Guided improvisation as dynamic calls to an offline model

Jérôme Nika, Dimitri Bouche, Jean Bresson, Marc Chemillier, Gérard Assayag

Guided improvisation as dynamic calls to an offline model

Jérôme Nika[1,2], **Dimitri Bouche**[1,2], **Jean Bresson**[1,2], **Marc Chemillier**[3], **Gérard Assayag**[1,2]
[1] IRCAM, UMR STMS 9912 CNRS, [2] Sorbonne Universités UPMC,
[3] Cams, Ecole des Hautes Etudes en Sciences Sociales
{jnika,bouche,bresson}@ircam.fr, chemilli@ehess.fr, assayag@ircam.fr

ABSTRACT

This paper describes a reactive architecture handling the hybrid temporality of guided human-computer music improvisation. It aims at combining reactivity and anticipation in the music generation processes steered by a "scenario". The machine improvisation takes advantage of the temporal structure of this scenario to generate short-term anticipations ahead of the performance time, and reacts to external controls by refining or rewriting these anticipations over time. To achieve this in the framework of an interactive software, guided improvisation is modeled as embedding a compositional process into a reactive architecture. This architecture is instantiated in the improvisation system ImproteK and implemented in OpenMusic.

1. INTRODUCTION

Human-computer improvisation systems generate music on the fly from a model and external inputs (typically the output of an "analog" musician's live improvisation). Introducing authoring and control in this process means combining the ability to react to dynamic controls with that of maintaining conformity to fixed or dynamic specifications.

This paper describes an architecture model (instantiated in the improvisation software ImproteK) where the specification is a formal structure, called *scenario*, that has to be followed during the performance. The scenario enables to anticipate the future and to generate music ahead of the current time. In this context, the system reactions to changes and external controls should use this knowledge of what is expected to generate an updated future. In addition, if the initial specification itself gets modified during the performance, the system may have to ensure a continuity with the past generated material at critical times.

To achieve this, a *scenario/memory* generation model is embedded in a reactive agent called *improvisation handler* which translates dynamic controls from the environment into music generation processes. This agent is in continuous interaction with an *improvisation renderer* designed as a scheduling module managing the connection with the real performance time.

After briefly discussing the existing approaches in guided improvisation systems in section 2, section 3 describes the scenario/memory generation model and gives some musical directions related to guided (or composed) improvisation. Then, section 4 describes how anticipation and dynamic controls are combined by embedding this generation model into the reactive improvisation handler. Finally, section 5 presents the improvisation renderer interweaving calls to the improvisation handler, scheduling and rendering of the generated material.

2. RELATED WORK

A number of existing improvisation systems drive the music generation processes by involving a user steering their parameters. In a first approach this user control can concern (low-level) system-specific parameters. This is for example the case of OMax [1, 2] or Mimi4x [3].

We refer here to *guided improvisation* when the control on music generation follows a more "declarative" approach, i.e. specifying targetted outputs or behaviours using an aesthetic, musical, or audio vocabulary independent of the system implementation. On the one hand, *guiding* is seen as a purely reactive and step by step process. SoMax [4] for instance translates the musical stream coming from an improviser into activations of specific zones of the musical memory in regards to a chosen dimension (for example the harmonic background). VirtualBand [5] and Reflexive Looper [6] also extract multimodal observations from the musician's playing to retrieve the most appropriate musical segment in the memory in accordance to previously learnt associations.

On the other hand, *guiding* means defining upstream temporal structures or descriptions driving the generation process of a whole improvisation sequence. Pachet and Roy [7] for instance use constraints in such generation process. In the works of Donzé *et al.* [8] the concept of "control improvisation" [9] applied to music also introduces a guiding structure via a reference sequence and a number of other specifications. This structure is conceptually close to the *scenario* used in our approach. PyOracle [10] proposes to create behaviour rules or scripts for controlling the generation parameters of an improvisation generation using "hot spots" (single event targets). Wang and Dubnov [11] extend this work in an offline architecture using sequences instead of single events as query targets. This idea of mid-term temporal queries is close to the musical issues raised with the notion of *dynamic scenario* in this paper.

Two conceptions of time and interactions are actually emphasized in these different approaches. The purely reactive one offers rich interaction possibilities but does not integrate prior knowledge about the temporal evolution. On the other hand, steering music generation with mid- or long-term structures enables anticipation but lacks reactivity with regard to external or user controls.

