Improvised Musical Interaction with Creative Agents

Master Thesis

Submitted by

Marco Fiorini
mfiori21@student.aau.dk
Aalborg University Copenhagen

Supervised by

Stefania Serafin
Aalborg University Copenhagen
SMC Coordinator

Gérard Assayag
IRCAM - STMS Lab
RepMus Head Director

Sound and Music Computing
Aalborg University Copenhagen

2023

Abstract

This thesis was written as the final project of the Master of Science in Sound and Music Computing at Aalborg University Copenhagen.

The presented work has been carried out in the Music Representations team at IRCAM - STMS Lab in Paris, between February and May 2023. During this stay, several applications for assisted cyber-human improvisation have been explored, finally focusing on the co-creative generative software Somax. New features have been researched, implemented and tested, and two artistic residencies with well known improvising musicians has been carried on, leading to performances in international renowned festivals.

This research brought up a number of interesting outcomes that will be explored in a future collaboration, regarding the behaviour of autonomous agents for human-machine improvisation as well as the extension of the artistic residencies in question.

Contents

Chapter 1

Introduction

Historically, the first digital systems for music generation were mainly based on sets of rules coded by a programmer or a composer, to achieve certain results. Examples of this protocol are the systems designed by Iannis Xenakis for his compositions [1], as well as Voyager by George E. Lewis [2]. In particular, the latter might be considered the progenitor of all modern co-creative improvising systems, as it defined a model of rules to achieve high levels of musical interaction [3].

During the years, music generation, or computer-assisted composition, has been extended with a wide number of tools and libraries. Among these, two of the most famous and important were developed at IRCAM [4]: OpenMusic [5] [6], an object-oriented visual programming environment for musical composition based on Common Lisp , and Bach [7], a toolbox for computer-assisted composition for Max/MSP [8], which among other things adapts and integrates several concepts from OpenMusic into Max/MSP. These works eventually helped extending the field of music generation and computer-assisted composition with the notion of machine, or corpus-based, improvisation.

Machine improvisation uses computer algorithms to create improvisation on existing music materials (corpus). This is usually done by sophisticated recombination of musical phrases extracted from existing music, either live or pre-recorded. In order to achieve credible improvisation in a particular style, machine improvisation uses machine learning and pattern matching algorithms to analyze existing musical examples. The resulting patterns are then used to create new variations in the style of the original music, developing a notion of *stylistic reinjection*, a term coined in [9]. This is different from other improvisation methods that use algorithmic composition to generate new music without performing analysis of existing music examples [10].

Style modeling implies building a computational representation of the

musical surface that captures important stylistic features from data. Statistical approaches are used to capture the redundancies in terms of pattern dictionaries or repetitions, which are later recombined to generate new musical data. Style mixing can be realized by analysis of a database containing multiple musical examples in different styles. Machine Improvisation builds upon a long musical tradition of statistical modeling that began with Hiller and Isaacson's *Illiac Suite* for String Quartet (1957) [11] and Xenakis' uses of Markov chains and stochastic processes [12]. Style mixing is possible by blending models derived from several musical sources, with the first style mixing done by Shlomo Dubnov in the piece *NTrope Suite*, using Jensen-Shannon joint source model [13]. Later the use of factor oracle algorithm (a finite state automaton constructed in linear time and space in an incremental fashion) [14] was adopted for music by Gérard Assayag and Shlomo Dubnov [15] and became the basis for several systems that use stylistic reinjection.

1.1 Related works

Over the past two decades, several real-time systems for machine improvisation that, to some extent, interacts or relates to a human musician have been developed. As previously mentioned, the term machine improvisation stems from the fact that the system has some sort of musical understanding of the content it is performing and that the output is being generated on the fly.

Among the more impactful examples of such a system is OMax, a software environment (Creative Agent) which learns in real-time typical features of a musician's style and plays along with him interactively, giving the flavor of a machine co-improvisation. It is based on a research on stylistic modeling carried out by Gérard Assayag and Shlomo Dubnov [15] and a research on improvisation with the computer by G. Assayag, M. Chemillier and G. Bloch (also known as the OMax Brothers) [9]. OMax reinjects in several different ways the musician's material that has gone through a machine-learning stage, allowing a semantics-level representation of the session and a smart recombination and transformation of this material in real-time [16]. As previously mentioned, during this research, the authors coined the term *stylistic reinjection* to express this particular way of interacting of one's own clone.

Several systems for human-machine improvisation have been developed that to some extent stems from OMax. Among these are Somax [17], [18], [19], which adds the concept of reactivity to the OMax model, allowing the system not only to draw its material from an input, but to react to a musician in real-time, hence creating a situation of co-improvisation. Another system, Improtek [20], adds the concept of temporal scenarios and contextual

awareness to the model. Recently, the DYCI2 project [21] [22] was designed with the intention to merge these three systems into a single framework and introducting longer term scenarios.

1.2 REACH: Raising Co-creativity in Cyber-Human Musicianship

The work carried on during this thesis is part of the framework of the European Research Council (ERC) Project REACH: Raising Co-creativity in Cyber-Human Musicianship [23], directed by Gérard Assayag, in the Music Representation Team at IRCAM STMS Lab [24].

The research objective of this inter-disciplinary project is to model and enhance co-creativity as it arises in improvised musical interactions between human and artificial agents in a spectrum of practices spanning from interacting with software agents to mixed reality involving instrumental physicality and embodiment [25]. Such creative interaction strongly involves co-improvisation, as a mixture of more or less predictable events, reactive and planned behaviors, discovery and action phases, states of volition or idleness. Improvisation is thus at the core of this project and indeed a fundamental constituent of co-creative musicianship, as well as a fascinating anthropological lever to human interactions in general.
The outline of the project unfolds as follows:

- Understanding, modeling, implementing music generativity and improvised interaction as a general template for symbiotic interaction between humans and digital systems (cyber-human systems);
- Creating the scientific and technological conditions for mixed reality musical systems' based on the interrelation of creative agents and active control in physical systems;
- Achieving distributed co-creativity through complex temporal adaptation of creative agents in live cyber-human systems, articulated to field experiment in musical social sciences.

The purpose of this thesis is thus to explore parts of this large research program, analyze and summarize the state of the art in co-creative musical interaction, improve features of existing software, experiment and prototype with musicians and creative instruments, in order to adapt to realistic experimental and creative situations.

Hence, this thesis will focus on Somax [26], IRCAM Music Representation team's state of the art in the field of AI interaction for human-machine co-creation. After an introduction given in chapter 1, chapter 2 will present the theoretical model of Somax, summarizing the work done in [18], [19] and [27]. In chapter 3 the work done to improve the real-time beat tracking of the environment will be presented. Chapter 4 will describe the implementation of interaction design strategies used during the artistic residencies, from the study of physical models to generate sound synthesis, to interaction strategies derived from the context of musical co-creativity. Chapter 5 will focus on the artistic research work that has proved fundamental to the development of the current version of Somax and to the success of the artistic residency projects that have accompanied this research. Finally, Chapter 6 will present some suggestions and ideas for future work in the field.

Chapter 2

Somax Concepts

Somax is an interactive system which improvises around a given musical material, aiming to produce a stylistically coherent improvisation while jointly listening to and adapting to live musical input . The system is trained on musical material selected by the user, from which it constructs a corpus that will serve as a basis for the improvisation. Somax can use either audio or MIDI files as its musical material (or a combination of the two), and it is able to listen and adapt to both audio and MIDI input from the external world.

Somax may serve as a co-creative agent in the improvisational process, where the system after some initial tuning is able to continuously listen and adapt to the musician in a self-sufficient manner. Of course, the input doesn't have to come from a live musician; any type of audio and/or MIDI input works, be it from an audio file, score editor, synthesizer, DAW, or another Somax player (agent). Somax also allows detailed parametric controls of its output and can even be controlled as an instrument in its own right. Also, the system isn't necessarily limited to a single agent or a single input source - it is possible to create an entire ensemble of agents where the user can control how the agents listen to various input sources as well as to each other.

The goal of this section is to provide a brief introduction to Somax and provide the reader with the fundamental knowledge about how its interaction model works, which in turn should serve as a basis for making informed choices when tuning and interacting with the system. The content of this chapter summarizes the Somax theoretical and interaction model detailed in [19], [27] and [28].

2.1 The Corpus and Navigation Model

As previously mentioned, the Somax system generates its improvisation material based on an external set of musical material, the corpus. This corpus can be constructed from one or multiple MIDI files, or a single audio file, freely chosen by the user. In contrast to many other generative approaches, the system does not construct a model that eventually is independent of the material that was used to train it. Rather, the model is constructed directly on top of the original data and provides a way to navigate through it in a non-linear manner. One way of seeing this is to consider that some fine-grained aspects of the musical stream are somehow too complex to be modeled, but will be preserved – to a certain extent – when weaving into this musical material.

In order for Somax to build a corpus from given music material, the first step is to segment the musical stream into discrete units or slices, which are vertical (polyphonic) segments of the original file, where the duration of a slice is the distance between two note onsets.[1] Each slice is analyzed and classified with regards to a number of musical features related to its harmony/texture, individual pitches, dynamics, etc., and these features along with the pattern structure they infer over the musical sequence will serve as the main basis for controlling the navigation model. Thus a navigation model is made of a musical memory (basically the corpus), plus a dynamic pattern matching and selection scheme to navigate into memory and reconstruct a generative signal. Slicing is illustrated in Figure 2.1 in the case of MIDI, where the polyphony is broken down in vertical columns with ties between notes, prior to analysis and classification of the content. This representation allows to weave back the polyphonic structure at generation time.

The procedure of mapping memory with patterns is actually repeated for each music feature (harmony, melody etc.) in the analysis, effectively resulting in a multilayer representation where each layer roughly corresponds to one feature, i.e., one layer for harmony, one for pitch, etc.

When a musician interacts with the system, a similar process of segmentation and multilayer analysis and classification is computed in real-time on the input stream, and at each point in time the result of this process is matched to the information in the navigation model, generating activations, or peaks, at certain points in the sequential memory where the input corresponds to

[1] In non-quantized MIDI files it is rare for any two notes that are perceived as simultaneous to be exactly simultaneous. Since one goal of Somax is to be able to maintain and reproduce the original timings within slices as recorded, notes with almost simultaneous onsets will still be grouped together in a single slice but with their internal timing offset preserved.

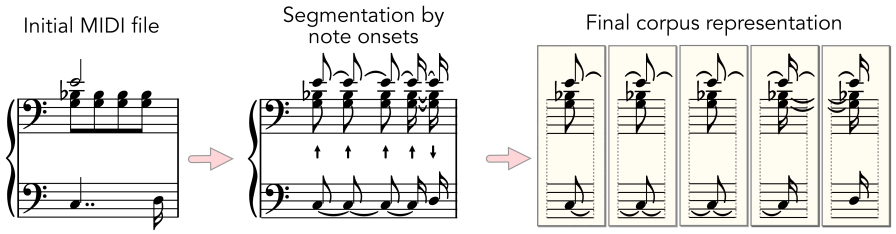


Figure 2.1: Constructing a corpus by segmenting MIDI data into slices.

the model. Each peak corresponds to a point in the memory, so the entire set of peaks can effectively be seen linearly as a one-dimensional curve over the corpus' time axis (see Figures 2.3 and 2.4 below for examples). The peaks in each layer are merged and scaled according to how the system has been tuned, and finally the output slice is selected from the sequential memory based on the distribution of the peaks, typically at a location that is a good match with the overall context at this particular moment.

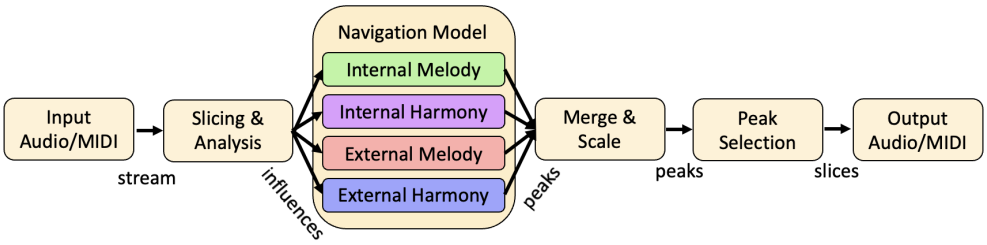


Figure 2.2: An overview of the steps through which the system generates its output at each given point in time.

The generated output of this process is a co-improvisation that recombines existing material in a way that is coherent with the sequential logic and statistical properties of the original material while at the same time adapting in real-time to match the input from the musician. This is because the multilayer peak profiles is shaped not only by the input, but also by the output of the navigation itself, which is called "self-listening". Self-listening conditions its own set of layers, that combine in turn with external listening ones. This process of attempting to balance the internal logic of the corpus with the external logic of the musician often provides a mix of coherency and unexpectedness in a way that convincingly gives the impression of an active agent in the improvisational process.

2.2 Interacting with Somax

When interacting with Somax, there are three main concepts that are important to understand: slices, influences and peaks.

2.2.1 Slices

A slice, as previously mentioned, is a short segment of the corpus and serves as the smallest building block of the output of the system. The slice can be manipulated to some extent (transposed, filtered with regards to voices/channels, stretched etc.) but will always maintain most fundamental properties of the original corpus.

2.2.2 Influences

When Somax listens to a musician, this musical stream is segmented and analyzed with respect to its musical parameters similarly to how the corpus was constructed, but with a slightly different set of methods to be able to operate in real-time. The result of this process are discrete chunks of multilayer data or influences, which the system uses to be able to compare the input to the memory, where the main purpose of the influence is to act as the guide that determines the output of the system. The concept of an influence may initially seem like an implementation detail, but will become increasingly important for more complex configurations with multiple agents and/or multiple input sources. The main takeaway is that the system cannot listen directly to a musical input stream, but will need to translate it into influences, and that the process of tuning the listener can be a very important factor for the quality of the co-improvisation. Thus influences are the main data that circulate between the Somax agents, hence the names 'audioinfluencer' and 'midiinfluencer' for the input modules. Typically, a Somax Player "listens" to a certain amount of cumulated influences (audio inputs, MIDI inputs, other agents' productions) and takes decisions based on these influences.

2.2.3 Peaks

Finally, a peak is, again, a point in the corpus where the input corresponds to the model, or simply a match between an incoming influence and a corresponding slice that would serve as an output candidate. Each peak has a height, corresponding to a probability (or viability) of that particular slice as an output candidate. Unlike influences (which are visible in the interface)

and slices (which are correlated to the audible output of the system), peaks are never interacted with directly, they're only part of the internal state of the system, but perhaps the most vital part. Each peak effectively corresponds to a slice in the corpus that could serve as an output at the current point in time, given the latest influence. Having at each point in time a reasonable number of peaks is thus vital for the quality of the output, since having no peaks means that the output has not taken the musician's influences into account, and on the opposite side, in most cases a large number of peaks indicate that the matching is imprecise.

2.2.4 Putting all together

To put the concept of peaks in context, it is needed to briefly explore in more detail how the system works. While the musician is playing, Somax is at each detected onset segmenting / analysing the input into influences, carrying information about the pitch, harmony, etc. of what the musician currently is playing. This process is carried out by agents of the system called influencers. This information is routed to a player, which handles the entire process of matching and generating output. The influence is routed in multiple layers by the player, as briefly mentioned, where each layer corresponds to one musical dimension (e.g. harmony, melody) of the influence. In each layer, a model of the corpus with respect to the particular layer's musical dimension exists, and upon receiving an influence, the model will look for sequences in the corpus that match the sequence of most recent influences from the input, and, in each of those places, insert new peaks.

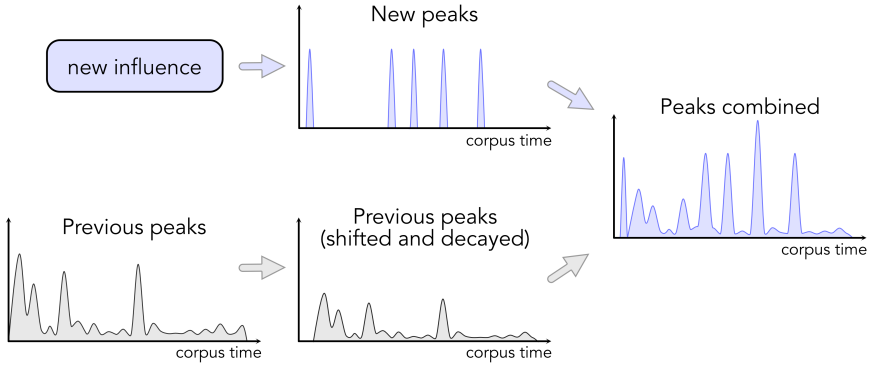


Figure 2.3: The process of shifting and decaying previous peaks in a single layer upon receiving new influences (the process of matching the incoming influence to the corpus has been omitted for clarity).

The system is also simulating a type of short-term memory inside this model by not immediately discarding peaks from previous influences, but rather shifting them along the time axis of the memory and decaying their height corresponding to the amount of time that has passed, followed by merging them with the new incoming set of peaks. This means that sequences continuously matching several consecutive influences will be more highly prioritized over others, as illustrated in Figure 2.3. Another powerful property of this system is that it allows some form of fuzzy pattern recognition since positions that are not perfect matches at time t, can still become good matches at time $t + i$, due to the propagation mechanism. Finally, the peaks from all layers will be merged together into a single set of peaks which the system will use to probabilistically determine which slice is the best output candidate[2]. The result of this multilayer peak merging process is an output that will not just strictly match the harmony and pitch of the influence but rather improvise around the most recent history of influences with regards to the corpus own structure, with both fidelity and agency. In addition, as already said, there are also two layers which listens to the output of the player itself, as feedback layers, that can be used to balance the player's consistency with the input with its continuity in its own performance. The balance between the different layers as well as control over the shift/decay time of old peaks, length of sequences to match in the memories, etc. are all available in the 'somax.player.app' and 'somax.player.ui' user interface.

Another important aspect of the interaction with Somax is its relation to time. According to the user's preference, each player can be assigned to either operate continuously in time as an autonomous agent, maintaining the pulse and exact within-slice timings of the original corpus (while possibly adapting to the tempo and/or phase of the input), or operate reactively, generating output synchronously as requested by the input's events. In the continuous case, this means that the player improvises freely over time while still taking the influences of the musician into account, while in the reactive case, it synchronizes strictly (note-by-note) or loosely (depending on tuning) with the input. Of course, the player is in the latter mode not strictly limited to the input from where it receives its influences, but could be connected to a third source of some sort, for example any type of step-sequencer or other generative approaches, thus giving the user multiple options for controlling the temporal domain of the system.

