc. of the International Computer Music Conference*, Belfast, 2008.

[16] ——, "A coupled duration-focused architecture for realtime music to score alignment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 6, pp. 974–987, 2010.

[17] S. Dubnov, G. Assayag, and R. El-Yaniv, "Universal classification applied to musical sequences," in *Proc. of the International Computer Music Conference*, Ann Arbor, Michigan, 1998, pp. 332–340.

[18] G. Assayag, S. Dubnov, and O. Delerue, "Guessing the composer's mind: Applying universal prediction to musical style," in *Proc. of the International Computer Music Conference*, Beijing, 1999, pp. 496–499.

[19] M. Chemillier, "L'improvisation musicale et l'ordinateur," *Terrain*, no. 2, pp. 66–83, 2009.

[20] ——, "Improviser des séquences d'accords de jazz avec des grammaires formelles," in *Proc. of Journées d'informatique musicale*, Bourges, 2001, pp. 121–126, (English summary). [Online]. Available: http://ehess.modelisationsavoirs.fr/marc/publi/jim2001/jim2001english.pdf

[21] ——, "Toward a formal study of jazz chord sequences generated by Steedman's grammar," *Soft Computing*, vol. 8, no. 9, pp. 617–622, 2004.

[22] J. H. Morris and V. R. Pratt, *A linear pattern-matching algorithm*, 1970.

[23] M. Crochemore, C. Hancart, and T. Lecroq, *Algorithms on strings*. Cambridge University Press, 2007.

[24] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor oracle: A new structure for pattern matching," in *SOFSEM 99: Theory and Practice of Informatics*. Springer, 1999, pp. 758–758.

[25] A. Lefebvre, T. Lecroq, and J. Alexandre, "Drastic improvements over repeats found with a factor oracle," 2002.

[26] G. Assayag and S. Dubnov, "Using factor oracles for machine improvisation," *Soft Computing*, vol. 8, no. 9, pp. 604–610, 2004.

[27] G. Assayag and G. Bloch, "Navigating the oracle: A heuristic approach," in *Proc. of the International Computer Music Conference*, Copenhagen, 2007, pp. 405–412.

[28] J. Bresson, C. Agon, and G. Assayag, "OpenMusic: visual programming environment for music composition, analysis and research," in *Proc. of the 19th ACM international conference on Multimedia*, Scotssdale, 2011, pp. 743–746.