This bi-partition in improvisation systems actually reflects the offline/online paradigmatic approaches in computer music systems regarding time management and planning/scheduling strategies. On the one hand, "offline" corresponds to computer-aided composition systems [12] where musical structures are computed following best effort strategies and where rendering involves static timed plans (comparable to timed priority queues [13]). In this case, the scheduling only consists in traversing a precomputed plan and triggering function calls on time. On the other hand, "online" corresponds to performance-oriented systems [14] where the computation time is part of the rendering, that is, computations are triggered by clocks and callbacks and produce rendered data in real-time [15]. In this case, only scheduling strategies matter and no future plan is computed.

3. GUIDED MUSIC IMPROVISATION AND DYNAMIC CONTROLS

Our objective is to devise an architecture at an intermediate level between the reactive and offline approaches for guided improvisation, combining dynamic controls and anticipations relative to a predefined plan.

The proposed architecture is structured around an offline generation module based on a scenario, embedded in a reactive framework and steered/controlled by external events. This generation module produces short-term anticipations matching its scenario, and may eventually rewrite these anticipations over time according to incoming control events.

3.1 Scenario/memory generation model

The offline generation process consists in finding a path matching the scenario through a structured and labeled memory, where:

- the *scenario* is a symbolic sequence of labels defined over an alphabet,
- the *memory* is a sequence of musical contents labeled by letters of the same alphabet.

The scenario can be any sequence defined over an arbitrary alphabet depending on the musical context, for example a harmonic progression in the case of jazz improvisation or a discrete profile describing the evolution of audio descriptors for the control of sound synthesis processes. The memory can be constituted online (recorded during a live performance) or offline (from annotated audio or MIDI files).

This model is used for example to improvise on a given chord chart using a memory constituted by live inputs and heterogenous set of jazz standards recordings as memory. The contents of the memory can be audio slices [1], MIDI notes or events [2], parameters for sound synthesis, etc.

The scenario/memory model is implemented in ImproteK [16, 17], a co-improvisation system inheritor of the software environment OMax, which specifically addresses the issues of authoring and control in human-computer improvisation. This generation model is a priori offline in the sense that one run produces a whole timed and structured musical gesture satisfying the designed scenario, which will then be unfolded through time during performance. Furthermore, it follows a compositional workflow: 1) chose or define an alphabet for the labels and describe its properties, 2) compose at the structure level (i.e. define a scenario).

3.2 In-time reaction

In the scope of music improvisation guided by a scenario, a *reaction* of the system to dynamic controls cannot be seen as a spontaneous instant response. The main interest of using a scenario is indeed to take advantage of this temporal structure to anticipate the music generation, that is to say to use the prior knowledge of what is expected for the future in order to better generate at the current time. Whether a reaction is triggered by a user control, by hardcoded rules specific to a musical project, or by an analysis of the live inputs from a musician, it can therefore be considered as a revision of the mid-term anticipations of the system in the light of new events or controls.

To deal with this temporality in the framework of a real-time interactive software, we consider guided improvisation as embedding an offline process into a reactive architecture. In this view, reacting amounts to composing a new structure in a specific timeframe ahead of the time of the performance, possibly rewriting previously generated material. The synchronization with the environment and the management of high-level temporal specifications are handled by a dynamic score launching the calls to the generation model. This module, using the environment Antescofo and the associated programming language [18, 19], is described in [16].

In this paper, we focus on handling these reactive calls to combine anticipation relative to the scenario and dynamic controls, by proposing an architecture made of two main components:

- An *improvisation handler* embedding the scenario/ memory articulations and generating musical sequences on request.
- An *improvisation renderer* handling the temporality and interactions in a system run.

After mentioning some musical issues raised by guided improvisation, we will introduce these two agents (both implemented in the OpenMusic [20] environment) in more details and formalize their interactions in sections 4 and 5.

[1] **See videos:** `http://repmus.ircam.fr/nika/ImproteK`
[2] **See videos:** `http://improtekjazz.org`

3.3 Playing with the scenario and with the dynamic controls

The articulation between the formal abstraction of "scenario" and reactivity enables to explore different musical directions with the same objects and mechanisms, providing dynamic musical control over the improvisation being generated.

In first approach, we differentiate two playing modes depending on the hierarchy between the musical dimension of the scenario and that of control. When scenario and control are performed on different features of the musical contents (3.3.1), the model combines long-term structure with local expressivity. When scenario and dynamic controls act on the same musical feature (3.3.2), it deals with dynamic guidance and intentionality.