[2]Actually, in addition to this, there are a number of parameters that scale the height of the peaks individually with regard to a number of other musical parameters of choice, but this is thoroughly documented in the help files and tutorials in Max and will not be discussed here

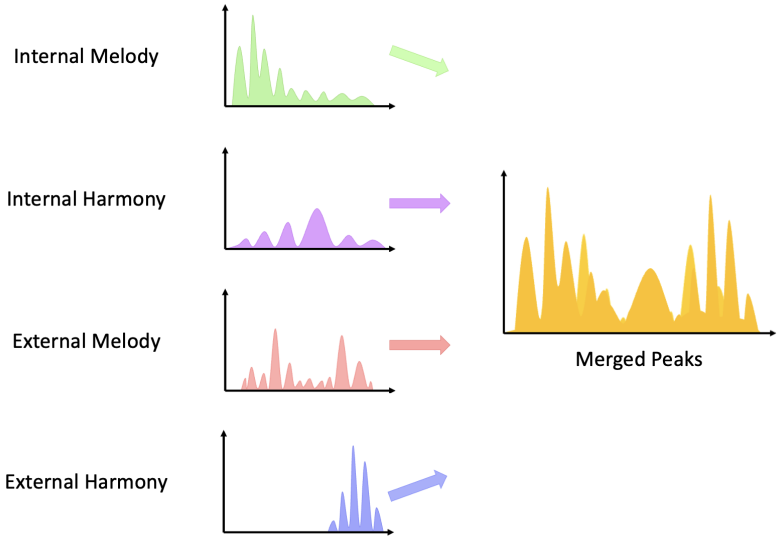


Figure 2.4: The four layers of peaks corresponding to different musical dimensions such as internal melody, internal harmony, external melody and external harmony, being merged into one set of peaks before the final scaling and peak selection. Here, all four layers are weighted equally, but it is possible to balance the contribution from each of the layers.

2.3 Somax User Interface

The user interface of Somax, which was discussed briefly in [19], is implemented in the Max [29] programming language. The user interface was originally designed as a thin client, where all of the computation is handled on the Python server (apart from the real-time signal processing required for audio signals used as influences). Being Max a visual programming language where the default means of programming is by connecting objects using patch cords, in most cases, the readability of a Max program is determined by how easy it is to follow the cords throughout the program. The Somax user interface was originally designed with this in mind to promote readability on both micro and macro levels of the program, but is from version 2.3 using wireless communication (`send` and `receive`) between objects on a macro level. While this approach to some extent obscures the readability (or at least the global signal flow) of the system, the benefits are manifold. First of all, the architecture becomes easier for the user to extend - adding new players can be done with a single keypress - and objects can dynamically select which other objects to interact with without having to modify the architecture. This new

architecture and its implications is now presented in [28] and [30].

Another purpose of this redesign is to make the system integrable into Ableton Live. Somax can as a system be described as a function that reads one or multiple audio and/or MIDI streams and outputs one or multiple MIDI streams. For compatibility with Live, this has to be split into several smaller objects based on Live's syntax of instruments (function that reads one MIDI stream and outputs one audio stream), audio effects (function that reads one audio stream and outputs one audio stream) and MIDI effects (function that reads one MIDI stream and outputs one MIDI stream). Using a wireless architecture, this goal is possible to achieve for Somax.

To accommodate these changes, the Python back-end has been updated to drastically increase the performance when using multiple players.

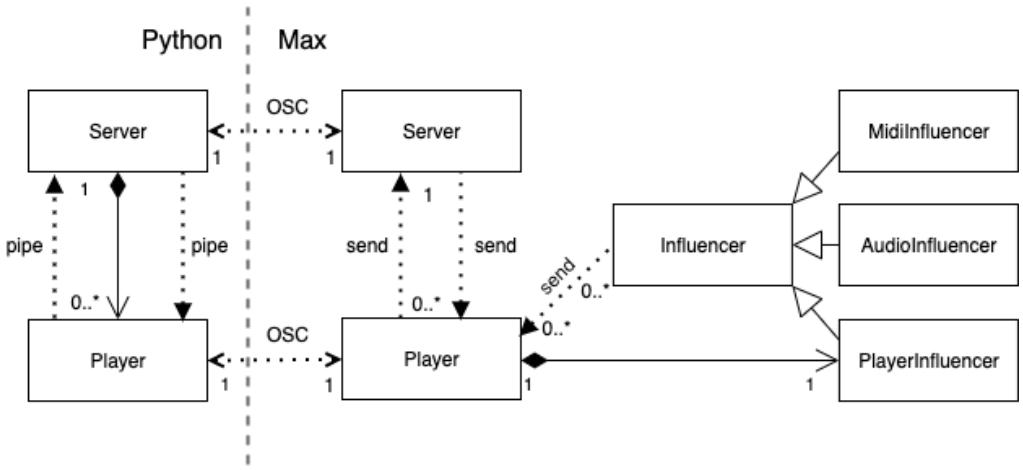


Figure 2.5: Interaction model for the wireless architecture. Dotted arrow lines denote some sort of "wireless" communication between objects while filled arrow lines denote their traditional UML relations (composition, generalization) and corresponding cardinality.

A simplified diagram over the entire wireless system can be seen in figure 2.5. Here we see that the only objects that do not have corresponding objects in the Python back-end are the influencers. Each influencer is given a name by the user, which the system ensures is unique, and this name will serve as the address on which the influencer sends its influences to players.

The new Somax2.5 has been redesigned both as a Max application and a Max library. Almost every object in the package has a 'core' version and an '.app' version (i.e., 'somax.player' and 'somax.player.app'). As their names say, core objects are pure Max abstractions, intended for users that want to use Somax in a fully Max-like programming style. On the other hand, .app

objects are abstractions providing a user interface to control the parameters, as well as other utility tools to immediately use the abstractions. They are really wrappers around the core objects (that are still contained inside any .app object) that provide the users instant access to a nice interaction, while maintaining the modularity of the core objects. Only .app objects appear in the 'somax2.maxpat' application that most people will use, unless they are skilled Max programmers and wish to build their own custom application patch. Some of the objects have also a '.ui' version (i.e., 'somax.player.ui'), which is a compact user interface version of the core object, and that could be used as an alternative to this one. Note that the .ui objects doesn't have the same utilities as the .app objects, and therefore they cannot be used as ready-to-play objects, but rather they are intended as visual feedback alternatives to the core objects. Since .ui objects are included in .app objects they will also serve in the following to show and explain some of the .app user interface details. In Figure 2.6 are presented the basic objects of the Somax application. These are:

- the Server;
- the Player;
- the Audio Influencer;
- the MIDI Influencer.

This is of course just one possible configuration of the Somax objects, as a big advantage of this application is that each object communicates with each other in a wireless way, without the need of patch chords. In this way it is possible to adapt the patches at one's need, and build personal configurations.

Figure 2.6: The 'somax2.maxpat' included in the distribution gives access to all the objects in the Somax application: server, player, audio influencer and MIDI influencer.

Chapter 3

Real Time Beat Tracking

Beat tracking is the identification of a regular pulse in a piece of music, similar to the informal task of tapping ones foot 'in time' to a piece [31]. To a human, even a non-musician, this is often a trivial task that can be achieved with little conscious effort. However, the automation of this process in computer systems has proved difficult to solve comprehensively [32]. As Goto [33] noted, "although in the brains of performers music is temporally organized according to its hierarchical beat structure, this structure is not explicitly expressed in music". Developing a real time beat tracker brings up a further challenge, since we have no access to future information. Therefore, a causal approach is needed, where past and present information alone are used to inform predictions of future beat locations.

3.1 State of the art research

Over the years, a wide number of different approaches tried to solve the problem of beat tracking. Among these are some of the most important:

- Goto [33], presented a system that is able to recognise beats at multiple metrical levels. The system, assuming a steady tempo and a time signature of 4/4, examines onset times, chord changes and drum patterns in the input signal;
- Ellis [34], proposed an algorithm based on dynamic programming, where an onset strength envelope is extracted by re-sampling the input audio to 8kHz and extracting short-time Fourier transform frames. These are converted to an "approximate auditory" representation with 40 bands on the Mel scale – a perceptual frequency scale. The first-order difference in each band is taken and the positive differences summed across

all bands and the final function smoothed. From this function a global tempo estimate for the signal is calculated from the weighted output of an auto-correlation function. This global tempo estimate is then used with the onset strength envelope to create a transition cost function which in turn is used in the calculation of a recursive function with peaks at likely beat locations [31];

- Davies and Plumbley [35], introduced a system that calculates an onset detection function based upon the complex spectral difference between adjacent Fourier transform frames. The auto-correlation of the onset detection function is then calculated and passed through a weighted comb filterbank. The output of this comb filterbank has peaks at lags that match the periodicities in the auto-correlation function and is used to estimate the beat period (or time between beats). Analysis of the most recent single beat period of the onset detection function is then performed to extract the beat phase.

- Finally, the models of Large and Kolen [36] and Toiviainen [37] are similar, and address the issue of rhythm perception through a method of adaptive oscillators (as well as the various evolutions of Large's model; see [38]; [39]; [40]; [41]). Two key parameters describe these oscillators: their phase and period, which can adapt to allow the oscillator to remain synchronized with the music input (represented as discrete events). In the absence of stimuli, the oscillator continues to produce a pulse at the current tempo. These two models differ mainly in the mechanism chosen to adapt their internal parameters. When a new event arrives (a note, represented by its attack time), Large's model instantly adapts its parameters (phase and period) according to a gradient descent method; Toivianen's model, largely inspired by Large's, introduces an adaptation function that delays the modifications made to the internal parameters, which allows to take into account the duration of the notes. A shortcoming of this type of model is that it is not able to find the right initial values of the phase and especially of the period. It should also be noted that these models have been used and tested in improvisations from the beginning (see [36]; [38]; [37]).

- Also worth mentioning, IRCAM has already implemented a plugin for real time beat detection, IrcamBeat [42]. Created by Geoffroy Peeters and developed by Geoffroy Peeters and Frédéric Cornu in the IRCAM Analysis-Synthesis team at STMS lab, the VAMP plugin created and developed by Pierre Guillot and Matthew Harris, offer a set of analyses

from the AudioSculpt 3 application, and could be loaded in the Partiels application [43].

3.2 Implementing a new beat tracker in Somax

As of today, Somax has an integrated beat tracker, implemented by Laurent Bonnasse-Gahot [44]. This is based on the two different models of Large and Kolen [36] and Toiviainen [37] but its imlementation goes back to 2010 and was specifically designed for Omax [15]. This implementation, in the particular context of Somax, was identified as a weak point by many members of the Music Representations team, so much so that it was not actually used in testing and performance by any of them. In fact, Omax has a different architecture, being implemented fully in Max/MSP, while Somax has a Python back-end and a Max front-end [45], so the way the algorithm was implemented in Omax doesn't really fit the internal structure of Somax. Moreover in this new application is very important to being able to synchronize agents (Somax players) with an external audio or MIDI influence, carrying some tempo information, or even within agents, sharing tempo information and syncing between them.

After a thorough research on existing real time beat tracking algorithm that could potentially be included in the current version of Somax, the choice fell on the btrack$\sim$ implementation by Stark [31]. The model draws on two existing systems: the tempo induction of the Davies and Plumbley [35] method and the dynamic programming approach of Ellis [34]. In this section, a brief explanation of the algorithm is given; for a full derivation see [46] and [31].

3.2.1 Input Feature

The input feature for this beat tracking system is the complex spectral difference onset detection function (DF) [47]; a continuous mid-level representation of an audio signal which exhibits peaks at likely note onset locations. The onset detection function $\Gamma(m)$ at sample m is calculated by measuring the Euclidean distance between an observed spectral frame $X_k(m)$, and a predicted spectral frame $\hat{X}_k(m)$ for all bins k:

$$\Gamma(m) = \sum_{k=1}^{K} |X_k(m)| - |\hat{X}_k(m)| \tag{3.1}$$

3.2.2 Beat Prediction

Stark's model for beat tracking assumes that the sequence of beats, γ_b, will correspond to a set of approximately periodic peaks in the onset detection function. Following the dynamic programming approach of Ellis [34], at the core of this method is the generation of a recursive cumulative score function, $C^*(m)$, which represents the best possible score of all possible beat sequences ending at the point m. This cumulative score value at m is calculated as the weighted sum of the current DF value $\Gamma(m)$ and the value of C^* at the most likely previous beat location:

$$C^*(m) = (1-\alpha)\Gamma(m) + \alpha \max_v (W_1(v)C(m+v)) \tag{3.2}$$

Specifically, the algorithm searches for the most likely previous beat over the interval (into the past) $[m - 2\tau_b, m - \tau_b/2]$ where m is the current input feature sample and τ_b is the beat period (the method for determining τ_b is given in a later section). A log-Gaussian transition weighting W_1 favours the time exactly τ_b samples in the past:

$$W_1(v) = exp\left(\frac{-(\eta \log(-v/\tau_b))^2}{2}\right) \tag{3.3}$$

where $v = -2\tau_b, ..., -\tau_b/2$ and η is the tightness of the transition weighting. The cumulative score C^* is updated with every new detection function sample $\Gamma(m)$ and its recursive calculation allows to carry some periodic *momentum* even in the presence of silence. This feature is the one allowing the algorithm to make predictions of future beat locations without observing the entire signal. Each predicted beat γ_{b+1} is made at a fixed point in time m_0 once the current beat γ_b has elapsed, $m_0 = \gamma_b + \tau_b/2$. From here, the future cumulative score for one beat into the future is generated and the next beat is predicted as:

$$\gamma_{b+1} = m_0 + \arg\max_v (C^*(m_0+v)W_2(v)) \tag{3.4}$$

where $v = 1, ..., \tau_b$ specifies the future one-beat window and $W(2)$ is a Gaussian weighting centred on the most likely beat location $(m_0 + \tau_b/2)$:

$$W_2(v) = exp\left(\frac{-(v-\tau_b/2)^2}{2(\tau_b/2)^2}\right) \tag{3.5}$$

An example of the beat prediction process in shown in Figure 3.1.

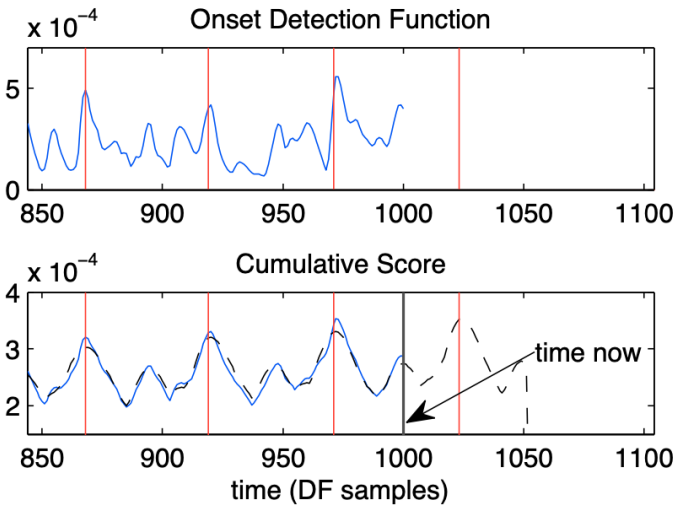


Figure 3.1: Top: Onset Detection with predicted beat locations. Bottom: Cumulative score (solid line) with future cumulative score (dotted line). Current time is shown as the bold grey vertical line. Taken from [46].

3.2.3 Tempo Induction

To be able to track beats in music that varies in speed in real time, a regular update of the tempo estimate is needed. In line with the beat prediction, the tempo is re-estimated once each new predicted beat has elapsed. To estimate the value of τ_b, the beat period estimate at the bth beat γ_b used in the beat prediction section, Stark adopted a method based on Davies and Plumbley method [35]. This can be summarised in five steps:

- a six second analysis frame (up to m_0) is extracted from the onset detection function $\Gamma(m)$;
- the peaks in $\Gamma(m)$ are preserved by applying an adaptive moving mean threshold to leave a modified detection function $\tilde{\Gamma}(m)$;
- the autocorrelation function of $\tilde{\Gamma}(m)$ is taken;
- this autocorrelation function is passed through a shift-invariant comb filterbank weighted by a tempo preference curve;
- the beat period is found as in the index of the maximum value of the comb filterbank output, $R(l)$.

An example of the comb filterbank output is shown in the top plot of Figure 3.2. For a complete derivation of R(l) see [35]. To minimise some common errors, like tapping tempo at different metrical levels [48], or switching between metrical levels [35], Stark restricted the range of possible tempi to a single tempo octave from $t_{min} = 80$ beats per minute (bpm) to $t_{max} = 160$ bpm. The output of the comb filterbank $R(l)$ is mapped from the lag domain to the tempo domain between t_{min} and t_{max} to give $R_b(t)$ by:

$$R_b(t - t_{min}) = R(|60/(f_r \times t)|) \tag{3.6}$$

where $t = t_{min}, ..., tmax$ and f_r is the temporal resolution of the onset detection function in seconds. Plots of $R(l)$ and $R_b(b)$ are shown in Figure 3.2.

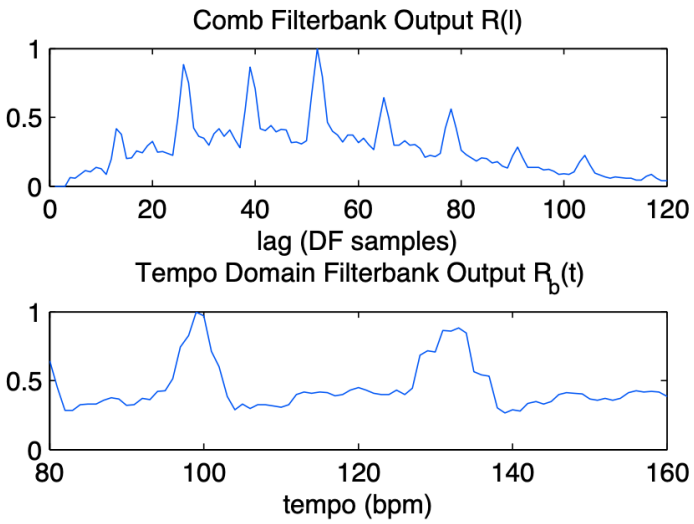


Figure 3.2: Top: Comb Filterbank Output R(l). Bottom: R(l) mapped into the tempo domain to give Rb(t). Taken from [46].

Finally, since it has been assumed in previous works [49] that tempo is a slowly varying process, Stark introduced a transition matrix $A(t_i, t_j)$ to favour changes in tempo (from the current tempo t_i to a new tempo t_j) that are small, so that the new tempo is close to the current tempo. In this matrix, each column contains a Gaussian of fixed standard deviation σ, so that each Gaussian is wide enough to capture small changes in tempo but narrow enough to favour the hypothesis that tempo is a slowly changing process.

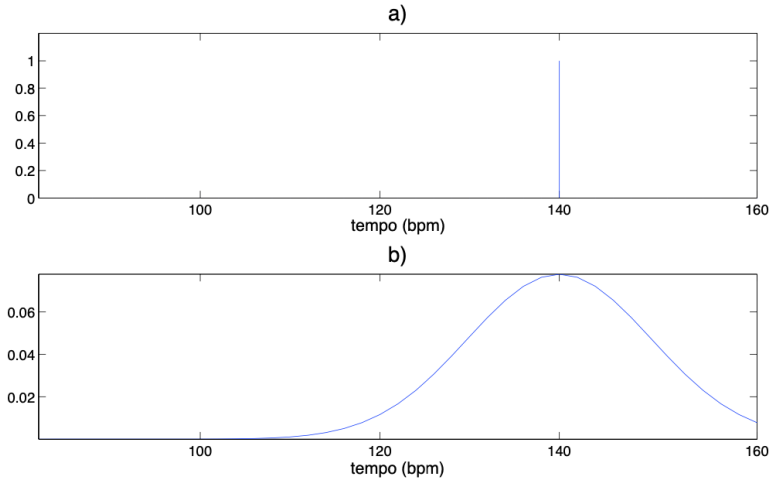


Figure 3.3: Top: Tempo initialisation of an artificial likelihood for the previous beat Δ_{b-1} at 140 bpm. Bottom: resulting tempo probability distribution after operations with the matrix. Taken from [46].