3.3.1 Long-term structure and local expressivity

We firstly consider the case where the specification of a scenario and the reaction concern different features, conferring them different musical roles (for example: defining the scenario as a harmonic progression and giving real-time controls on density, or designing the scenario as an evolution in register and giving real-time controls on energy). In this case, a fixed scenario provides a global temporal structure on a conduct dimension, and the reactive dimension enables to be sensitive to another musical parameter. The controlled dimension has a local impact, and deals with expressivity by acting at a secondary hierarchical level, for example with instant constraints on timbre, density, register, syncopation etc.

This playing mode may be more relevant for idiomatic [21] or composed improvisation with any arbitrary vocabulary, in the sense that a predefined and fixed scenario carries the notions of high-level temporal structure and formal conformity to a given specification anterior to the performance, as it is the case for example with a symbolic harmonic progression.

3.3.2 Guidance and intentionality

When specification and reaction act on the same musical dimension, the scenario becomes dynamic. A reaction does not consist in a local control on a secondary parameter as in the previous playing mode, but in the modification of the scenario itself.

In this case, the current state of a dynamic scenario at each time of the performance represents the short-term "intentionality" attributed to the system, which becomes a reactive tool to guide the machine improvisation by defining instant queries with varying time windows. The term "scenario" may be inappropriate in this second approach since it does not represent a fixed general plan for the whole improvisation session. Nevertheless, we will use this term in the following sections whether the sequence guiding the generation is dynamic or static (i.e. whether the reaction impacts the guiding dimension or another one) since both cases are formally managed using the same mechanisms.

4. EMBEDDING AN OFFLINE GENERATION MODEL INTO A REACTIVE ENVIRONMENT

Thanks to the scenario, music is produced ahead of the performance time, buffered to be played at the right time or rewritten. For purposes of brevity (and far from any anthropomorphism),

- *anticipations* will be used to refer to pending events: the current state of the already generated musical material ahead of the performance time,
- *intentions* will be used to refer to the planned formal progression: the current state of the scenario and other generation parameters ahead of the performance time.

This section presents how the evolving anticipations of the machine result from successive or concurrent calls to the generation model. Introducing a reaction at a time when a musical gesture has already been produced amounts then to rewrite buffered anticipation. The rewritings are triggered by modifications of the intentions regarding the scenario itself or other generation parameters (these two different cases correspond to the different musical directions introduced in 3.3).

4.1 Improvisation handler

To give control over these mechanisms, that is dynamically controlling improvisation generation, we define an *improvisation handler* agent (H) which contains and articulates the generation model with:

- a scenario (S);
- a set of generation parameters;
- current position in the improvisation t^p (*performance time*);
- the index of the last generated position t^g (*generation time*);
- a function f responsible for the output of generated fragments of improvisation (*output method*).

This improvisation handler agent H links the real time of performance and the time of the generation model embedded in an *improviser* structure (see Figure 1). The improviser structure associates the generation model and the memory with a set of secondary generation parameters and an execution trace described below.

The set of *generation parameters* contains all the parameters driving the generation process which are independent from the scenario: parametrization of the generation model (e.g. minimal / maximal length or region of the sub-sequences retrieved from the memory, measure of the linearity/non-linearity of the paths in the memory etc.) and content-based constraints to filter the set of possible results returned by the scenario matching step (e.g. user-defined thresholds, intervals, rules etc.).

The *execution trace* records history of paths in the memory and states of these generation parameters for the last runs of the generation model so that coherence between

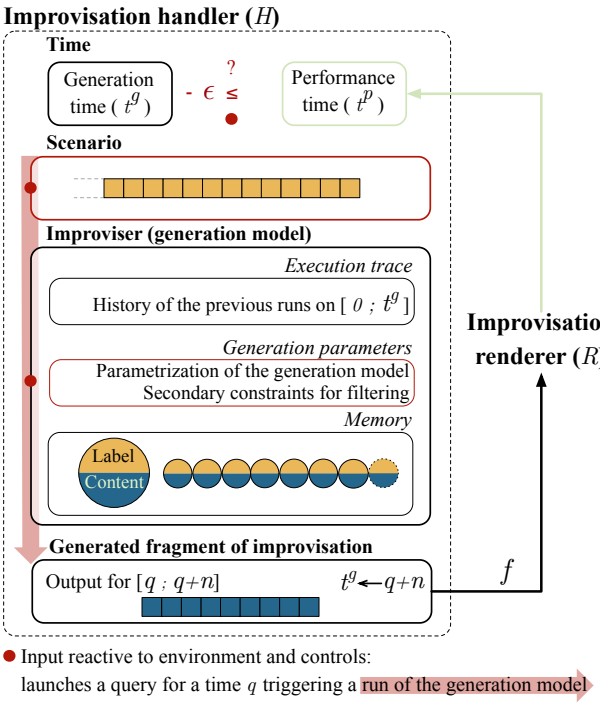

Figure 1. *Improvisation handler* agent.

successive generations phases associated to overlapping queries is maintained. This way, the process can go back to an anterior state to insure continuity at the first position where the generation phases overlap.

The interactions of the improvisation handler with the environment consist in translating dynamic controls on reactive inputs into reactive queries and redirecting the resulting generated fragments. We call *reactive inputs* the entities whose modifications lead to a reaction: the scenario and the set of generation parameters. In this framework, we call *reaction* an alteration of the *intentions* leading to a call to the generation model to produce a fragment of improvisation starting at a given position in the scenario.

We note Q a *query* launched by a reaction to generate an improvisation fragment starting at time q in the scenario [3]. Q triggers a run of the improviser to output a sub-sequence (or a concatenation of sub-sequences) of the memory which:

- matches the current state of the scenario from date q (i.e. a suffix S_q of the scenario, see [17]),
- satisfies the current state of the set of generation parameters.

The *output method* of the improvisation handler (f) is a settable attribute, so that generated improvisations can be redirected to any rendering framework. For instance, the improvisation handler can interface with Max via the dynamic score written in the Antescofo language mentioned in 3.2. In this case, f determines how resulting improvisation segments are sent back to the dynamic score where they are buffered or played in synchrony with the non-metronomic tempo of the improvisation session. Section 5 details how f is used to couple the improvisation handler with an improvisation renderer in order to unify music generation, scheduling and rendering.

[3] q is the time at which this fragment will be played, it is independent from t^p and from the date at which the query is launched by the improvisation handler.

4.2 Triggering queries for rewriting anticipations

We describe here the way control events are translated into generation queries triggered by the improvisation handler. This mechanism can be *time-triggered* or *event-triggered*, i.e. resulting respectively from depletion of previously generated material or from parameters modifications.

4.2.1 Time-triggered generation

Rendering may lead to the exhaustion of generated improvisation. New generation queries have therefore to be launched to prevent the time of the generation t^g from being reached by the time of the performance t^p. To do so, we define ϵ as the maximum allowed margin between t^p and t^g. Consequently, a new query for time $q = t^g + 1$ is automatically triggered when the condition $t^g - t^p \leq \epsilon$ becomes true.

Depletion of the previously generated improvisation generation occurs when generation over the whole scenario is not performed in a single run. Figure 2 illustrates two successive generation phases associated to queries Q_1 and Q_2 for time q_1 and q_2 respectively. A generation *phase* matches a scenario sub-sequence starting at a queried position q to a sub-sequence of the memory, i.e. the generation model searches for a prefix of the suffix S_q of the scenario S in the memory (phase q_1 in figure 2) or an equivalent non-linear path (phase q_2 in figure 2). The generation process waits then for the next query.

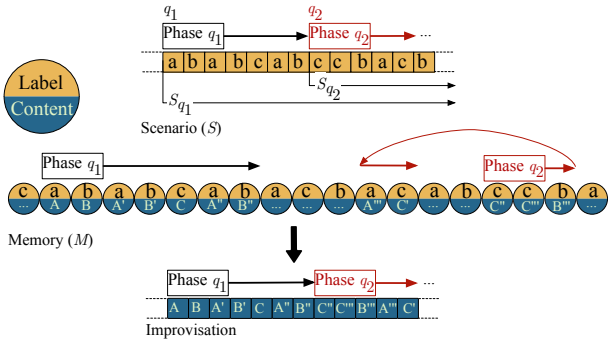

Figure 2. Phases of the guided generation process.

Defining such phases enables to have mid-term anticipations generated ahead of the performance time while avoiding generating over the whole scenario if an event modifies the intentions.