At each step, the previous tempo likelihood Δ_{b-1} is stored and multiplied by the transition matrix A; the maximum value from each column is taken to create a tempo probability distribution θ_b. The tempo likelihood for the current iteration Δ_b is then calculated as $\Delta_b(t_j) = R_b(t_j)\theta_b(t_j)$. The current tempo is then found as the index of the maximum value of Δ_b and converted from bpm to the beat period τ_b, as shown in Figure 3.3.

3.3 Evaluation

Evavaluation of Stark's beat tracking model has been carried on an existing annotated database [50] comprised of 222 audio files each approximately 60 seconds in length. Here each file was accompanied by a sequence of beat annotations, recorded as beat times in seconds and created by a human listener tapping in time to the piece [31].

To evaluate the proposed beat tracker, compared to the ones of Davies and Plumbley [35] and of Klapuri [49], Stark utilised two different evaluation measures, both developed by Hainsworth [50] and Klapuri [49]:

- AML_c (Allowed Metrical Levels, with Continuity required), that requires that there be some continuity in the beats output by the beat tracker;

- AML_t (Allowed Metrical Levels, total number of correct beats), which is a measure of the total number of correct beats, regardless of continuity. For more information on both measures see [31], 3.1.6.

The results of this evaluation are collected in Table 3.1. As can be seen, the proposed btrack∼ algorithm is comparable in efficiency to the existing state of the art algorithms.

	AML_c(%)	AML_t(%)
DP	70.5	79.1
KL_c	65.7	76.5
BT	**66.0**	**74.9**

Table 3.1: Results of the evaluation of btrack∼ (BT) [31] compared to Davies and Plumbley (DP) [35] and casual Klapuri (KL_c) [49], with the two proposed evaluation measures AML_c (Allowed Metrical Levels, with Continuity required) and AML_t (Allowed Metrical Levels, total number of correct beats).

3.4 Max/MSP external

Stark's algorithm, implemented in C++, is available under GNU License [51] as a public repository[1]. Starting from this code, a Max/MSP external, shown in Figure 3.4, has been compiled in XCode [52], adapting the algorithm to the latest version of Max. The btrack∼.mxo external has then been inserted in an experimental version of Somax for testing with live audio inputs. Having a Max/MSP external is very convenient in term of adding it to the overall application. In this way, a routing of the tempo of both the audio and MIDI influencers can be sent to the players in Somax; moreover, each player could send its own tempo information, creating tempo feedback within the players themselves. It is hoped that a solid beat tracker in the Max/MSP environment, based on a C++ implementation of the algorithm, could increment the tempo performances of the system, as well as provide its maintenance in future developments.

3.4.1 Gen∼ development

As a proof of concept and to deepen the state of the art research in the Max/MSP environment, additional experimentation has been carried on on

[1] `https://github.com/adamstark/BTrack`

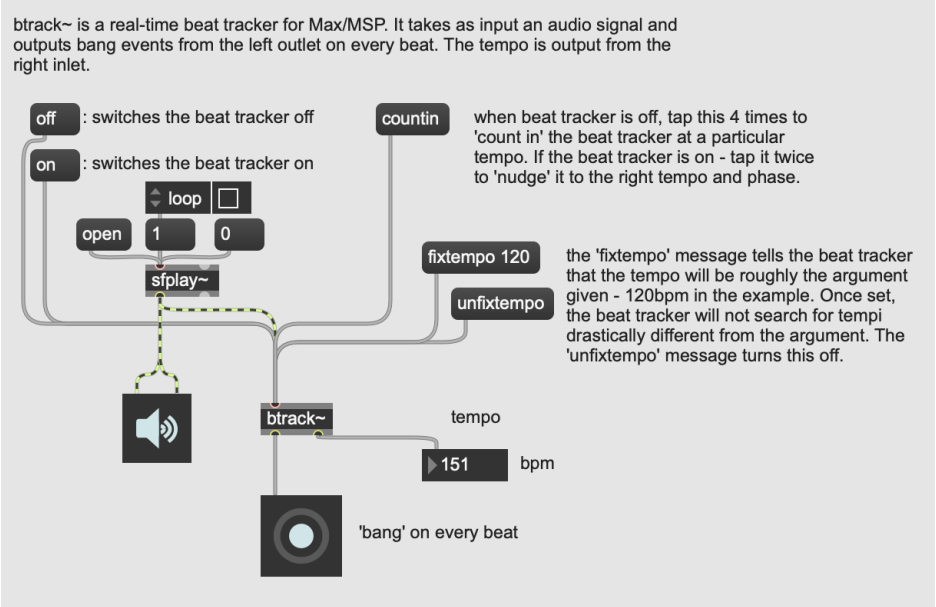


Figure 3.4: btrack∼ Max patch developed by Stark, Davies and Plumbley following Stark's research in [31], compiled as a Max/MSP external for the latest architectures.

the signal analysis side. In particular, a wide range of audio descriptors, as described by Stark in [31], have been implemented in the gen∼ environment, under the supervision of Graham Wakefield, gen∼ developer and author of [53].

Gen∼ is part of the Max/MSP environment, but contrary to this, operates at sample rate, providing the possibility to develop complex DSP algorithms. This process is already part of a wide range of products, as the Shaper Synthesizer [54], OWL Modular Eurorack [55] or MOD Duo [56]. Unfortunately, however, gen∼ is not well documented, but thanks to the aforementioned support of Graham Wakefield and after a considerable number of attempts, I have been able to implement these descriptors, with particular emphasis on the complex spectral difference descriptor, defined in [47]. The results can be seen in Figure B.2 and B.1 in Appendix.

3.5 Extensions to the algorithm

Following the work on swarm robots and their temporal interaction, as described in [57], the previous algorithm has been extended with the following features:

- Regular beat-tracking mode that does everything described thusfar, but does not handle boundary tempo octave conditions;
- A count-in mode that uses the first two onsets to define the initial tempo estimate before switching back into regular beat-tracking mode;
- Tempo-locked mode, which, after an initial tempo has been established, just plays evenly-spaced beats that are not influenced by audio input. This is accomplished by setting the cumulative score α parameter, as described in Equation 3.2, to unity.

An example of this is presented in Figure B.3 in Appendix.

Chapter 4

Interaction Design

This chapter will focus on the music technology techniques used for the development of Somax in the context of the artistic and research residencies that made up this research project. In the same way, the interaction strategies that have led to the concrete realization of these works of artistic co-creativity will be described. In particular, for the digital synthesis of sound, in relation to the project A.I. Komboï, which will be described in the next chapter, studies on physical modeling have made it possible to reach a satisfactory level of sound likelihood with the harpsichord, although maintaining the unique characteristics relating to digital synthesis, thus being consistent with the research work in progress. All the interaction strategies explored during the residency and the use of Somax as the main environment for musical co-interaction will also be described.

4.1 Digital Harpsichord

In order to be coherent with the original idea of Komboï, written for percussion and harpsichord (an analysis of it will be given in the next Chapter), while the percussion part was played with the exact set described by Xenakis in the score, the harpsichord part has been synthesized. A digital harpsichord sound has been designed using physical modelling technique on a digital hardware synthesizer, an Arturia MiniFreak [58].

4.1.1 Digital Waveguides

Proposed by Smith in [59], digital waveguide methods follow a different path from the ones based on numerical integration of the wave equation. In this implementation, the wave equation is first solved in a general way to obtain

travelling waves in the medium. In the lossless case, a travelling wave between two points in the medium can be implemented using nothing but a delay line. The advantage of this is that, operating with Linear Time-Invariant systems (LTI), most of the simulation still consists of delay lines, keeping the computational costs very low. The following explanations are taken from [59], where further explorations could be found.

4.1.2 The Ideal Vibrating String

Fully derived in [60] and in most textbooks on acoustic, the wave equation for an ideal (lossless, linear and flexible) string is given by:

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2} \tag{4.1}$$

where $y(t, x)$ represents the vertical string displacement over time t and horizontal space x, and c is the speed of the wave propagating in the string.

Following Bilbao's implementation, the former equation could also be written as:

$$Ky'' = \epsilon \ddot{y} \tag{4.2}$$

where $K \triangleq$ string tension, $\epsilon \triangleq$ linear mass density, $y \triangleq y(t, x)$ string displacement, $\ddot{y} \triangleq \frac{\partial^2 y}{\partial t^2}$ and $y'' \triangleq \frac{\partial^2 y}{\partial x^2}$. In the following form, the wave equation can be interpreted as a statement of Netwon's second law $\vec{F} = m\vec{a}$ on a microscopic scale. Considering transverse vibration of the string, the force is given by the string tension times the curvature of the string (Ky'') and it is balanced by the mass times the transverse acceleration ($\epsilon \ddot{y}$). For the physical modeling of musical instruments, this is the most common equation and can be applied to any perfect elastic medium which is displaced along one dimension, describing transverse vibration in an ideal string, longitudinal vibration in an ideal bar or pressure in an acoustic tube [61] [62]. For example, the air column of a clarinet or organ pipe can be modeled using the one-dimensional wave equation, substituting air pressure deviation for string displacement, and longitudinal volume velocity of air in the bore for transverse velocity on the string. We refer to this general class of media as *one-dimensional waveguides*, though extensions to more dimensions are mathematically possible [59].

4.1.3 Travelling-Wave Solution

The wave equation can be solved by any string shape which travels to the left or right with speed $c = \sqrt{K/\epsilon}$. If we denote right-going travelling waves as $y_r(x - ct)$ and left-going travelling waves $y_l(x + ct)$, then the general solution for the lossless one-dimensional wave equation can be expressed as:

$$y(x,t) = y_r(x - ct) + y_l(x + ct) \tag{4.3}$$

while an example of the appearance of the travelling wave components is shown in Figure 4.1.

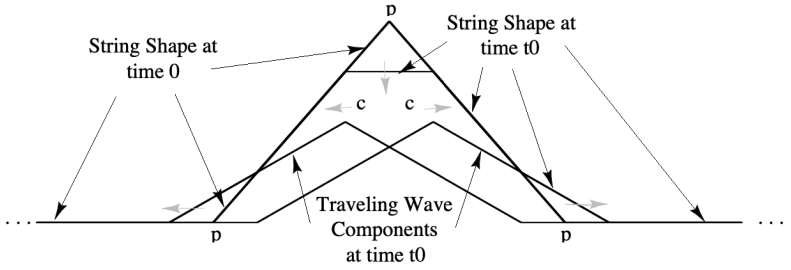


Figure 4.1: An example of the appearance of the travelling wave components shortly after plucking an infinitely long string at three points as shown in [59]. The plucking points are labeled by "p" and are plucked simultaneously, producing an initial triangular displacement. This initial displacement is modeled as the sum of two identical triangular pulses which at $t = 0$ are exactly on top of each other but then begin to separate at $t = t0$.

4.1.4 Sampling the Travelling Waves

To carry the travelling-wave into the digital domain, it is necessary to sample the travelling-wave amplitudes at intervals of T seconds, corresponding to a sampling rate $f_s = 1/T$ (samples per second). The spatial sampling index X is defined as $X = cT$ (meters), representing the distance sound propagates in one temporal sampling interval T. In air, where c is the speed of sound ($331m/s$), for a $f_s = 44100Hz$, $X = 331/44100 = 7.5mm$ of spatial sampling interval.

Sampling is carried out by the following change of variables:

$$\begin{aligned} x &\to x_m = mX \\ t &\to t_n = nT \end{aligned} \tag{4.4}$$

Substituting them into equation 4.3 gives:

$$\begin{aligned} y(t_n, x_m) &= y_r(t_n - x_m/c) + y_l(t_n + x_m/c) \\ &= y_r(nT - mX/c) + y_l(nT + mX/c) \\ &= y_r[(n-m)T] + y_l[(n+m)T] \end{aligned} \tag{4.5}$$

Since T multiplies all the arguments we can suppress it by defining

$$y^+ \triangleq y_r(nT) \qquad\qquad y^- \triangleq y_l(nT) \tag{4.6}$$

Now, the left- and right-going travelling waves could be summed into:

$$y(t_n, x_m) = y^+(n-m) + y^-(n+m) \tag{4.7}$$

Any ideal one-dimensional waveguide can be simulated in this way. Here, the term $y_r[(n-m)T] = y^+(n-m)$ can be thought as the output of an m-sample delay line whose input is $y^+(n)$, resulting in the right-going component in the upper rail of Figure 4.2. Similarly, the term $y_l[(n+m)T] = y^-(n+m)$ can be thought as the input of an m-sample delay line whose output is $y^-(n)$, resulting in the left-going component in the lower rail in the same Figure.

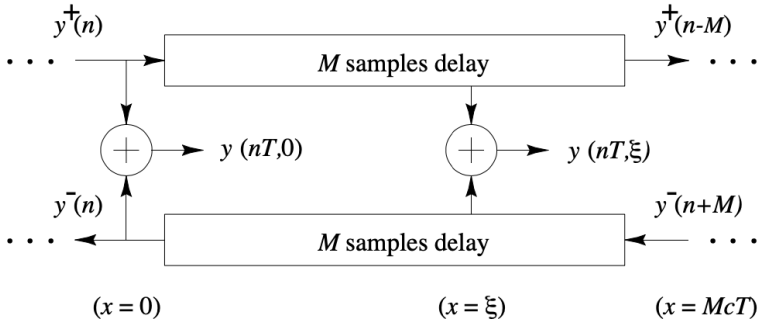


Figure 4.2: Simplified picture of ideal digital waveguide as shown in [59]. The figure emphasizes that an ideal and lossless waveguide is simulated by a bidirectional delay line.

4.1.5 Rigid Terminations

A rigid termination is the simplest case of a string termination. It imposes the constraint that the string cannot move at all at the termination. If we terminate a length L ideal string at $x = 0$ and $x = L$ we have the so-called *boundary conditions*.

$$y(t,0) = 0 \qquad\qquad y(t,L) = 0 \tag{4.8}$$

Since $y(t,0) = y_r(t) + y_l(t) = y^+(t/T) + y^+(t/T)$ and $y(t,L) = y_r(t - L/c) + y_l(t + L/c)$, the constraints of the sampled travelling waves becomes:

$$\begin{aligned} y^+(n) &= -y^-(n) \\ y^-(n + N/2) &= -y^+(n - N/2) \end{aligned} \tag{4.9}$$

where $N \triangleq 2L/X$ is the time in samples to propagate from one end of the string to the other and back (total string loop delay).

The ideal plucked string with rigid terminations for the bridge and the nut, implemented in this project, is then defined as a string with initial displacement and zero initial velocity. An example of an initial pluck excitation in this digital waveguide string model is shown in Figure 4.3

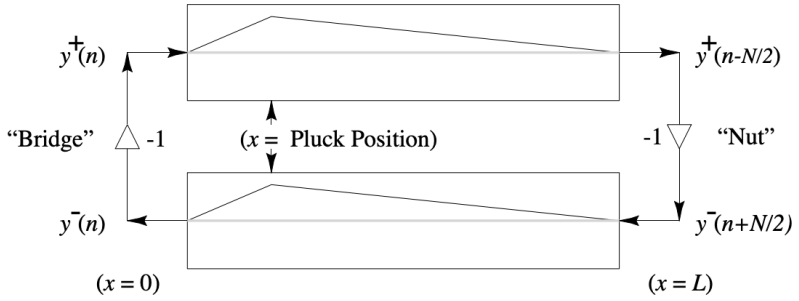


Figure 4.3: Initial conditions for the ideal plucked string as shown in [59].

4.1.6 Implementation

To design the harpsichord sound, according to the previously described physical model techniques, an Arturia MiniFreak keyboard has been chosen. The choice fell on this particular keyboard for multiple reasons. First of all, the MiniFreak is a polyphonic hybrid keyboard that combines dual digital sound

engines with analog filters, modulations, effects and two oscillators. The two oscilallators, providing different sound engines, make it possible to generate plucked string sounds through a special case of digital waveguide synthesis, as described above, the Karplus-Strong algorithm, which will described in the next paragraph. Moreover, the MiniFreak comes also with a VST version of it, the MiniFreak V (see Figure 4.4), that enables advanced preset management and exact mirroring with the physical keyboard. Lastly, the keyboard has embedded MIDI in and out, essential for sending MIDI messages from the Max/MSP patch used in the performance.

Figure 4.4: Arturia MiniFreak V patch for designing of Harpsichord sound, generated through physical modelling.

Karplus-Strong

Karplus–Strong string synthesis is a method of physical modelling synthesis that loops a short waveform through a filtered delay line to simulate the sound of a hammered or plucked string or some types of percussion [63] [64]. At first glance, this technique can be viewed as subtractive synthesis based on a feedback loop similar to that of a comb filter for z-transform analysis. However, it can also be viewed as the simplest class of wavetable-modification algorithms now known as digital waveguide synthesis, because the delay line

acts to store one period of the signal. Alexander Strong invented the algorithm, and Kevin Karplus did the first analysis of how it worked. Together they developed software and hardware implementations of the algorithm, including a custom VLSI chip. They first named the algorithm "Digitar" synthesis, as a acronym for "digital guitar".

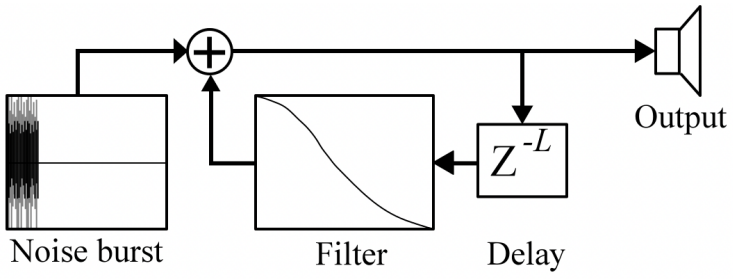


Figure 4.5: Scheme of Karplus-Strong algorithm.

As shown in Figure 4.5, the Karplus-Strong algorithm works in the following way:

- A short excitation waveform (of length L samples) is generated. In the original algorithm, this was a burst of white noise, but it can also include any wideband signal, such as a rapid sine wave chirp or frequency sweep, or a single cycle of a sawtooth wave or square wave.

- This excitation is output and simultaneously fed back into a delay line L samples long.

- The output of the delay line is fed through a filter. The gain of the filter must be less than 1 at all frequencies, to maintain a stable positive feedback loop. The filter can be a first-order lowpass filter (as pictured). In the original algorithm, the filter consisted of averaging two adjacent samples, a particularly simple filter that can be implemented without a multiplier, requiring only shift and add operations. The filter characteristics are crucial in determining the harmonic structure of the decaying tone.