A generated fragment of improvisation resulting from a query Q for time q contains n slices where:

$$1 \leq n \leq length(S) - q, n \in \mathbb{N}$$

The search algorithm of the generation model runs a generation phase [4] to output a sub-sequence of the memory in time $\Theta(m)$ and does not exceed $2 * m - 1$ comparisons, where m is the length of the memory. In first approximation, the maximum margin ϵ is empirically initialized with a value depending on the initial length m. Then, in order to take into account the linear time complexity, ϵ increases proportionally to the evolution of m if the memory grows as the performance goes. Future works on this point will consist in informing the scheduling engine with the similarities between the scenario and the memory to optimize anticipation. Indeed, the number of calls to the model depends on the successive lengths n of the similar patterns between the scenario and the memory. For example, the shorter the common factors, the higher the number of queries necessary to cover the whole scenario.

4.2.2 Event-triggered generation

As introduced previously, the musical meanings of a reaction of the improvisation handler impacting the scenario itself (3.3.2) or an other musical dimension (3.3.1) are quite different. Yet, both cases of reaction can be formally managed using the same mechanisms of event-triggered generation. The reactive inputs (4.1) are customizable so that any relevant slot of the improvisation handler can easily be turned into a reactive one. Modifying the scenario or one of these reactive slots launches a generation query for the time q affected by this modification. The triggering of a query by a reaction can indeed take effect at a specified time q independent of performance time t^p.

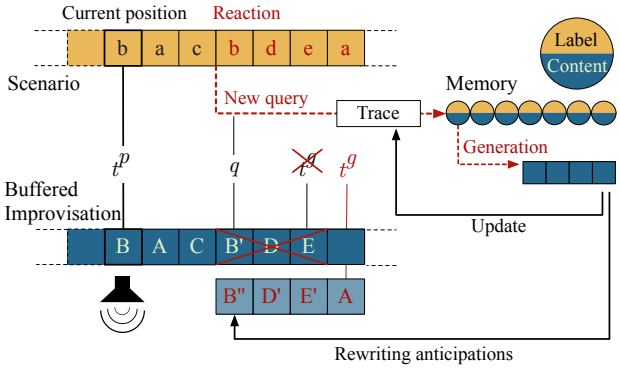

Figure 3. Reactive calls to the generation model.

As illustrated in figure 3, the new improvisation fragment resulting from the generation is sent to the buffered improvisation while the improvisation is being played. The new fragments overwrites the previously generated material on the overlapping time interval. [5] The execution trace introduced in 4.1 enables to set mechanisms providing continuity at the tiling time q.

[4] 1) Index the prefixes of the suffix S_q of the scenario in the memory, 2) select one of these prefixes depending on the generation parameters, 3) output this prefix or an equivalent non-linear path.

[5] **See video:**
`https://vimeo.com/jeromenika/improteksmc15`

4.3 Rewriting intentions: concurrent queries

Anticipation may be generated without ever being played because it may be rewritten before being reached by the time of the performance. Similarly, an intention may be defined but never materialized into anticipation if it is changed or enriched by a new event before being reached by a run of generation.

Indeed, if reactions are frequent or defined with delays, it would be irrelevant to translate them into as many independent queries leading to numerous overlapping generation phases. We then define an intermediate level to introduce evolving queries, using the same principle for dynamically rewriting intentions as that defined for anticipations.

This aspect is dealt with by handling concurrency and working at the query level when the improvisation handler receives new queries while previous ones are still being processed by the generation module. Algorithm 1 describes how concurrency is handled, with:

- Run(Q): start generation associated to Q. This function outputs generated data when it finishes,
- Kill(Q): stop run associated to Q and discard generated improvisation,
- Relay(Q_1,Q_2,q): output the result of Q_1 for $[q_1; q[$, kill Q_1 and run Q_2 from q. The execution trace is read to maintain coherence at relay time q,
- WaitForRelay(Q_1,Q_2,q): Q_2 waits until Q_1 generates improvisation [6] at time q. Then Relay(Q_1,Q_2,q).

Algorithm 1 Concurrent runs and new incoming queries

Q_i, query for improvisation time q_i
RQ, set of currently running or waiting queries
$CurPos(Q)$, current generation index of Run(Q)

when new Q received **do**
for $Q_i \in RQ$ **do**
  if $q = q_i$ **then**
    if Q and Q_i from same inputs **then**
      Kill(Q_i)
    else
      Merge Q and Q_i
    end if
  else if $q > q_i$ **then**
    if $q < CurPos\ (Q_i)$ **then**
      Relay(Q_i,Q,q)
    else
      WaitForRelay(Q_i, Q, q)
    end if
  else if $q < q_i$ **then**
    WaitForRelay(Q, Q_i, q_i)
  end if
end for

[6] More precisely, new generation phases are launched if needed until q is reached.