- The filtered output is simultaneously mixed into the output and fed back into the delay line.

4.2 Designing Co-Interaction

Based on the theoretical model previously described in Chapter 2, the basic workflow of interacting with the Somax system, as shown in Figure 4.6, unfolds as following:

- Somax generates its improvisation material based on an external set of musical material, the corpus. This corpus can be constructed from one audio or MIDI file freely chosen by the user, thanks to the dedicated corpusbuilder objects. The constructed corpus is then stored in a bigger database called corpora, accessible by the Player from a corpuspath folder. So, the corpus is the actual musical material loaded into a Player.

- The Audio and MIDI Influencers listen to a continuous stream of audio or MIDI input data (from any type of source, including live musicians) and segments it temporally, where each slice is analyzed with respect to onset, pitch and chroma, which then is sent to the Player. Thus, the influences are the triggers for the co-improvisational behaviour of the Player.

- The Player is the main agent of Somax. It listens to the influencers and, based on that, it recombines the content of the corpus, generating some output.

- The Server is the core of Somax, handling all the scheduling and communication with the background Python app and all instances of somax.player through OSC.

- The 'somax.player' does not play back, it only provides a list of sequential events. Playback is handled independently via the somax.audio/midi renderer objects (or by implementing ones own sequencer/playback patch). However, if using the 'somax.player.app' object, rendering and playback of the output is handled here, as this is a ready-to-play application object.

4.2.1 Interaction Parameters

The interaction parameters described in this section refer to the latest add-ons and advances made in the design and development of Somax in the last years. For a full dissertation of those, see [28].

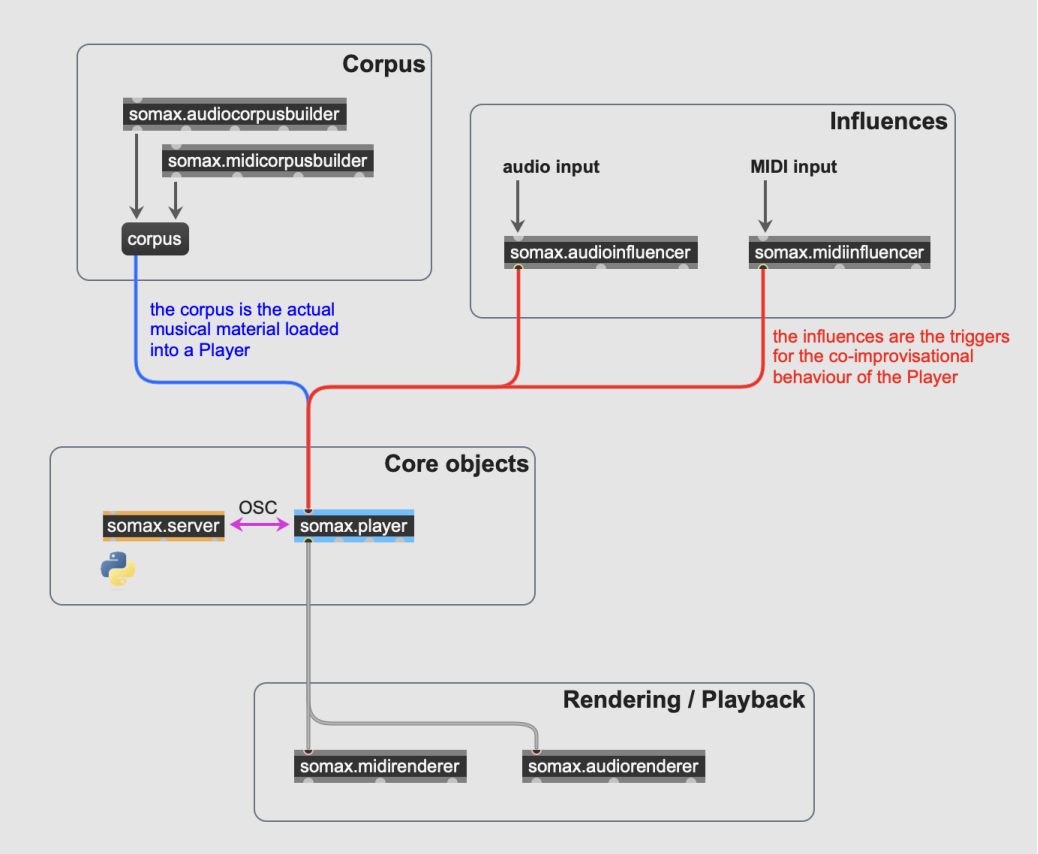


Figure 4.6: Basic Workflow of the Somax system.

While the 'somax.player' has a wide set of parameters, fully documented in the parameters tab of the 'somax.player.app.maxhelp', the 'somax.player.ui' gives access to a selection of main parameters, as shown in Figure 4.7, and described in the following section.

Playing Mode

Controls the player's mode:

- Reactive: Output will be triggered whenever the player receives input from an influencer (note-by-note);
- Continuous: Output will be triggered continuously regardless of input.

Figure 4.7: The user interface of 'somax.player.ui' shows a set of main parameters to control the interaction.

Timeout

In reactive mode, this option controls whether the player should continue playing if no new trigger has arrived by the time the player has finished playing its current event. Setting this to a non-zero value will make the player continue for that number of seconds. It's also possible to disable this to make the player continue endlessly.

Continuity

Continuity controls the order of the current state index of the player's output:

- Continuity > 1 will prioritize continuation (and result in fewer jumps);
- Continuity $= 1.00$ will result in no alterations (no bias introduced by this parameter);
- Continuity < 1 will prioritizing jumping.

Quality / Sparse

The Quality Threshold sets a minimum score required for a match to qualify as output. When combined with Sparse, this will ensure that no events are played unless they are considered good matches.

If Sparse is On:

- quality 0.0 plays any found match;
- quality 1.0 will almost never play;
- in-between values will act as a threshold, to select matches above this threshold and play them.

If Sparse is Off, it will replace the voids (no-play) by a default event, generally the next event, or a jump resulting from self-influence.

Cut

In reactive mode, output is generated each time a new trigger (onset) arrives. If the player is in the middle of playing an event when a trigger arrives, cut controls whether the currently played event should be interrupted or not:

- Allowed: Interrupt the current event and trigger the new event immediately when the new trigger arrives.
- Not Allowed: Delay the trigger so that the new event starts playing once the current event has been completed.

Probability

It will condition each generated output with a probability, so that it may or may not play the event. This parameter is inactive when set to 1.0 (off), but any value lower than 1.0 will result in less than 100% of the events being played. For example, when set to 0.2, only 20% of the generated events will be played

Internal and External Influences

Control the balance between different internal and external type of influences (layers). As it can be seen in Figure 4.8, the four colors (green, purple, red, blue) correspond to the four different layers introduced in Section 2.2.4:

- Green (internal melody): The feedback layer which listens to the melody (pitch) of the previous output of the player itself;

- Purple (internal harmony): The feedback layer which listens to the harmony (chroma) of the previous output of the player itself;
- Red (external melody): The melody layer which listens to melodic (pitch) influences from exernal sources (audio/MIDI influencers);
- Blue (external harmony): The harmony layer which listens to harmonic (chroma) from external source (audio/MIDI influencers).

Figure 4.8: User interface to control the Balance between the dimensions, length of matching sequences for each dimension as well as decay time of peaks (Memory lengths).

State

The last state that was output is visualized in the lower part of the interface, as well as a Region Filter to select the desired range of states you want the player to jump to. Here you have also a visual feedback of the occurrence of a match or not, through the Match light. If green, a match occurred and is being played, if red, you don't have a match, if yellow, the player is playing a default sequence, generally linear, which happens either because there is no match — but Sparse is Off , so it defaults to the fallback behaviour — or because it is playing the time-out sequences.

Number of Matches

The green, purple, red and blue indicators (see Figure 4.10 shows the number of matches in each layer. Note that this may contain overlapping matches. The white indicator shows the total number of matches when all layers have

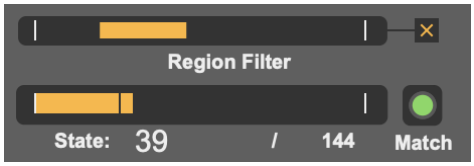


Figure 4.9: This part of the player's interface shows the last output state, as well as the region filter (if enabled), and if a match occured or not.

been merged. The merged result will not contain duplicates, and will in most cases be lower than the sum of the individual layers.

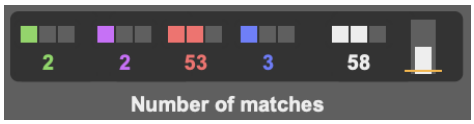


Figure 4.10: While the user doesn't interact directly with the peaks, they are still indicated in the user interface. Here, the colors green, purple, red, blue and white correspond to the number of peaks in the internal feedback layer (green: pitch, purple: harmony), external pitch layer (red), external harmonic layer (blue), and total number of peaks after merge (white) .

4.2.2 Manual Corpus Segmentation

When using certain corpora, particularly conceived and designed for a specific piece or performance, it has been noted that a custom segmentation of the corpus provided by the user could be highly effective in arranging and organizing the sound material. This is especially true in the case of ad-hoc composed material, where an elevate and refined understanding of the events in the corpus could positively affect the resulting interaction. In order to achieve this, a module for manual segmentation has been added in the Audio Corpus Builder Max object. This relies on a Python algorithm, using RegEx (Regular Expression) [65] to match pattern in the .txt files generated from certain DAWs, when manually adding markers and exporting them. The DAWs currently supported are ProTools, Reaper, Audacity and SoundStudio. These has been chosen according to the needs of the RepMus members and Somax beta testers, but it will be possible to extend this feature to other environments, in the future. A full overview of the class implemented for this is presented in the Appendix.

Chapter 5

Artistic Applications

When it comes to music technology systems, it is necessary not only to test that the implementation of these systems is well structured, but that their use in concrete music creation and co-creation contexts is satisfactory and leads to good musical results. In this sense, the work done in the last period has been essential to the development of Somax.

Thanks to the context provided by the REACH research program, it has been possible to organize several artistic residencies with world-class musicians in the field of improvisation and artistic research. These residencies are aimed at concerts in mainstream musical situations of clear reputation and high level, such as the ManiFeste festival in Paris, which will be held at the Centre George Pompidou, and Klang Festival, the most important international festival of contemporary music, based in Copenhagen and hosted in the Concert Hall of the Royal Danish Academy of Music.

Specifically, while the first concert will feature double bassist Joëlle Léandre as the musician interacting with the system and will be mainly marked by free improvisation, the second will involve percussionist Lorenzo Colombo, as part of a project in which Somax will attempt to break through the barriers of improvisation, extending into the field of assisted composition and real-time reinterpretation of pre-composed material. In fact, although Somax began as a co-improvisation system, its development and organic nature has allowed it to be used in broader contexts, where improvisation and composition are continuously in dialogue. To achieve this goal, the residency in question was marked by the work of Iannis Xenakis, and his piece Komboï, for percussion and harpsichord.

This chapter will thus expound on the musical and artistic research studies that were instrumental in setting up and finalizing the artistic residency works carried out over the past few months, using Somax as the main human-machine interaction environment.

5.1 A.I. Komboï

A.I. Komboï is a project conceived by the two musicians and researchers, Marco Fiorini and Lorenzo Colombo. While sharing different backgrounds, spacing from contemporary music to non idiomatic improvisation, they have the common intent of constantly experimenting new means of expression, and a shared deep knowledge of electronic music.

Willing to fulfill the inter-disciplinary objectives designed by the REACH project, this collaboration intends to bring up the potential of Somax as an application for musical improvisation and composition in the real world of high level performers and musicians, facing real life scenarios. In this way, the outcome of previous scientific research works is applied to concrete creative situations, building important interactions between music researchers, composers and performers, to reveal the mechanisms of co-creativity and build the epistemological boost required for this creative interaction to succeed.

Our ongoing research consists of 3 fundamental steps:

- Compositional research, starting from Xenakis's piece Komboï
- Collaborative research towards the commission of new compositions exploring Somax's potential;
- New solo/duo music work as conclusion of the data accrued from the above mentioned research.

Iannis Xenakis represents one of the pioneering figures in the creative, musical and broader artistic progress of the last 100 years. His compositional spirit and research influenced generations of composers and performers and his work still resonates to this day. Xenakis' work was avant-garde in many ways, developing ideas that furthered musical progress by implementing mathematical formulas and theories predicting the modern world, where computers and technology have become integral parts in most creative processes. This research is meant to celebrate Xenakis as a composer and pioneer, starting from the *Journée d'études en hommage à Iannis Xenakis* organized in May 2022 by the REACH project together with Mâkhi Xenakis, daughter of the composer.

We will start our research from Komboï, written in 1981 for percussion and harpsichord. The title, Komboï, means "knots," in this case of rhythms, timbres, structures, personalities (Xenakis 1982b). Komboï features some of the most relevant techniques developed by Xenakis such as the sieves and the stochastic ones. The mathematical approach generates a lot of possibilities, giving us the chance to research with Somax as an organic composer of material and music ideas.

Is it possible to instruct Somax to generate music material from the interaction between AI and a human musician? Let's imagine a percussionist performing a faithful version of Komboï. Our goal is to instruct Somax to perform the harpsichord part by feeding the software with data belonging to the same composition, generating an interaction between computer and musician with a real-time exchange of information and stimuli.

Our work will serve to reply the following questions:

- How can artificial intelligence create a coherent piece of music after being instructed to use the harpsichord material composed by Xenakis?
- Can artificial intelligence react to the percussion material by active listening, creating coherent music in real time?
- In a duo context, can we think of an inverse composition process where one of the two voices is generated as a real-time reaction to the other one?

With Komboï, Xenakis intended to enhance the timbre characteristics of the harpsichord and in particular its electronics, almost synthetic qualities. In our case, we want to keep the percussion part faithful to the original, experimenting with digital synthesizers as an alternative to harpsichord, still maintaining its electronic sound qualities already described. This operation is meant to give a modern sound to this masterpiece and to study different modes of interaction between acoustic and electronic sounds, human and digital performers. The module will be played directly from Somax, communicating with MIDI data.

A.I. Komboï is not a simple research project, but rather the necessity to explore technology as a resource for new creative possibilities. The project will be presented at various festivals becoming also material for a physical and digital release. The live activity will be accompanied by a large number of lectures with the aim of showing Somax and its immense possibilities. Another fundamental aspect will be open debates on artificial intelligence applied to music, art and creativity. Last, musicological and compositional aspects will be investigated, through open calls for commission pieces written specifically for the project in the form of a duo (percussions and Somax). In this way, composers could explore new forms of interactions as well as agogic research on notation and composition using the new opportunities that a tool like Somax brings up.

5.1.1 Komboï

Komboï (in Greek: *κομβοι*, meaning "knots") is a 1981 stochastic composition for amplified harpsichord and percussion by Greek composer Iannis Xenakis. It is one of the two compositions for harpsichord and percussion written by Xenakis, the other one being *Oophaa*.

Xenakis composed Komboï after a long collaboration with both harpsichordist Elisabeth Chojnacka and percussionist Sylvio Gualda, which at that time formed a duo and commissioned works for both instruments to other composers. As Xenakis worked previously with both musicians (he also composed Khoai for Chojnacka in 1976 and *Psappha* for Gualda in 1975), the composer was much more focused in exploring the timbral capabilities of both instruments by creating an homogeneous sound texture. It is indeed dedicated to Gualda and Chojnacka, the latter being the dedicatee of all five compositions for harpsichord by Xenakis. It was eventually premiered by the duo on December 22, 1981, at the Rencontres Internationales de Musique Contemporaine in Metz.

Analysis

The composition is in only one movement and takes approximately 17 minutes to perform. It is scored for one harpsichord, one vibraphone, two wood blocks, two bongos, three congas, four tom-toms, one bass drum, and seven terracotta flower pots. As put by Xenakis, Komboï explores "non-octave scales", its rhythm examines "anthypheresis" (as displacement of musical stress), and its timbres exploit "the antitheses or homeophanies of the amplified harpsichord and percussion." In this sense, Komboï means knots, as "knots of rhythms, timbres, structures, and personality," interweaving each one with others. To make instruments blend more effectively, Xenakis used amplification in all of his works for harpsichord [66]. Both harpsichord and percussion are capable aggressors, particularly when the harpsichord is amplified as required for this piece (and all of Xenakis' harpsichord music). But Komboï instead seeks to explore the more delicate and refined colors of these instruments. Xenakis uses the knots metaphor to interweave characters, personalities, materials and patterns. It is a tribute both to the dedicatees and to their instruments.

The composition can be divided in five sections, which also contains interludes and variations. The opening section features an ostinato played by the bongos, while the harpsichord plays mainly rising tone clusters. In this case, the bongos stress beats and offbeats unevenly to produce the sensation of "anthypheresis". Here, Xenakis explores the relationship of the sonority

between a somewhat ordered percussion with the clusters played by the harpsichord. The relationship between the "ordered" sounds of percussion instruments and the scales of pitched notes is one of the "knots" Xenakis explores here. The second section, marked Crystalline, mixes the harpsichord and the vibraphone, and the relationship between these two instruments seems to fuse more effectively. The sonorities fuse in a way that is utterly magical and the composer explores the combination through fluctuating clouds of notes that pass the spotlight back and forth between the two. After that, the vibraphone changes to the wood blocks and, later on, the harpsichord starts a lengthy solo.

Then, the percussion joins with the sound of the flower pots, which blends with the needle-like sound of the harpsichord, which uses an ostinato of seven chords. The delicate, ringing tone of the flower-pots blends extremely well with the needle-like tone of the harpsichord. It makes for wonderful theater to watch the percussionist playing them, of course, but the incorporation of this sound is a stroke of genius. Set against the flower-pots, Xenakis introduces a randomized ostinato for the harpsichord built from just seven chords. It is another knot, matching the limited collection of five sonorities in the percussion with a limited collection of sonorities in the keyboard. It is the rhythmic and timbral interplay that carries the focus and Xenakis would make much use of similar passages in later pieces. Komboï, in any case, is an evocative, engaging work that highlights a gentler side of this iconoclastic composer.

5.1.2 Sieve Theory

Komboï, as most other late pieces of Xenakis, is based on Sieve Theory. Xenakis developed Sieve Theory during his stay in Berlin, having received a Ford Foundation grant to live and work in West Germany, in 1963, and published first material on it in [67] and [68], leading to an ultimate publication of it in [69]. The theory mainly concerns the creation of scales, arrived at through the combination of residue class sets.

The Sieve of Eratosthenes

The primordial sieve in mathematics is known as the Sieve of Eratosthenes, an ancient algorithm for finding all prime numbers up to any given limit, as shown in Algorithm 1. The earliest known reference to the sieve is in Nicomachus of Gerasa's *Introduction to Arithmetic*, an early 2nd century CE book which attributes it to Eratosthenes of Cyrene, a 3rd century BCE Greek

mathematician, though describing the sieving by odd numbers instead of by primes [70].

Algorithm 1 Sieve of Eratosthenes

Require: An integer n > 1
Ensure: All prime numbers from 2 through n
  let A be an array of Boolean values, indexed by integers 2 to n, initially all set to true;
  for $i = 2, 3, 4, ...,$ not exceeding $\sqrt{n}$ **do**
    instructions;
    if $A[i]$ is true **then**
      for $j = i2, i2 + i, i2 + 2i, i2 + 3i, ...,$ not exceeding n **do**
        set $A[j]$:= false;
      end for
    end if
  end for
  Return all i such that $A[i]$ is true.

The importance of this technique to Xenakis is fundamental; it has provided him with a method for filtering elements in order to create and manipulate structures. Furthermore, Xenakis's and Eratosthenes's methods share a common origin in the foundations of arithmetic.

An intuitive way to unfold the algorithm consists on the following simple procedure: we write down in a matrix, in ascending order, all the integers from 2 to n. We leave the first element (2) and erase all its multiples, we leave the next number that has not been erased (3) and erase all its multiples, and so on. We proceed until we reach prime number p, where $p \leq \sqrt{n}$. The remaining integers are the prime numbers between 2 and n. The result consists of four parts (each for one stage of the process) and shows the cross-outs for each element: in the top left part of the table we have erased all the multiples of 2 (every second number), in the top right part all the multiples of 3 (every third number), and so on for 5 and finally 7, which is the greatest prime ≤ 50.

Sieve Theory in Xenakis music

What Xenakis drew from this is not merely the idea of filtering – passing the elements of a set through a sieve – but also the process of using starting points and steps of a specific distance. However, Xenakis's application of Sieve Theory is not intended to determine primes: his sieves allow both the starting points and all the following steps. Each of the four stages in the Sieve

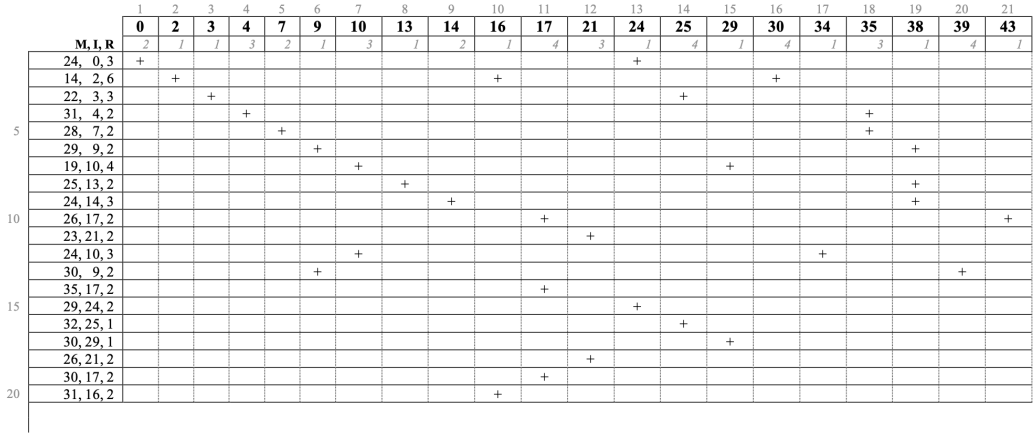
of Eratosthenes is for Xenakis an infinite set of numbers, that might coincide with each other in a more or less complex way. The degree of complexity is a matter of compositional decision and aesthetics. This was done in a period when he would attempt to take further his investigation towards formalisation; this time though not with stochastics and probabilities, but with the aid of the deterministic laws that govern Number and Set Theory. However, both cases were for him a matter of generating outside-time structures of music . A sieve, then, refers to a selection of points on a straight line; this is the abstract image of sieves: "Every well-ordered set can be represented as points on a line, if it is given a reference point for the origin and a length u for the unit distance, and this is a sieve". Xenakis referred to unit distance (e.g. the semitone in the major diatonic scale) also as Unit of Elementary Displacement (ELD) [69].

The theory was used in order to construct symmetries at a desired degree of complexity. This was achieved by the combination of two or more modules. A module is notated by an ordered pair (m, r) that indicates a *modulus* (period) and a *residue* (an integer between zero and $m - 1$) within that modulus [1]. For example, for $m = 3$ and $r = 1$ we have the following module: $(3, 1) = \{1\ 4\ 7\ 10\ 13\ ...\}$. Elements that lie in distance equal to the value of the modulus are said to be congruent modulo m. In other words, elements that yield the same residue (r) when divided by the same number (m) belong to the same congruence class. In the example, elements 4, 7, and 10 are congruent modulo 3. By applying the set-theoretical operations of union, intersection, and complementation, or a combination of them, it is possible to construct more complex sieves. Since this section is intended to give an introduction to the theory, for further discussion and in-depth analysis see [71].

Xenakis provided two computer programs for the treatment of sieves. They are found both in his article of 1990 [69] and as Chapter XII of the 1992 edition of Formalized Music [1]. Their titles are indicative of their function: "A. Generation of points on a straight line from the logical formula of the sieve", and "B. Generation of the logical formula of the sieve from a series of points on a straight line".

A Model for Sieve composition

One of the basic problems in Sieve Theory is the redundancy of formulae for a single sieve. That is, whereas the problem of decomposed formulae redundancy is overcome by prime factorisation, the redundancy of simplified formulae is not as straightforward to resolve. It is therefore needed to determine another level of periodicity, different from the overall period of the



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	0	**2**	**3**	**4**	**7**	**9**	**10**	**13**	**14**	**16**	**17**	**21**	**24**	**25**	**29**	**30**	**34**	**35**	**38**	**39**	**43**
M, I, R	*2*	*1*	*1*	*3*	*2*	*1*	*3*	*1*	*2*	*1*	*4*	*3*	*1*	*4*	*1*	*4*	*1*	*3*	*1*	*4*	*1*
24, 0, 3	+												+								
14, 2, 6		+								+						+					
22, 3, 3			+											+							
31, 4, 2				+														+			
28, 7, 2					+													+			
29, 9, 2						+													+		
19, 10, 4							+								+						
25, 13, 2								+											+		
24, 14, 3									+										+		
26, 17, 2											+										+
23, 21, 2												+									
24, 10, 3							+										+				
30, 9, 2						+														+	
35, 17, 2											+										
29, 24, 2													+								
32, 25, 1														+							
30, 29, 1															+						
26, 21, 2												+									
30, 17, 2											+										
31, 16, 2										+											

Figure 5.1: Simplified Matrix of the sieve of Nekuïa, from [71].

sieve. To overcome this problem, Xenakis investigated formulas to decompose and simplify the development of sieves, as well as models to control their possible operations, such as inversion, simmetry, periodicity, etc. One important piece he wrote to apply these concepts is *Nekuïa*, which sieve can be compared to the one of Komboï.

The period of a sieve is external to it and symmetry is an internal property; but when a sieve is asymmetric, 'a more hidden symmetry' might exist. The external periodicity is none other than the sieve's overall period. The inner periodicities of the sieve are shown by a simplified formula in the form of elementary modules. Each one of these elementary modules has a single modulus whose multiples produce some of the points of the sieve. These points are conveniently shown by a matrix, that Exarchos calls "simplified matrix" [71]. To construct a simplified matrix we write all the sieve's elements in the top row and all the modules of the simplified formula in the leftmost column (such that this column represents the simplified formula). The intervallic structure of the sieve (in semitones) is shown under the actual points of the sieve. We then mark all the cells where the elements of each module (rows) meet a point (columns) of the given sieve.

The simplified matrix of the sieve of *Nekuïa* is shown in Figure 5.1. The labels (M, I, R) stands for: Modulus, Initial point, Reprises of the Modulus. The matrix shows that each of the twenty modules cover several points of the sieve. When a simplified formula is based on the inner periodicities there is no information indicating the overall period, This means that the focus is now on the sieve's internal structure. he process of constructing an inner-periodic formula ignores the period and takes into account only the

intervallic structure of the sieve. The redundancy of inner- periodic formulae can be overcome by checking every single point of the sieve and assigning to it the smallest possible modulus. More specifically, we can find for each point, the module with the smallest modulus, that either starts on this point or produces it later. By progressing to a simplified notation, Xenakis showed a new way of revealing the hidden symmetry of sieves, which are now viewed as multiplicities of periodicities.

Sieve of Komboï

The opening sieve of Komboï was also used partly in another work of the same year, *Pour la Paix*. Is deviates from Xenakis's usual practice of constructing sieves with the greatest interval the major 3rd. As shown in Figure 5.2, there is only one major 3rd, the lowest interval of the sieve. It is therefore a sieve with greater density than one would expect: D = 0.52 (44 elements occupy a range of 84 semitones). The simplified formula includes 19 modules. Recall that the same number of modules produced the sieve of Nekuïa which is slightly less dense, with D = 0.48. In this respect then, the opening sieve of Komboï is more inner-symmetric than that of Nekuïa. Indeed, whereas in the latter there are two non-periodic modules, in the chart of Figure 5.2 only one module is outside the inner-symmetry curve, and more periodic modules are concentrated in the region of the chart where R = 3 than that where R = 2. But if we compare it with the sieve of Nekuïa, which has 20 modules all of which are periodic, we see that in the sieve of Komboï most of the modules are more periodic, with a non-periodic breaking the inner-symmetry.

There are several sieves in the duo, but most of them appear only partly, and this prevents a complete analysis. However, later in the work one sieve appears in a great range; it is shown in Figure 5.3. This sieve is less dense (it includes intervals of major 3rd); there are 37 points in a 82-semitone (D = 0.47) range with 14 modules, one of which is non-periodic. The small number of modules does not imply that these are, in average, more periodic than those of the opening sieve, due to the fact that the density and range of the sieve (as it appears in the work) are smaller than that of the opening one.

Although pure sieve composition is beyond the scope of this current artistic research, it is interesting to point out that this is an open field of studies, still bringing out new possibilities and perspectives to music generation techniques and scenarios. As an example of that, Ariza in [72] demonstrates a new object-oriented model and full Python implementation of the Xenakis sieve process

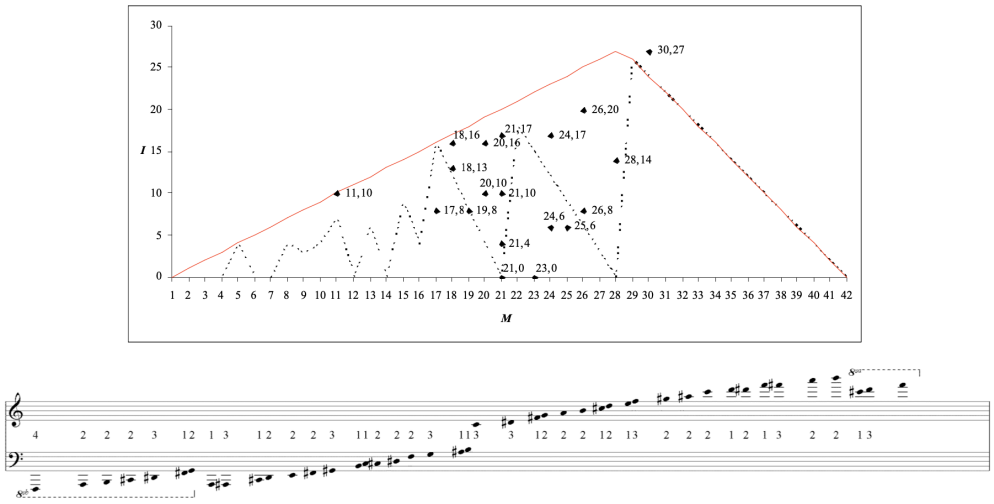


Figure 5.2: Opening Sieve of Komboï, from [71], with reference to Modulus M on the x-axis and Initial point I of each period of the sieve on the y axis.

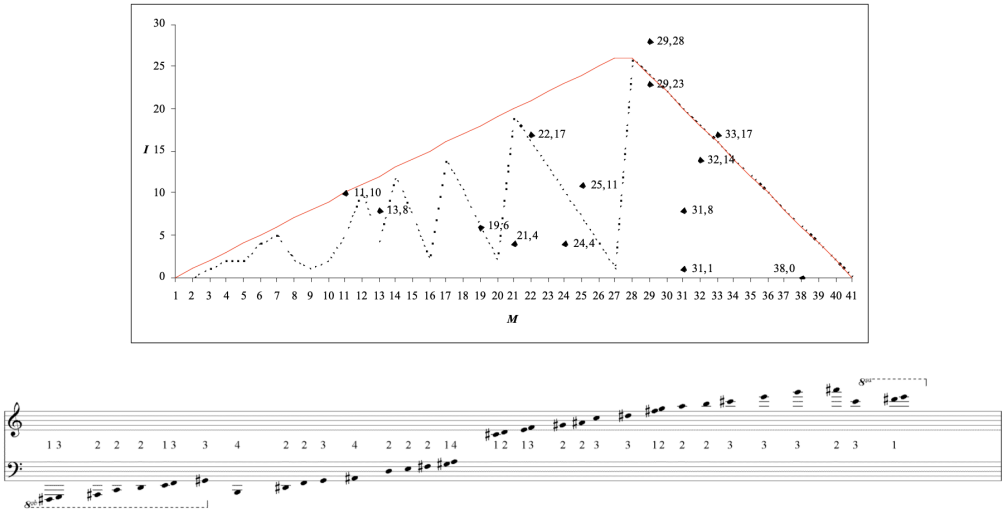


Figure 5.3: Sieve of Komboï, from [71], with reference to Modulus M on the x-axis and Initial point I of each period of the sieve on the y axis.

5.1.3 Setup

The full technical and instrumental setup for the performance to be premiered at Klang Festival in Copenhagen on June 6th 2023 is shown in Figure 5.4. This consists of two musicians, Lorenzo Colombo on percussion and Marco Fiorini on Somax. The full percussion set is composed by one vibraphone, two wood blocks, two bongos, three tumbas, four tom-toms, one large bass drum and seven flowerpots.

All of these are captured by a pair of overhead narrow condenser microphones, sending the audio of all the percussion to an Ableton Live session. From here the audio signal of the percussion is sent to a second computer, the one running Somax. These signals are received in input as influences in the Somax environment. On this machine, a Max/MSP patch using Somax as the core generative co-creative system has been prepared to manage the communication between the external musician and the software, and to be played in real-time. The Somax players used in this session are set as MIDI only, as the corpora used in the performance have been transcribed with various techniques, according to the different composed sections, in MIDI files, as described in the next section. The MIDI output of the players is then sent to the Arturia MiniFreak for the sound generation; from here the audio goes to the front of house as a stereo signal. The control of the Somax parameters, as well as all the cues to navigate the piece, comes from a Novation Launchpad XL, connected to the same Max/MSP patch.

5.1.4 Interaction Strategies

This artistic research work has brought to light new possibilities for the use of Somax in contexts initially unrelated to it. Although born as a software dedicated to human-machine improvisation, the system was also found to be stable and effective in an augmented composition situation, or mixed music interpretation according to the paradigm of computer-aided composition.

Komboï is a piece that features not only complex compositional techniques, but sections with very different overall harmonic and rhythmic character, with passages of considerable performance difficulty. Our research focused on trying to make the overall design of the piece as faithful as possible to the original, but leaving room for the co-creativity introduced by Somax and the surprise factor related to musical improvisation. To succeed in this performance, various synchronization techniques and interaction design strategies were tested and considered, until a series of concepts were solidified and proved successful.

Following the setup illustrated in the previous paragraph and visible in

Figure 5.4: Technical and instrumental setup for A.I. Komboi, as performed by Lorenzo Colombo and Marco Fiorini at Klang Festival in Copenhagen on June 6th 2023.

Figure 5.4, Lorenzo Colombo studied the percussion part in a manner faithful to the original. His signal is then captured and sent in real time to the computer with Somax (controlled by me). The signal is then analyzed according to Somax's system of influences, coding into onset (tempo), pitch (melodic content) and chroma (harmonic content) parameters. Since a large part of the percussion set consists of instruments that are not properly tuned, a bonk algorithm, as opposed to a yin algorithm, was chosen for the sections concerning these. This is because this algorithm works much better with non-tuned or polyphonic material, making it ideal in general for percussion instruments. During the vibraphone section, on the other hand, the yin algorithm was used, given the presence of strongly melodic material.

In terms of rhythmic context, the piece has precisely sections with a very different character. In the opening of the piece, as can be seen in Figure 5.5, both percussion and harpsichord play on the same pulse, but varying the rhythmic subdivision. For sections of this type, an onset sending time from the influences to the Somax players of 0 seconds was used, thus achieving an instantaneous reaction. With regard to rhythmic subdivision, on the other hand, the corpus used for this section was transcribed rhythmically continuous, but using the Output Probability parameter (described in Section 4.2.1) it was possible to alter the rhythmic scansion, thus producing syncopations

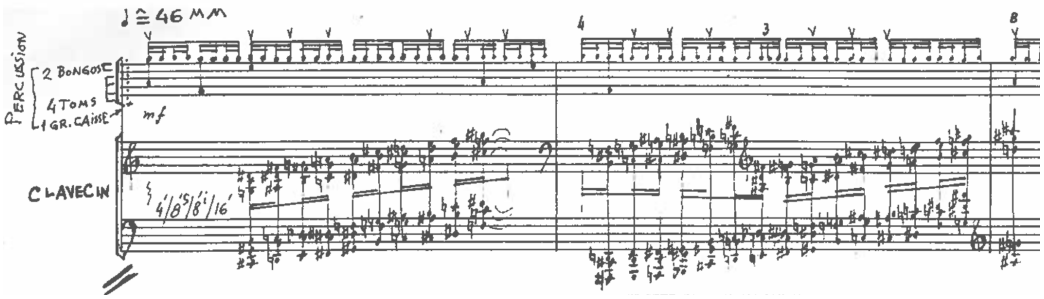


Figure 5.5: First two bars of Komboï. Both percussion and harpsichord share a common beat but have different subdivisions (32-notes against 16-notes).

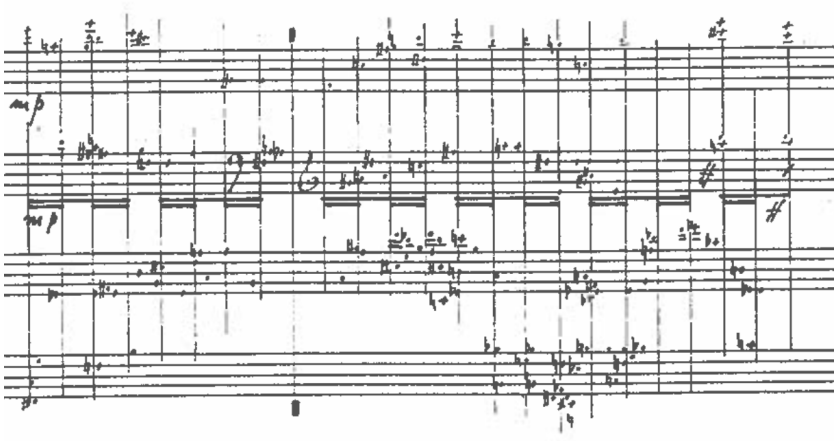


Figure 5.6: One of the many knots, or clouds of notes during the piece. Both percussion and harpsichord play with the concept of density over a common beat, that is however not explicitly highlighted.

and enlargements of rhythmic continuity, arriving at subdivisions virtually identical to those written in the score.