This way, if closely spaced in time queries lead to concurrent processing, relaying their runs of the generation model at the right time using the execution trace enables to merge them into a "dynamic query".

5. RENDERING AND SCHEDULING: IMPROVISATION RENDERER

The reactive architecture presented in the previous section embeds the data, specifications, and mechanisms managing music generation, reaction, and concurrency in the *improvisation handler*. This improvisation handler receives from the environment: (1) the live inputs (in the case of online learning), (2) the control events, and (3) the current performance time t^p. In return, it sends improvisation fragments back to the same environments (4).

Live inputs and interaction (1,2) are managed autonomously by the improvisation handler. The connection with the real performance time (3,4) is managed in an unified process through the continuous interaction with an *improvisation renderer*. This renderer is designed as a scheduling module which runs in parallel to the improvisation handler all along the performance.

5.1 Scheduling strategy

To describe the scheduling architecture we need to introduce a number of additional concepts: an *action* is a structure to be executed (including a function and some data); a *plan* is a list of ordered timed actions; the *planner* is an entity in charge of extracting plans from musical structures; and the *scheduler* is the entity in charge of rendering plans.

A hierarchical model is used to represent musical data (for example, a chord as a set of notes, a note as a set of MIDI events...) and to synchronize datasets rendering. To prepare a musical structure rendering, the planner converts the object into a list of timed actions. Triggering the rendering of a "parent object" synchronizes the rendering of its "children" [7] . Then, the scheduler renders the plan, i.e. triggers the actions on time [24].

The planner and scheduler cannot operate concurrently on a same plan, but they can cooperate. Scheduling is said *dynamic* when the scheduler is likely to query the planner for short-term plans on the fly, and/or when the planner is likely to update plans being rendered by the scheduler [25, 26]. Our strategy is based on a *short-term lookahead* planner: instead of planning a list of actions representing the whole content of a musical object, the planner is called on-time by the scheduler and outputs plans applicable in a specified time window.

The flowchart on figure 4 summarizes the plan extraction algorithm used by the scheduler to render musical objects. Typically, the scheduler calls the planner for a plan applicable in a time window W of duration w, then the scheduler can render this short-term plan on time and query the planner for the next one. The lower w, the most reactive the system is, at a cost of more computations (w can be tweaked accordingly). If the planner returns no plan (i.e. there is nothing to render in the queried time interval), the scheduler can query again for the next time window until a plan is returned. Therefore, the time window W can be far ahead of the actual rendering time of the structure, and might not be the same across concurrently rendered objects. Plan queries themselves can also be planned as actions to execute in the future. For instance, a limit of successive plan queries can be set to avoid overload (e.g. if there is nothing else to play): in this case sparse planning queries can be planned at the end of each time windows.

[7] As introduced in [22]. In terms of scheduling, the hierarchical representations also eases the development of optimized strategies [23].

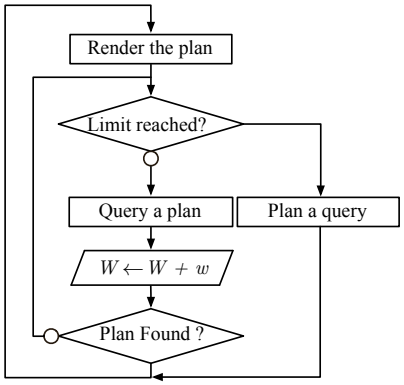

Figure 4. Short-term plan extraction flowchart.

5.2 Application for guided improvisation

The *improvisation renderer* (R) connected to the improvisation handler:

- receives and renders the produced fragments,
- communicates the current performance time t^p.

With regard to the scheduling architecture, R is a structure containing two children objects, the mutable priority queues:

- RC (*render action container*) containing actions to render, extracted from improvisation fragments.
- HC (*handler action container*) containing time marker actions to send back to the handler H.

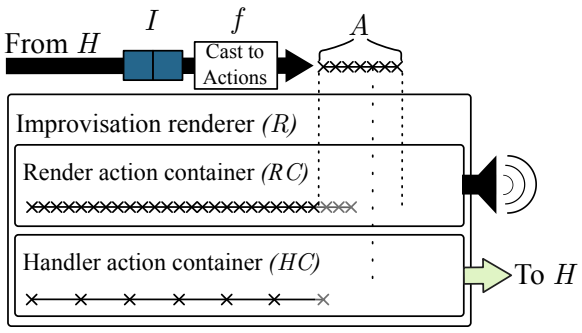

Figure 5. The *improvisation renderer*.

Figure 5 depicts the improvisation renderer and its communication with the improvisation handler. An improvisa-

tion fragment I is outputted from the improvisation handler, and this fragment is casted into a list of actions A integrated into the render action container RC. This translation can be defined depending on the type of improvised data. For instance, if the improvisation slices contain MIDI, actions will consist in calls to *midi-on* and *midi-off*. If the list of actions is overlapping with existing content (i.e. with previously generated fragments of improvisation already stored as actions in RC), the new actions substitute the overlap and add the last generated improvisation to RC. At the same time, information about slices timing of I is extracted to feed the handler action container HC with time markers that will be sent back on time to the improvisation handler.

To perform the previous operations, we define the following functions:

- $Cast(I)$: cast an improvisation fragment I into a list of timed actions A,
- $Timing(I)$: extract a list of actions from an improvisation fragment I, corresponding to the slices' respective times,
- $Tile(C, A)$: integrate an action list A in the action container C, overwriting the overlapping actions.

In order to connect the improvisation handler (section 4.1) to the improvisation renderer, the output method f of the improvisation handler shall therefore be the function of an improvisation fragment I and the improvisation renderer R defined as:

$$f(I, R) = \begin{cases} Tile(RC, Cast(I)) \\ Tile(HC, Timing(I)) \end{cases}$$

R can then be planned and rendered as a standard musical object, although this object will be constantly growing and changing according to performer's inputs or user controls. The short-term planning strategy will allow for changes not to affect the scheduled plans if they concern data at a time ahead of the performance time by at least w. In the contrary case (if data is modified inside the current time window) a new short-term plan extraction query is immediately triggered to perform a re-planning operation.

6. CONCLUSIONS AND PERSPECTIVES

We presented an architecture model to combine dynamic controls and anticipation with respect to a formal structure for guided human-computer music improvisation. It is achieved by embedding offline processes into a reactive framework, out of the static paradigm yet not using pure last moment computation strategies. It results in a hybrid architecture dynamically rewriting its musical output ahead of the time of the performance, in reaction to the alteration of the scenario or of a reactive parameter. The generation model and the improvisation handler agent are customizable and designed in such a way that it can be easily interfaced with any rendering environment (for example Antescofo/Max or OpenMusic). This architecture is instantiated in the improvisation software ImproteK.

In the scope of guided improvisation, the reactive architecture described in this paper proposes a model to answer the question "how to react?", but does not address the question "when to react and with what response?". Indeed, the model defines the different types of reactions that have to be handled and how it can be achieved. It choses to offer genericity so that reactions can be launched by an operator using customized parameters, or by a composed reactivity defining hardcoded ruled specific to a particular musical project. Yet, integrating approaches such as that developed in [4] could enable to have reactions launched from the analysis of live musical inputs.

Considering the genericity of the alphabets in the generation model and that of the reactive architecture presented in this paper, future works will focus on chaining agents (using the output of an improvisation handler as an input for an other). A preliminary study of such chaining will involve using an ascending query system through the tree of plugged units to avoid data depletion, and message passing scheduling between multiple agents [27] to ensure synchronization. Other perspectives suggest to make use of such reactive music generation to produce evolving and adaptive soundscapes, embedding it in any environment that generates changing parameters while including a notion of plot, as video games for example.

Acknowledgments

The authors would like to thank José Echeveste and Jean-Louis Giavitto for fruitful discussions. This work is supported by the French National Research Agency projects EFFICACe ANR-13-JS02-0004 and DYCI2 ANR-14-CE2 4-0002-01.

7. REFERENCES

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

[2] B. Lévy, G. Bloch, and G. Assayag, "OMaxist dialectics," in *12th International Conference on New Interfaces for Musical Expression*, Ann Arbor, MI, USA, 2012, pp. 137–140.

[3] 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.

[4] L. Bonnasse-Gahot, "An update on the SOMax project," *Ircam - STMS, Internal report ANR project Sample Orchestrator 2, ANR-10-CORD-0018*, 2014.

[5] J. Moreira, P. Roy, and F. Pachet, "Virtualband: Interacting with Stylistically Consistent Agents," in *14th International Society for Music Information Retrieval*, Curitiba, Brazil, 2013, pp. 341–346.

[6] F. Pachet, P. Roy, J. Moreira, and M. d'Inverno, "Reflexive Loopers for Solo Musical Improvisation," in *14th SIGCHI Conference on Human Factors in Computing Systems*, Paris, France, 2013, pp. 2205–2208.