A similar strategy was applied in the sections characterized by the idea of "rain", nodes in which the rhythmic pulsation remains implicit but the agogics of the performative gesture is underlined by real cascades of notes, which respond and articulate each other (see Figure 5.6). In these sections, the vibraphone activates the generation of the Somax response, which rapidly plays sequences of notes, rhythmically articulated again by the Output Probability parameter. To obtain a convincing and organic sound cloud effect, two different corpora, one for the right hand (high register of the harpsichord) and one for the left hand (low register), have been loaded onto two different Somax players, which play simultaneously.

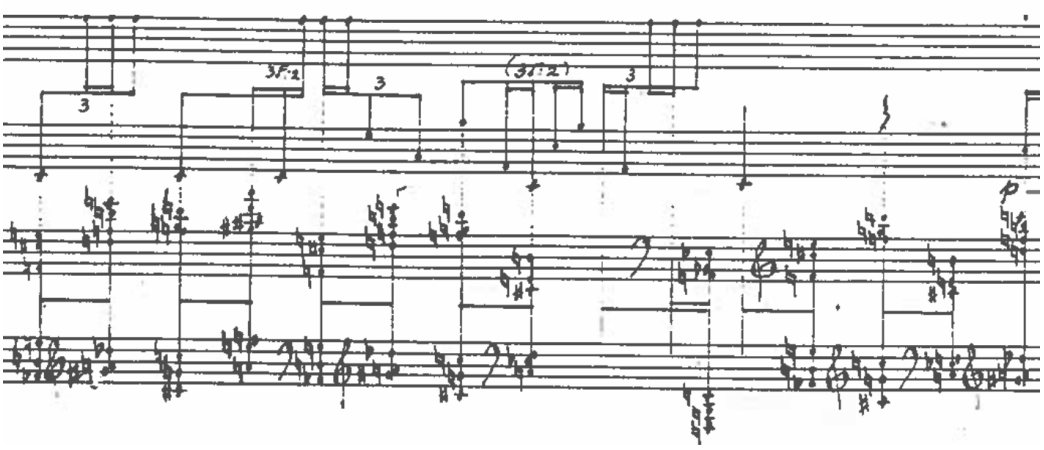


Figure 5.7: A moment of heterogeneous rhythmicity in Komboï. Both instruments play in cues, but with different rhythmic divisions interlocked together.

A decidedly more complicated situation was that of synchronizing particularly precise but different rhythmic interventions. This occurred both when Xenakis overlaps polyrhythms that do not imply a subdivision reaction of the same meter (see Figure 5.7), or other moments in the piece in which the two instruments do not play at the same time, but in consecutive interventions, always rhythmically synchronized. To solve the problem, given that, as explained in Chapter 3, rhythmic synchronization and in general the concept of temporal adaptation between different agents is a research field still open in Somax, a synchronization system managed by Ableton Live was used. Here, bangs relating to the start of the various cues are sent from one computer to another, thus generating onset influences according to the desired moment. Although this strategy does not fall within the solutions natively offered by Somax, it is a well-established technique, even in contemporary and electronic music performances in general, and it has proved to be a winning idea for extending the rhythmic synchronization of the two performers.

In total, 22 MIDI corpora have been transcribed, keeping in mind the compositional processes conceived by Xenakis, and the different possibilities of rhythmic interaction. These corpora are loaded during the performance into two different Somax players. 16 different cues were necessary for the synchronization of the different corpora and for the executive scrolling of the piece. In the various cues, in addition to loading the corpus, via messages in Max/MSP, the character of the players was changed, according to parameters designed ad-hoc for the success of each different section. The dynamics were instead controlled directly by the master volume potentiometer on the Arturia MiniFreak, defining a dynamic range that goes from piano, to mezzo-

forte, up to fortissimo. The control of the cues and of the internal parameters of Somax (mainly the Output Probability parameter was the one with which the performer interacted in real time, while the others were programmed and set at each new cue) was managed by the Novation Launchcontrol XL controller.

5.2 REACHing OUT - Jöelle Léandre

In addition to the artistic residency described in the previous sections, another important artistic research work, fully focused on improvisation, has been carried on with Joëlle Léandre, and will be briefly discussed in this section. Joëlle Léandre is a renowned double bass player, known for her collaborations with other musicians in the field of improvised and contemporary music [73]. Born in France on September 12th, 1951, she has performed with Pierre Boulez's Ensemble InterContemporain, and worked with Merce Cunningham, Morton Feldman, John Cage and Giacinto Scelsi. Both Cage and Scelsi have composed works specifically for her [74].

5.2.1 Setup

Dealing with Jöelle Léandre, a chamaleontic improviser who often sings while playing her double bass, we used different microphones to capture both her bass and her voice. I used also an electric guitar, providing more influences to Somax and trying to extend the common paradigm of a single instrument interacting with the software (more on this in the following sections). These instruments were captured through microphones and D.I. boxes, and their signal was sent to all the three different Somax stations, allowing each computer musician to choose which audio influence use to drive the different Somax players. A wide range of different corpora has been tested, ranging from contemporary pieces of Cage, Scelsi, Stockhausen and Xenakis. During the first studio sessions, different solo improvisations of Jöelle Léandre have been recorded. These recordings have been used in the further sessions to build different corpora and let her play with them, investigating another hypothesis advanced during the residency (see further sections) and experimenting over the topic of recalled memory in a musical performance.

5.2.2 Studio Sessions

The studio sessions of this artistic research spanned from June 2022 to May 2023, in the IRCAM Studios 2, 3 and 5. The lineup of musicians and re-

searchers consisted of:

- Jöelle Léandre: double bass and voice;
- Gérard Assayag: Somax;
- Mikhail Malt: Somax;
- Marco Fiorini: electric guitar and Somax;
- Manuel Poletti: Spatialization and Live effects.

Before our first studio session, I analysed a wide number of previous performances using Somax. The factor common to all of these performance is the fact that everyone performing with Somax uses it as if it as instrument itself. The usual performance scenario consists of a duo where:

- Player 1: plays an acoustic or electric instrument;
- Player 2: plays Somax, as a RIM (Réalisateur en Informatique Musicale, term coined in 1977 by Pierre Boulez at IRCAM to define expert musicians in charge of the electronic part of a performance).

This analysis brought up a few hypothesis/questions I tried to fulfill during the upcoming studio sessions:

- what if I want to play both Somax and an instrument?
- what if I want Somax to be an extension of my instrument?

This should apply to different performance constellations, covering a wide array of possibilities, e.g. solo, duo, trio, etc.

During the first studio session we experimented with two different scenarios, trying to collect feedback and opinions on these research questions:

- *Scenario 1 - Playing both guitar and Somax*: this turned out to be not comfortable, because the player needs to use his/her hands both for the instrument and Somax, and one could easily take away the attention from the other. A possible statement is that it can probably work with singers, having hands free while performing
- *Scenario 2 - Focusing more on playing guitar and letting Somax playing as an independent player/agent itself*: the problem here is that there is no fine tuning of the parameters that dynamically create the little interesting, evolving and responsive nuances in a performance.

To fulfill both these scenarios I designed two possible solutions that have been tested first individually and then in the second studio session:

Solution 1

- Record or prepare multiple corpora of performances with your main instrument;
- Build these corpora in Somax;
- Perform using only Somax.

In this case one can totally focus on playing Somax but with the corpus of his/her instrument and his/her past self playing (keeping also the philosophical shadow of the player/agent itself/yourself), while playing together with another player (The usual performance scenario).

Solution 2

- Use Somax as an extension of your instrument: e.g. augmented electric guitar;
- Control it with a MIDI pedalboard: this keep the player's hands free and it's a common practice routine for an electric guitarist which is very used to interact with pedals;
- But which corpus should we use?

To fulfill this workflow, one could recall NMF (Non-negative Matrix Factorization) [75] [76], a technique that uses bases to build a dictionary that can be recalled in real time based on the activation of these bases. The peak matching algorithm of Somax [77] already does something very similar conceptually to the NMF matching, so the main problem would consist in building the dictionaries (different corpi) and interactingt with them. I thought that it could be interesting to separate different techniques/parameters into different corpora, according to the instrumental techniques of choice. For instance on an electric guitar I could have corpora for:

- guitar harmonics (thus working on timbre, register and harmony);
- extended techniques (timbre);
- chords only (harmony);
- intervals only (melody).

As a starting point, I recorded 18 different harmonics on guitar (the ones the sounds louder and clearer), built a corpus with it and instantiated Somax with two players, both using this corpus. I then expression pedal and a MIDI interface to control the Quality Threshold of a player with cut and sparse and long timeout (more independent, being able also to propose some new material), while I had another player without timeout (more reactive player). Here, cut, sparse and timeout are new parameters, implemented in the last months as part of the new Somax version. Specifically, cut controls if Somax should play another audio segment from its corpus, if a new match with the incoming influences is not found. Sparse is a control that is related to a new quality threshold parameter: when sparse is active, every incoming event from the influencers that has a matching quality inferior to the one set by the threshold will be excluded. Finally, timeout is a parameters that sets an interval in which the Somax players could still go on outputting events from the corpus, even if no incoming new matches are found. In this way a good interaction between reaction and interaction has been reached, and a resulting piece can be heard[1].

Following the considerations and the outcome of the first studio session, we designed a couple of different scenarios for the improvised piece of this one.

The first take has been performed asking Jöelle Léandre to play with a previous memory of herself. Specifically, we used as Somax corpus a solo of her, recorded on the previous session. This was to focus our research on a sort of augmented instrument with a memory, particularly interesting in the case of a player like Léandre who worked so much with this concepts in Scielsi and Cage music.

Another interesting issue was trying to have multiple audio influencers routed to the same Somax, since previously every instance of the software took as input only one influencer for each class (audio or MIDI). This has been tested in two different takes, one where Jöelle Léandre was playing double bass and singing at the same time and another one where I played only guitar without focusing on the software. This is also connected to the possible ways of controlling and interaction discussed after the first studio session. More work and research on solo interaction with MIDI pedals will be held in the future as part of the autonomous agents and behaviour exploration.

The overall feedback on these sessions was quite satisfactory from all participants and led to the decision of an upcoming preparation for a concert with this setup in the well known ManiFeste festival at Centre George Pompi-

[1] `https://drive.google.com/file/d/1PVUoBCga8bHDPeJ5CqNMxUFasyEpUqQY/view?usp=share_link`

dou on June 16th 2023. The artistic residency continued focusing on deeper agent behaviour and different forms of interactions, such as a sort of installative scenario generated by Somax that could be perturbed and renewed by the intervention a musician. Finally, to give some form of structure in the context of a whole improvised concert with such a wide constellation of performers, a set-list has been thought. In this set-list, pre-fixed configurations of duo/trios/tutti has been decided, with the possibility to be spontaneously extended with the entrance of other improvisers.

This residency turned out to be a solid work of research both in the scientific and artistic domain, leading to many improvements not only in the Somax system but also in the way musicians could interact with it. The feedback collected during this period will be used for further research on agent architecture and interaction control, as it will be briefly described in the next chapter.

Chapter 6

Future Work

Although the work carried out by the Music Representation team at IRCAM - STMS Lab in recent years is to be considered highly satisfying and important in the development of music co-creativity systems, thanks to the work within the same team, it has been possible for me to envision a future trajectory of possible improvements and new research paths to be taken and expanded. Some suggestions resulting from these reflections and recent new collaborations are briefly set out in the following chapter.

6.1 Mapping

The main question to be solved is related to the actual choice of which mapping strategy to implement. The ultimate goal in designing new Digital Musical Instruments (DMI) is to be able to obtain similar levels of control subtlety as those available in acoustic instruments, but at the same time extrapolating the capabilities of existing instruments [78]. Considering mapping as part of an instrument or an application, two main directions could be deduced from the analysis of the existing literature:

- The use of generative mechanisms, such as neural networks.
- The use of explicitly defined mapping strategies.

Although possible mapping with a neural network have already been explored in previous iterations of Somax [79], in the end our decision was to use explicit mapping strategies, presenting the advantage of keeping the designer in control of the implementation of each of the instrument's component parts, therefore providing an understanding of the effectiveness of mapping choices in each context [80].

6.1.1 Explicit mapping strategies

The available literature generally considers mapping of performer actions to sound synthesis parameters as a *few-to-many relationship* [81]. Considering two general sets of parameters, three intuitive strategies relating the parameters of one set to the other can be devised as [80]:

- *one-to-one*, where one synthesis parameter is driven by one performance parameter,
- *one-to-many*, where one performance parameter may influence several synthesis parameters at the same time, and
- *many-to-one*, where one synthesis parameter is driven by two or more performance parameters.

Concerning explicit mappings between two sets of parameters, many ways of abstraction of the performance parameters have been proposed, from perceptual parameters [82] to focusing on continuous parameter changes represented by gestures produced by the user [78] [83].

6.1.2 Three-layer mapping

The latest version of Somax consists of a very large number of controls, taking care of different individual aspects of the architecture paradigm. Interacting with such a wide number of parameters is common practice to expert computer designers or Max/MSP programmers but may constitute an issue for a musician with a low confidence in computer usage. Another problem could arise if a player wants to control Somax at the same time as playing his/her instrument (this will be explored in a further section). In this case, having a control on a smaller range of parameters could be beneficial.

In designing an explicit mapping for this wide range of parameters, a good approach could be to adopt a three-layer mapping model. In this model, the first layer is interface-specific, since it converts the incoming information into a set of chosen (intermediate or abstract) parameters that could be perceptually relevant. These are then mapped – in a second independent mapping layer – onto the specific controls needed for a particular synthesis engine. The advantages of this model are that the first mapping layer is a function of the given input device and the chosen abstract parameters, while the synthesis engine could be changed by changing the second mapping layer, e.g. from model controls to FM synthesis or physical modelling, since the second mapping layer is not dependent on the control parameters directly. The proposed mapping layers (see Figure 6.1) are thus:

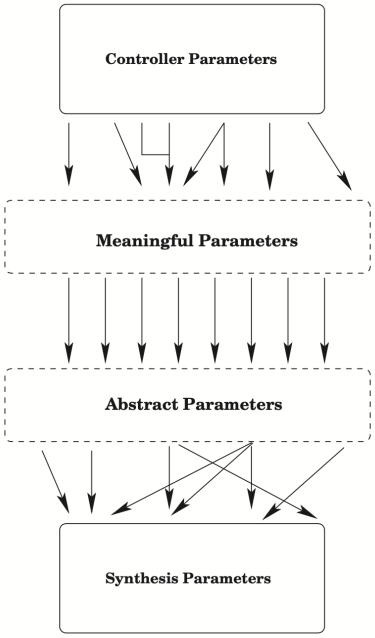


Figure 6.1: Three-layer mapping model, as presented in [84].

- Extraction of meaningful performance parameters.
- Connection of performer's (meaningful) parameters to some intermediate representation set of parameters (for instance, perceptual or abstract).
- Decoding of intermediate parameters into system-specific controls.

6.2 Towards an Agent-Oriented application

This layer of Control Parameters could also be intended as a set of behaviours, more related to human perception. In [85] Golvet et al. investigated the idea of adopting congruent and interacting behaviours in the context of collective free improvisation. Exploring the idea that the goal of delivering a creative output could eventually benefit from disagreements and autonomous behaviors, their research suggested that relational intents might function as a primary resource for creative joint actions.

This could bring to the extension of the Somax paradigm integrating more agents to the system, similarly to what happens in George E. Lewis' *Voyager*

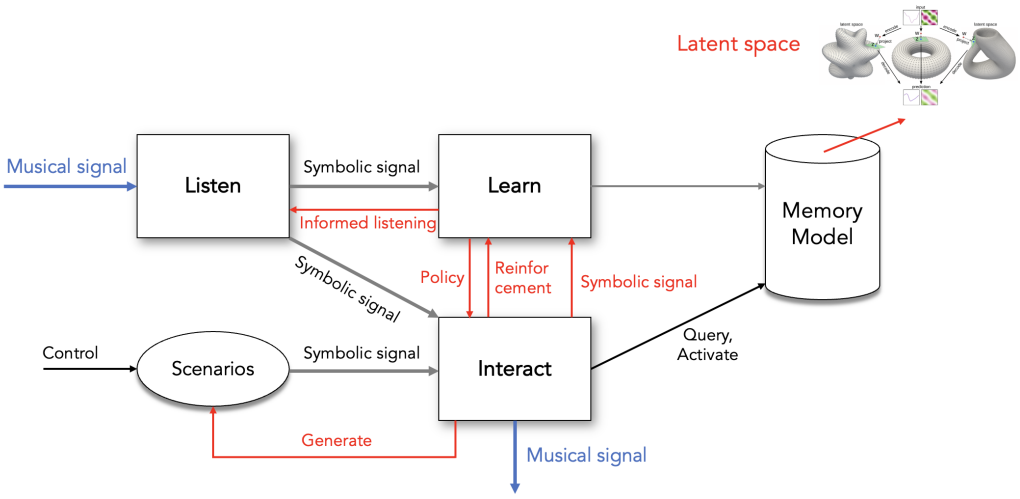


Figure 6.2: Envisioned future agent architecture idea for real-time learning.

[2] [3]. These behaviours could in a way define limits for the wide range of parameters, where the agents could analyse different input features (e.g. based on audio or MIDI descriptors) and take care of dynamically changes the inner parameters of Somax. An expected result of this agent-oriented approach could be the possibility to have organic and autonomous changes in Somax behaviour, augmenting its idea of agency and providing new ways of interaction for the musicians (e.g. solo performance with Somax acting as an augmented instrument, as described in the previous chapter), as shown in Figure 6.2. In this envisioned architecture, an agent could query a representation space (Memory) as a response to a live context (Listen). In addition to input-based learning processes, the agent is thus learning from the very course of interaction through reinforcement. An agent could be submitted to external scenarios (related to composition), but can also generate its own scenarios as a result of interaction. Finally, as we have the musicians,and the system (model of music making), there could be also a latent space where they join; especially now, in the era of deep learning and other A.I. methodology, it could be possible to build a powerful latent space where the musician and the system can meet and have a match as a way of playing together, that can become more rich.

Future work will be carried on on this topic, also based on the model proposed by Saint-Germier and Canonne in [86].

Chapter 7

Conclusion

The goal of this thesis research was to explore the major aspects of the REACH project, focusing in particular on Somax. At the current time, Somax represents the state of the art in the Music Representations team at IRCAM STMS Lab, concerning human-machine co-creativity and AI interaction. This projects finds its place in a broader path of research, thanks to the European project REACH[1] and the national project MERCI[2]. Thanks to this environment, the work described in this thesis has been part of a wider set of research activities.

During my stay in the Music Representations team at IRCAM, first as an intern and then as a research assistant, I could benefit from this rich and inspiring setting, following and contributing to the developing of the current version of Somax. Since its latest update, the software has undergone a series of improvements, regarding its features, and its user interface. One notable improvement is its new modular architecture, which now makes it both a Max application and a Max library, compatible with the workflow of Max programmers and computer music designers who are familiar with it.