[7] F. Pachet and P. Roy, "Markov constraints: steerable generation of markov sequences," *Constraints*, vol. 16, no. 2, pp. 148–172, 2011.

[8] A. Donzé, R. Valle, I. Akkaya, S. Libkind, S. A. Seshia, and D. Wessel, "Machine improvisation with formal specifications," in *40th International Computer Music Conference*, Athens, Greece, 2014, pp. 1277–1284.

[9] D. J. Fremont, A. Donzé, S. A. Seshia, and D. Wessel, "Control improvisation," *arXiv preprint arXiv:1411.0698*, 2014.

[10] G. Surges and S. Dubnov, "Feature selection and composition using pyoracle," in *9th Artificial Intelligence and Interactive Digital Entertainment Conference*, Boston, MA, USA, 2013.

[11] C. Wang and S. Dubnov, "Guided music synthesis with variable markov oracle," in *3rd International Workshop on Musical Metacreation*, Raleigh, NC, USA, 2014.

[12] G. Assayag, "Computer Assisted Composition Today," in *1st symposium on music and computers*, Corfu, Greece, 1998.

[13] M. Kahrs, "Dream chip 1: A timed priority queue," *IEEE Micro*, vol. 13, no. 4, pp. 49–51, 1993.

[14] R. Dannenberg, "Real-Time Scheduling and Computer Accompaniment," in *Current Directions in Computer Music Research*, Cambridge, MA, USA, 1989, pp. 225–261.

[15] P. Maigret, "Reactive Planning and Control with Mobile Robots," in *IEEE Control*, 1992, pp. 95–100.

[16] J. Nika, J. Echeveste, M. Chemillier, and J.-L. Giavitto, "Planning Human-Computer Improvisation," in *40th International Computer Music Conference*, Athens, Greece, 2014, pp. 1290–1297.

[17] J. Nika and M. Chemillier, "Improvisation musicale homme-machine guidée par un scénario temporel," *Technique et Science Informatique, Numéro Spécial Informatique musicale*, vol. 7, no. 33, pp. 651–684, 2015, (in french).

[18] J. Echeveste, A. Cont, J.-L. Giavitto, and F. Jacquemard, "Operational semantics of a domain specific language for real time musician-computer interaction," *Discrete Event Dynamic Systems*, vol. 4, no. 23, pp. 343–383, 2013.

[19] J. Echeveste, J.-L. Giavitto, and A. Cont, "A Dynamic Timed-Language for Computer-Human Musical Interaction," INRIA, Rapport de recherche RR-8422, 2013.

[20] J. Bresson, C. Agon, and G. Assayag, "OpenMusic. Visual Programming Environment for Music Composition, Analysis and Research," in *ACM MultiMedia 2011 (OpenSource Software Competition)*, Scottsdale, AZ, USA, 2011.

[21] D. Bailey, *Improvisation: its nature and practice in music*. Da Capo Press, 1993.

[22] X. Rodet, P. Cointe, J.-B. Barriere, Y. Potard, B. Serpette, and J.-P. Briot, "Applications and developments of the formes programming environment," in *9th International Computer Music Conference*, Rochester, NJ, USA, 1983.

[23] R. J. Firby, "An Investigation into Reactive Planning in Complex Domains," in *6th National Conference on Artificial Intelligence*, Seattle, WA, USA, 1987, pp. 202–206.

[24] D. Bouche and J. Bresson, "Planning and Scheduling Actions in a Computer-Aided Music Composition System," in *Scheduling and Planning Applications woRKshop, 25th International Conference on Automated Planning and Scheduling*, Jerusalem, Israel, 2015, pp. 1–6.

[25] M. E. desJardins, E. H. Durfee, J. Charles L. Ortiz, and M. J. Wolverton, "A Survey of Research in Distributed, Continual Planning," *AI Magazine*, vol. 20, no. 4, 1999.

[26] E. C. J. Vidal and A. Nareyek, "A Real-Time Concurrent Planning and Execution Framework for Automated Story Planning for Games," in *AAAI Technical Report WS-11-18*, 2011.

[27] J. Lee, M. J. Huber, E. H. Durfee, and P. G. Kenny, "UM-PRS: An implementation of the Procedural Reasoning System for multirobot applications," in *AIAA/NASA Conference on Intelligent Robotics in Field, Factory and Space*, Houston, TX, USA, 1994, pp. 842–849.