Through different ongoing artistic residencies with well known improvising musicians and direct confrontation with other researchers at STMS, we've been able to design a longer trajectory in this research that will address enhancements to the Somax co-creative improvisation environment, including high level autonomous agents [87], automatization of refined interaction strategies and behaviours, exploration of time structures and scenarios [88], and embodiment through physical instruments.

The work here described has been presented at the last IRCAM Forum Workshop in Paris [89] and will be presented in the second International

[1] `http://repmus.ircam.fr/reach`

[2] `https://www.ircam.fr/projects/pages/merci/`

Conference on Hybrid Human-Artificial Intelligence HHAI2023 in Munich in June 2023.

On the artistic side, I will perform using the latest release of the described software in two important international festivals. The whole Somax team will take part to a concert for ManiFeste festival at Centre George Pompidou together with the world renowned bass player and improviser Joëlle Léandre, and the rock band Horse Lords[3]. Another field of artistic application for Somax will be explored in a performance for Klang, the most important international festival for contemporary music in Denmark[4]. Here we will use Somax to reconstruct a pre-composed piece by Iannis Xenakis, thus extending the scope of the software to assisted composition and pure composition practices, while maintaining its strong root of co-creative interactive system. These artistic collaborations are crucial, as they are a concrete manifestation of a research project that is as ambitious as it is successful.

The use of Somax in important contexts of musical creation and research necessarily opens the way for new forms of musical creation and interaction, which are increasingly taking hold in the form of mixed reality environments and post-modern artistic research. As a researcher and a musician, I firmly believe that new forms of co-creativity are needed to enhance the artistic creation of post-modern times. Mixed reality and new interactive tools will evolve the practice of creation in all art forms, and enrich the criteria by which we enjoy art, just as the introduction of major technological and philosophical innovations have already done in the past century. I expect these tools to be increasingly part of everyone's artistic creation, providing new means of expression and research for musical intent of communication and interaction with the unexpected aspects of improvisation.

With this, it is hoped that the research done during this study can work as a personal foundation for future relevant investigations in these fields and could lead to improvements to the current state of the art of cyber-human musicianship.

[3] `https://manifeste.ircam.fr/agenda/reaching-out-28/`

[4] `https://klang.dk/2023/program/lorenzo-colombo-debutkoncert`

Appendix A

Manual Corpus Segmentation

Here it is presented the `manual_text_formats.py` Python class for manual corpus segmentation, addressing custom user segmentation derived from ProTools, Reaper, SoundStudio and Audacity. The file is available in the Somax2 GitHub repository[1].

```
import inspect
import logging
import re
import sys
from abc import ABC, abstractmethod
from typing import List, Type, Tuple, Dict, Any, Optional

from somax.features.feature import CorpusFeature


class Constants:
    FLOAT = r"-?(?:\d+\.[\d]*|\.[\d]+)(?:e-?\d+)?"
    COMMENT = r"^\\s*?/\\*"
    EMPTY = r"^\s*$"
    TEMPO = COMMENT + r"\\s*?tempo"  # Format: /* tempo (...)
    \d+ (...)


class ParsingError(Exception):
    def __init__(self, message: str):
        super().__init__(message)


class TextFormat(ABC):
    @staticmethod
    @abstractmethod
```

[1] https://github.com/DYCI2/Somax2/blob/913192ce0dbeef634deaf0b734321ccb5e7975cc/python/somax/somax/corpus_builder/manual_text_formats.py

```
26      def keyword() -> str:
27          """ """
28
29      @classmethod
30      @abstractmethod
31      def parse_file(cls,
32                     analysis_file_path: str,
33                     use_tempo_annotations: bool = False,
34                     pre_analysed_descriptors: Optional[List[
    Type[CorpusFeature]]] = None,
35                     ignore_invalid_lines: bool = False) ->
    Tuple[List[float], List[Optional[float]]]:
36          """ raises: RuntimeError if parsing failed """
37
38      @staticmethod
39      @abstractmethod
40      def format_line(onset: float, duration: float, features:
    List[CorpusFeature]) -> str:
41          """ returns a single line to write to a file """
42
43      @staticmethod
44      def keywords() -> List[str]:
45          """ raises: KeyError """
46          return [cls.keyword() for cls in TextFormat.
    _introspect().values()]
47
48      @staticmethod
49      def _introspect() -> Dict[str, Type['TextFormat']]:
50          return dict(inspect.getmembers(sys.modules[__name__],
51                                         lambda member: inspect
    .isclass(member) and
52
    issubclass(member, TextFormat) and
53                                                        not
    inspect.isabstract(member) and
54                                                        (member
    .__module__ == __name__  # static classes
55                                                         or (
    member.__module__ == 'abc'
56
    and not member.__name__ == 'ABC'))))  # dynamic classes
57
58      @staticmethod
59      def from_keyword(keyword: str) -> Type['TextFormat']:
60          """ raises: KeyError """
61          return {cls.keyword().lower(): cls for cls in
    TextFormat._introspect().values()}[keyword.lower()]
62
63      @staticmethod
```

```
64      def default() -> str:
65          return SoundStudio.keyword()
66
67
68  class UniformTextFormat(TextFormat, ABC):
69
70      @staticmethod
71      @abstractmethod
72      def parse_line(line_str: str,
73                     keys: List[Type[CorpusFeature]]) -> Tuple[
    float, Optional[float], Dict[Type[CorpusFeature], Any]]:
74          """ returns: onset, offset, feature_dict """
75
76      @classmethod
77      def parse_file(cls,
78                     analysis_file_path: str,
79                     use_tempo_annotations: bool = False,
80                     pre_analysed_descriptors: Optional[List[
    Type[CorpusFeature]]] = None,
81                     ignore_invalid_lines: bool = False
82                     ) -> Tuple[List[float], List[Optional[
    float]]]:
83          with open(analysis_file_path, 'r') as f:
84              onsets: List[float] = []
85              offsets: List[float] = []
86              for i, line in enumerate(f):  # type: int, str
87                  if use_tempo_annotations and re.match(
    Constants.TEMPO, line, flags=re.IGNORECASE):
88                      raise NotImplementedError("Tempo is not
    supported yet")
89                  if re.match(Constants.EMPTY, line):
90                      logging.debug(f"Line {i + 1}: Ignoring
    empty line")
91                  else:
92                      try:
93                          onset_s: float
94                          offset_s: Optional[float]
95                          descriptor_dict: Dict[Type[
    CorpusFeature], Any]
96                          onset_s, offset_s, descriptor_dict =
    cls.parse_line(line,
97
                                       keys=pre_analysed_descriptors)
98                      except ParsingError as e:
99                          err_msg: str = f"invalid line {i +
    1}: '{str(e)}'"
100                         if ignore_invalid_lines:
101                             logging.warning(err_msg)
102                             continue
```

```
                        else:
                            raise RuntimeError(err_msg)

                    onsets.append(onset_s)
                    offsets.append(offset_s)

        return onsets, offsets


class ProTools(TextFormat):

    @staticmethod
    def keyword() -> str:
        return ProTools.__name__

    @classmethod
    def parse_file(cls,
                   analysis_file_path: str,
                   use_tempo_annotations: bool = False,
                   pre_analysed_descriptors: Optional[List[Type[CorpusFeature]]] = None,
                   ignore_invalid_lines: bool = False
                   ) -> Tuple[List[float], List[Optional[float]]]:
        with open(analysis_file_path, 'r') as f:
            sample_rate: Optional[float] = None
            while sample_rate is None:
                try:
                    sample_rate = cls.parse_sample_rate(next(f))
                    if sample_rate is not None and sample_rate <= 0:
                        raise RuntimeError(f"Invalid sample rate: {sample_rate}")

                except StopIteration as e:
                    # End of file without successfully parsing any sample rate
                    raise RuntimeError from e

            found_markers: bool = False
            pattern: re.Pattern[str] = re.compile("M A R K E R S\s\sL I S T I N G")
            while not found_markers:
                try:
                    found_markers = bool(re.match(pattern, next(f)))
                except StopIteration as e:
```

```
143                     # End of file without successfully
    parsing markers listing header
144                     raise RuntimeError from e
145
146             # Ignore headers of markers list
147             next(f)
148
149             onsets: List[float] = []
150             offsets: List[Optional[float]] = []
151
152             reached_end_of_markers: bool = False
153             while not reached_end_of_markers:
154                 try:
155                     onset: Optional[int] = cls.parse_line(
    next(f))
156                     if onset is None:
157                         reached_end_of_markers = True
158                     else:
159                         onsets.append(onset / sample_rate)
160                         offsets.append(None)
161
162                 except ParsingError as e:
163                     raise RuntimeError from e
164                 except StopIteration:
165                     reached_end_of_markers = True
166
167         return onsets, offsets
168
169     @staticmethod
170     def format_line(onset: float, duration: float, features:
    List[CorpusFeature]) -> str:
171         raise RuntimeError("Not implemented")
172
173     @staticmethod
174     def parse_line(line: str) -> Optional[int]:
175         if re.match(Constants.EMPTY, line):
176             logging.debug("Ignoring empty line")
177             return None
178
179         pattern: re.Pattern[str] = re.compile(f"^\\s*\\d+\\s
    +(?:\\d+|\\d+:\\d+)\\s+(\\d+)")
180         tokens = re.match(pattern, line)
181
182         if tokens is None:
183             raise ParsingError(f"Invalid format: {line}")
184
185         return int(tokens.group(1))
186
187     @staticmethod
```

```
188     def parse_sample_rate(line: str) -> Optional[float]:
189         pattern: re.Pattern[str] = re.compile(f"SAMPLE RATE
    :\\s*({Constants.FLOAT})")
190         tokens = re.match(pattern, line)
191 
192         return None if tokens is None else float(tokens.group
    (1))
193 
194 
195 class Reaper(TextFormat):
196 
197     @staticmethod
198     def keyword() -> str:
199         return Reaper.__name__
200 
201     @classmethod
202     def parse_file(cls,
203                    analysis_file_path: str,
204                    use_tempo_annotations: bool = False,
205                    pre_analysed_descriptors: Optional[List[
    Type[CorpusFeature]]] = None,
206                    ignore_invalid_lines: bool = False
207                    ) -> Tuple[List[float], List[Optional[
    float]]]:
208         with open(analysis_file_path, 'r') as f:
209 
210             found_markers: bool = False
211             pattern: re.Pattern[str] = re.compile("    #
                                Name          Start")
212             while not found_markers:
213                 try:
214                     found_markers = bool(re.match(pattern,
    next(f)))
215                 except StopIteration as e:
216                     # End of file without successfully
    parsing markers listing header
217                     raise RuntimeError from e
218 
219             # Ignore headers of markers list
220             next(f)
221 
222             onsets: List[float] = []
223             offsets: List[Optional[float]] = []
224 
225             reached_end_of_markers: bool = False
226             while not reached_end_of_markers:
227                 try:
228                     onset: Optional[float] = cls.parse_line(
    next(f))
```

```
                    if onset is None:
                        reached_end_of_markers = True
                    else:
                        onsets.append(onset)
                        offsets.append(None)

                except ParsingError as e:
                    raise RuntimeError from e
                except StopIteration:
                    reached_end_of_markers = True

        return onsets, offsets

    @staticmethod
    def format_line(onset: float, duration: float, features:
    List[CorpusFeature]) -> str:
        raise RuntimeError("Not implemented")

    @staticmethod
    def parse_line(line: str) -> Optional[float]:
        if re.match(Constants.EMPTY, line):
            logging.debug("Ignoring empty line")
            return None

        pattern: re.Pattern[str] = re.compile(f"^\\s*.\\d
    +(?:\\d+|\\s*|.+)\\s+({Constants.FLOAT})")
        tokens = re.match(pattern, line)

        if tokens is None:
            raise ParsingError(f"Invalid format: {line}")

        return float(tokens.group(1))


class SoundStudio(UniformTextFormat):
    @staticmethod
    def keyword() -> str:
        return SoundStudio.__name__

    @staticmethod
    def parse_line(line_str: str,
                   keys: List[Type[CorpusFeature]]) -> Tuple[
    float, Optional[float], Dict[Type[CorpusFeature], Any]]:
        """ format: <ONSET>\n
                    <ONSET>\n
                    ...

            where ONSET = mm'ss,ffff
        """
```

```
275         tokens = re.match("\s*(\d+)'(\d+),(\d+)\s*", line_str
    )
276         if tokens is None:
277             raise ParsingError(line_str)
278 
279         try:
280             onset: float = int(tokens.group(1)) * 60 + int(
    tokens.group(2)) + 0.0001 * int(tokens.group(3))
281             descriptors = {}
282             return onset, None, descriptors
283         except IndexError:
284             raise ParsingError(line_str)
285 
286     @staticmethod
287     def format_line(onset: float, duration: float, features:
    List[CorpusFeature]) -> str:
288         minutes, seconds = divmod(onset, 60)  # type: float,
    float
289         seconds, hundreds = divmod(seconds, 1)  # type: float
    , float
290         return f"{int(minutes)}'{int(seconds):0>2},{int(
    hundreds * 1000):0<4}\n"
291 
292 
293 class Audacity(UniformTextFormat):
294     REGEX = re.compile(f"\\s*({Constants.FLOAT})\\s({
    Constants.FLOAT}).*")
295 
296     @staticmethod
297     def keyword() -> str:
298         return Audacity.__name__
299 
300     @staticmethod
301     def parse_line(line_str: str,
302                    keys: List[Type[CorpusFeature]]) -> Tuple[
    float, Optional[float], Dict[Type[CorpusFeature], Any]]:
303         """ format:
304                 <LINE>\n
305                 <LINE>\n
306                 ...
307 
308             where LINE = <ONSET>\t<OFFSET>\t[<MARKER_NAME>]
309                   ONSET       = float
310                   OFFSET      = float
311                   MARKER_NAME = string | <empty>
312 
313             Note that in the case where ONSET     OFFSET,
    OFFSET will be parsed as the next marker's onset
314 
```

```
315             """
316         if len(keys) > 0:
317             raise RuntimeError(f"Format '{Audacity.keyword()
    }' does not support manual descriptors")
318         tokens = re.match(Audacity.REGEX, line_str)
319         if tokens is None:
320             raise ParsingError(line_str)
321 
322         try:
323             onset: float = float(tokens.group(1))
324             offset: Optional[float] = float(tokens.group(2))
325             if offset - onset < 0.01:  # 10 ms
326                 offset = None
327             descriptors = {}
328             return onset, offset, descriptors
329         except IndexError:
330             raise ParsingError(line_str)
331 
332     @staticmethod
333     def format_line(onset: float, duration: float, features:
    List[CorpusFeature]) -> str:
334         return f"{onset:.3f}\t{onset + duration:3f}\n"
```

Appendix B

Additional Beat Tracker Figures

In this appendix are presented additional figures addressing the implemented devices and tools following the research described in chapter 3. These are the complex spectral difference descriptor implemented in the gen~ environment of Max/MSP, the patch used to compute the onset detection from it and the patch showing the onset, tempo and beat detecting object implemented with the extensions described in the same chapter.

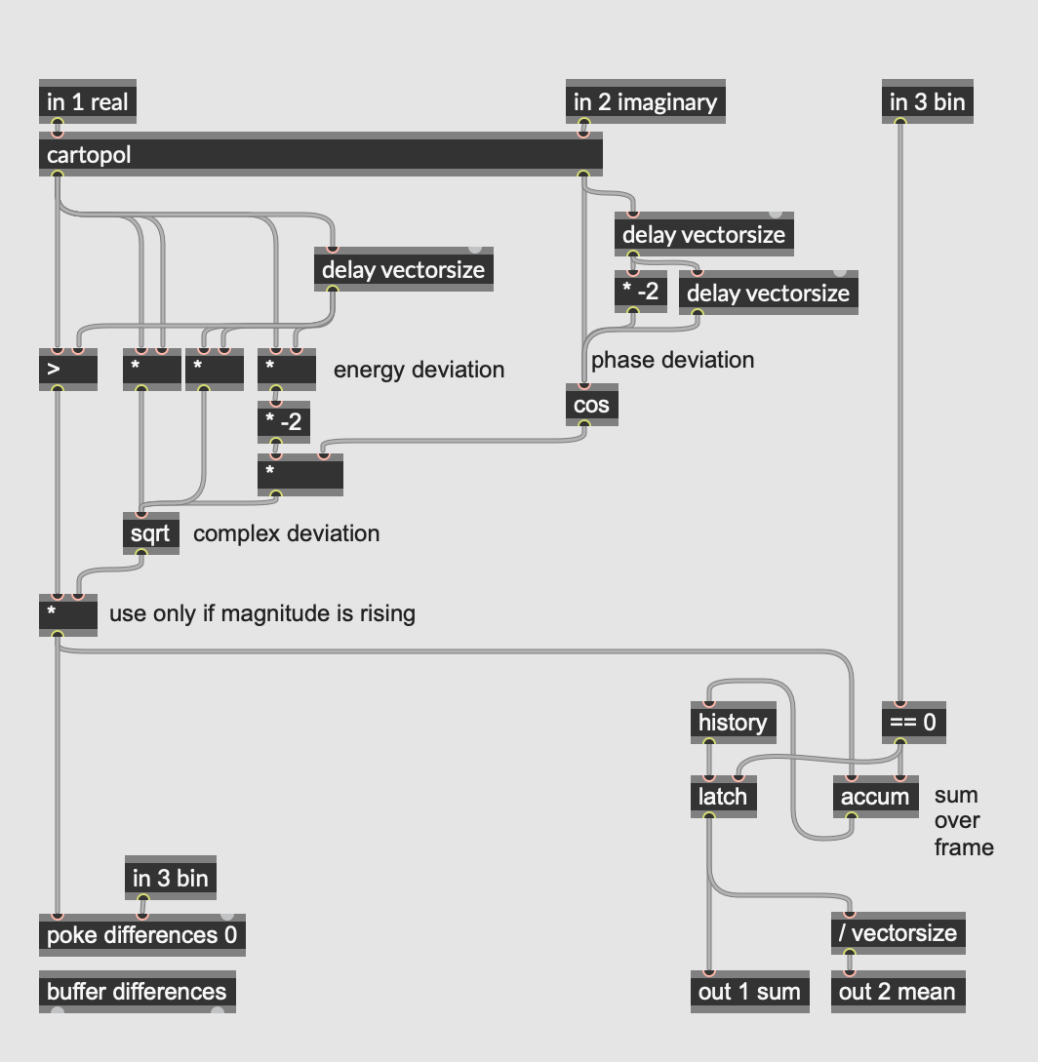


Figure B.1: gen~ implementation of a complex spectral difference descriptors, as described in [31] and [47]. Help and support in the gen~ environment has been kindly given by Graham Wakefield, developer of gen~ and author of [53].

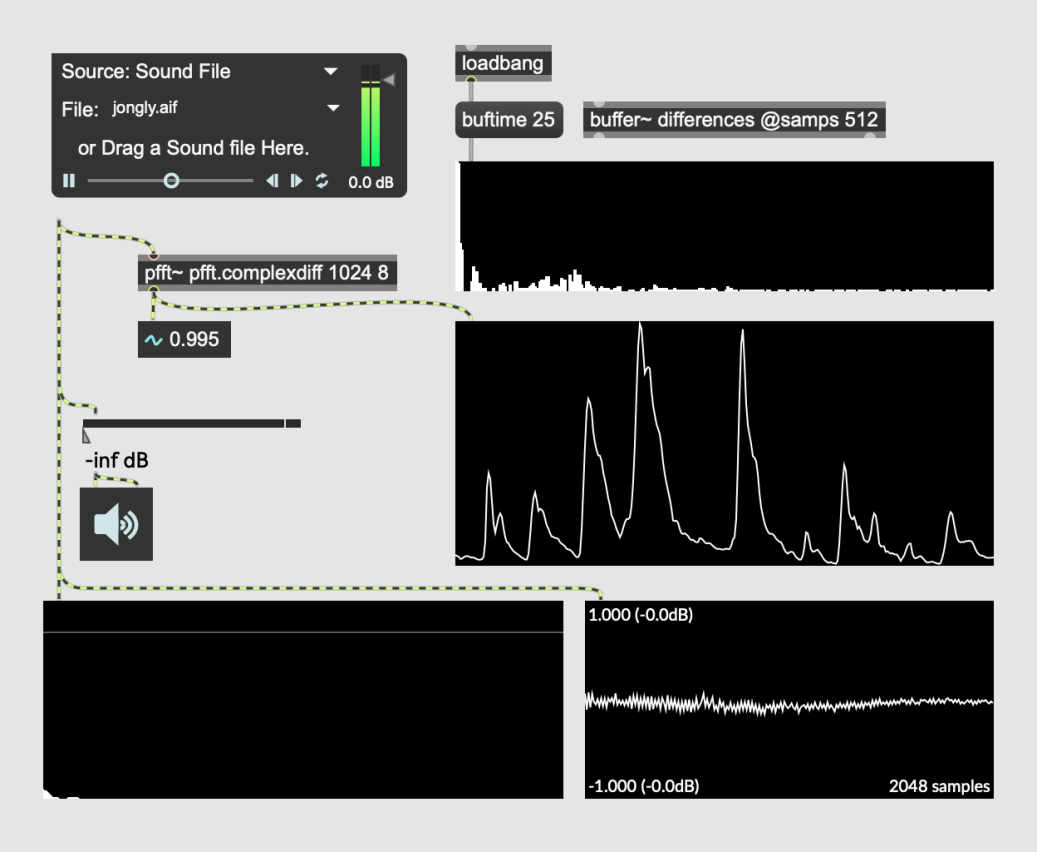


Figure B.2: Max/MSP patch used to compute onset detection of an audio signal, throught the complex spectral difference descriptor shown in Figure B.1.

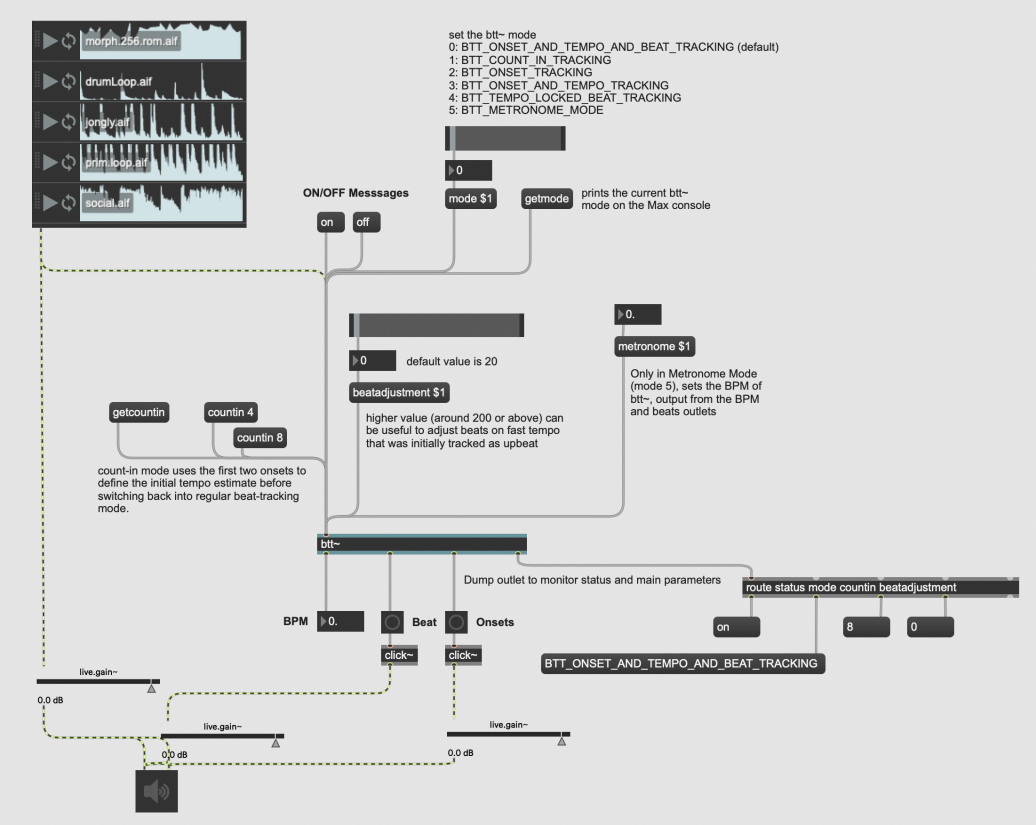


Figure B.3: New Max/MSP implementation of the presented beat tracker with added features and modes. These include different configurations of onset, tempo and beat detection, countin, beat prediction adjustment and metronome mode.

Acknowledgment

I would like to thank Gérard Assayag for his supervision and the fantastic reception in the Music Representations team, and Stefania Serafin for her kindness and continuous support. Thanks to Joakim Borg, Mikhail Malt, Manuel Poletti, Georges Bloch, Pierre Saint-Germier, Clément Canonne, as well as to all the great researchers I've met at IRCAM, without whom my work would have not been the same. Thanks to Lorenzo Colombo, Joëlle Léandre and Horse Lords for their extraordinary music contributions to this project and the moments spent together during our artistic research.

Thanks to Anaïs for always being here.

References

[1] I. Xenakis, *Formalized music: thought and mathematics in composition*. Pendragon Press, 1992, no. 6.

[2] G. E. Lewis, "Too many notes: Computers, Complexity and Culture in Voyager," *Leonardo Music Journal*, vol. 10, pp. 33–39, 2000.

[3] V. R. Mogensen, "Evaluating a swarm algorithm as an improvising accompanist: an experiment in using transformed analysis of George E. Lewis's Voyager," *Conference: Conference on Computer Simulation of Musical Creativity (CSMC16)*, 2016.

[4] "IRCAM." [Online]. Available: https://www.ircam.fr/

[5] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer-assisted composition at ircam: From patchwork to openmusic," *Computer Music Journal*, vol. 23, no. 3, pp. 59–72, 1999.

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

[7] A. Agostini and D. Ghisi, "Bach: An environment for computer-aided composition in max," in *ICMC*, 2012.

[8] "Cycling74 - Max," https://cycling74.com/products/max/, accessed: 2022-01-31.

[9] G. Assayag, G. Bloch, M. Chemillier, A. Cont, and S. Dubnov, "Omax brothers: A dynamic yopology of agents for improvization learning," *Proceedings of the ACM International Multimedia Conference and Exhibition*, pp. 125–132, 2006.

[10] M. Toro, C. Agon, C. Rueda, and G. Assayag, "GELISP: A framework to represent musical constraint satisfaction problems and search strategies," *Journal of Theoretical and Applied Information Technology*, vol. 86, no. 2, p. 327–331, 2016.

[11] L. A. Hiller Jr and L. M. Isaacson, "Musical composition with a high speed digital computer," in *Audio Engineering Society Convention 9*. Audio Engineering Society, 1957.

[12] S. Luque, "The stochastic synthesis of iannis xenakis," *Leonardo Music Journal*, vol. 19, pp. 77–84, 2009.

[13] S. Dubnov, "Stylistic randomness: About composing NTrope Suite," *Organised Sound*, vol. 4, no. 2, p. 87–92, 1999.

[14] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor oracle: a new structure for pattern matching," *26th Seminar on Current Trends in Theory and Practice of Informatics (SOFSEM'99)*, pp. 291–306, 1999.

[15] G. Assayag and S. Dubnov, "Using factor oracles for machine improvisation," *Soft Computing*, vol. 8, pp. 604–610, 2004.

[16] B. Lévy, G. Bloch, and G. Assayag, "Omax dialectics," *New Interfaces for Musical Expression*, pp. 137–140, 2012.

[17] L. Bonnasse-Gahot, "Prototype de logiciel d'harmonisation/arrangement à la volée: Somax v0," *STMS IRCAM, Internal Report*, 2012.

[18] ——, "An update on the somax project," *STMS IRCAM, Internal Report*, 2014.

[19] J. Borg, "Somax 2: A real-time framework for human-machine improvisation," *STMS IRCAM & Aalborg University, Internal Student Report*, 2019.

[20] J. Nika, M. Chemillier, and G. Assayag, "Improtek: introducing scenarios into human-computer music improvisation," *Computers in Entertainment*, vol. 14, pp. 1–27, 2017.

[21] J. Nika and T. Carsault, "Mapping and clustering on audio descriptors spaces for dyci2," *STMS IRCAM, Internal Student Report*, 2022.

[22] J. Nika, K. Déguernel, A. Chemla, E. Vincent, and G. Assayag, "Dyci2 agents: merging the "free", "reactive" and "scenario-based" music generation paradigms," *International Computer Music Conference, Shangai, China*, 2017.

[23] "REACH: Raising Co-creativity in Cyber-Human Musicianship." [Online]. Available: http://repmus.ircam.fr/reach

[24] "STMS Lab." [Online]. Available: https://www.stms-lab.fr/team/representations-musicales

[25] "La co-créativité musicale entre humain et machine : ERC Advanced Grant de Gérard Assayag." [Online]. Available: https://www.ins2i.cnrs.fr/fr/cnrsinfo/la-co-creativite-musicale-entre-humain-et-machine-erc-advanced-grant-de-gerard-assayag

[26] "SoMax2 Project Page." [Online]. Available: http://repmus.ircam.fr/somax2

[27] M. Fiorini, "Somax 2 User's Guide," *STMS IRCAM, Documentation Report*, 2023.

[28] J. Borg, "The Somax 2 Theoretical Model," *STMS IRCAM, Technical Report*, 2021.

[29] "Max/MSP, Cycling '74." [Online]. Available: https://cycling74.com/

[30] J. Borg, "The Somax 2 Developer's Documentation," *STMS IRCAM, Technical Report*, 2023.

[31] A. M. Stark, "Musicians and machines: Bridging the Semantic Gap In Live Performance," *PhD Thesis, Centre for Digital Music, Department of Electronic Engineering and Computer Science, Queen Mary, University of London*, 2011.

[32] P. Grosche and C. S. Sapp, "What makes beat tracking difficult? a case study on chopin mazurkas," *Proceedings of the 11th International Society for Music Information Retrieval Conference*, 2010.

[33] M. Goto, "An audio-based real-time beat tracking system for music with or without drum-sounds," *International Journal of Phytoremediation*, vol. 21, pp. 159–171, 2001.

[34] D. P. Ellis, "Beat tracking by dynamic programming," *Journal of New Music Research*, vol. 36, no. 1, pp. 51–60, 2007.

[35] M. E. Davies and M. D. Plumbley, "Context-dependent beat tracking of musical audio," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, pp. 1009–1020, 3 2007.

[36] E. Large and J. Kolen, "Resonance and the perception of musical meter," *Connection Science*, vol. 6, pp. 177–208, 1994.

[37] P. Toiviainen, "An interactive midi accompanist." *Computer Music Journal*, vol. 22, pp. 63–75, 1998.

[38] E. Large, "Beat tracking with a nonlinear oscillator," *In Working Notes of the IJCAI-95 Workshop on Artificial Intelligence and Music*, vol. 24031, 1995.

[39] E. Large and M. Jones, "The dynamics of attending : How people track time-varying events," *Psychological Review*, vol. 106, pp. 119–159, 1999.

[40] E. Large, "On synchronizing movements to music," *Human Movement Science*, vol. 19, pp. 524–566, 2000.

[41] E. Large and C. Palmer, "Perceiving temporal regularity in music," *Cognitive Science*, vol. 26, pp. 1–37, 2002.

[42] "IrcamBeat Vamp Plugin — Ircam Forum." [Online]. Available: https://forum.ircam.fr/projects/detail/ircambeat-vamp-plugin/

[43] "Partiels — Ircam Forum." [Online]. Available: https://forum.ircam.fr/projects/detail/partiels/

[44] L. Bonnasse-Gahot, "Donner à omax le sens du rythme : vers une improvisation plus riche avec la machine," *École des Hautes Études en sciences sociales, Internal Report*, 2010.

[45] J. Borg, "The Somax 2 Software Architecture Rev. 0.2.0," *STMS IRCAM, Technical Report*, 2021.

[46] A. M. Stark, M. E. P. Davies, and M. D. Plumbley, "Real-time beat-synchronous analysis of musical audio," *Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx-09), Como, Italy, September 1-4*, 2009.

[47] J. P. Bello, C. Duxbury, M. Davies, and M. Sandler, "On the use of phase and energy for musical onset detection in the complex domain," pp. 553–556, 6 2004.

[48] M. F. McKinney, D. Moelants, M. E. Davies, and A. Klapuri, "Evaluation of audio beat tracking and music tempo extraction algorithms," *Journal of New Music Research*, vol. 36, pp. 1–16, 3 2007.

[49] A. P. Klapuri, A. J. Eronen, and J. T. Astola, "Analysis of the meter of acoustic musical signals," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, pp. 342–355, 1 2006.

[50] S. Hainsworth, "Techniques for the Automated Analysis of Musical Audio," *PhD thesis, University of Cambridge*, 2004.

[51] "GNU General Public License." [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html

[52] "XCode." [Online]. Available: https://developer.apple.com/xcode/

[53] G. Wakefield and G. Taylor, "Generating sound and organizing time — cycling '74." [Online]. Available: https://cycling74.com/books/go

[54] "Shaper." [Online]. Available: https://apps.apple.com/us/app/shaper-synthesizer/id1536713443

[55] "OWL Modular Eurorack." [Online]. Available: https://www.rebeltech.org/product/owl-modular/

[56] "Max and MOD Duo." [Online]. Available: https://cycling74.com/feature/modduo

[57] M. Krzyżaniak, "Musical Robot Swarms and Equilibria," *University of Oslo*, 2020.

[58] Arturia, "Minifreak - Spontaneous algorithmic synthesizer." [Online]. Available: https://www.arturia.com/products/hardware-synths/minifreak/overview

[59] J. O. Smith, "Physical modeling using digital waveguides," *Computer Music Journal*, vol. 16, p. 74, 24 1992.

[60] P. M. Morse, "Vibration and sound," 1936.

[61] S. D. Bilbao, *Numerical sound synthesis : finite difference schemes and simulation in musical acoustics*. Wiley Publishing, 2009.

[62] S. Willemsen, *The Emulated Ensemble, Real-Time Simulation of Musical Instruments using Finite-Difference Time-Domain Methods*. Aalborg Universitet, 2021.

[63] K. Karplus and A. Strong, "Digital synthesis of plucked string and drum timbres," *Computer Music Journal*, vol. 2, no. 7, pp. 43–55, 1983.

[64] K. Karplus, A. Strong, and J. O. Smith, "Extensions of the karplus-strong plucked string algorithm," *Computer Music Journal*, vol. 2, no. 7, pp. 56–69, 1983.

[65] H. Leung, "Regular Languages and Finite Automata," *New Mexico State University*, 2010.

[66] J. Harley, "Iannis Xenakis Komboï, for percussion & harpsichord," *Rovi Corporation*, 2014.

[67] I. Xenakis, "La voie de la recherche et de la question," *Preuves 177*, 1965.

[68] ——, "Vers un métamusique," *La Nef 29*, 1967.

[69] ——, "Sieves," *Perspectives of New Music*, vol. 28, no. 1, 1990.

[70] N. Geraseni, *Pythagorei Introductionis arithmeticae.* Hoche, Richard, ed., 1866, vol. 2.

[71] D. Exarchos, "Iannis xenakis and Sieve Theory - An Analysis of the Late Music (1984-1993)," *PhD Thesis, Goldsmiths College, University of London*, 2007.

[72] C. Ariza, "The Xenakis Sieve as Object: A New Model and a Complete Implementation," *Computer Music Journal*, vol. 29, no. 2, pp. 40–60, 2005.

[73] "Jöelle Léandre - official website." [Online]. Available: https://www.joelle-leandre.com/

[74] "Jöelle Léandre - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Jo%C3%ABlle_L%C3%A9andre

[75] I. S. Dhillon and S. Sra, "Generalized nonnegative matrix approximations with bregman divergences," *NIPS'05: Proceedings of the 18th International Conference on Neural Information Processing Systems*, pp. 283–290, 2005.

[76] R. Tandon and S. Sra, "Sparse nonnegative matrix approximation: new formulations and algorithms sparse nonnegative matrix approximation:

new formulations and algorithms sparse nonnegative matrix approximation: new formulations and algorithms," *Max Planck Institute for Biological Cybernetics, Technical Report*, 2010.

[77] J. Borg, "A Gentle Introduction to Somax," *STMS IRCAM, Tutorial Report*, 2020.

[78] M. M. Wanderley and P. Depalle, "Gestural control of sound synthesis," *Proceedings of the IEEE*, vol. 92, pp. 632–644, 2004.

[79] B. Feldman, "Improving the latent harmonic space of SOMax with Variational Autoencoders," *Centrale Supelec & STMS IRCAM, Internal Student Report*, 2021.

[80] A. Hunt and M. M. Wanderley, "Mapping performer parameters to synthesis engines," *Organised Sound*, vol. 7, pp. 97–108, 2002.

[81] M. Lee and D. Wessel, "Connectionist models for real-time control of synthesis and compositional algorithms." *Proc. of the 1992 Int. Computer Music Conference*, pp. 277–280, 1992.

[82] G. Garnett and C. Goudeseune, "Performance factors in control of high-dimensional spaces," *Proc. of the 1999 Int. Computer Music Conf.*, pp. 268–271, 1999.

[83] A. Mulder, S. Fels, and K. Mase, "Empty-handed gesture analysis in max/fts," *Kansei, The Technology of Emotion. Proc. of the AIMI Int. Workshop*, 1997.

[84] A. Hunt, M. M. Wanderley, and M. Paradis, "The importance of parameter mapping in electronic instrument design," *International Journal of Phytoremediation*, vol. 21, pp. 429–440, 2003.

[85] T. Golvet, L. Goupil, P. Saint-Germier, B. Matuszewski, G. Assayag, J. Nika, and C. Canonne, "With, against, or without? familiarity and copresence increase interactional dissensus and relational plasticity in freely improvising duos." *Psychology of Aesthetics, Creativity, and the Arts*, 2021.

[86] P. Saint-Germier and C. Canonne, "Coordinating free improvisation: An integrative framework for the study of collective improvisation," *Musicae Scientiae*, 2020.

[87] J. Nika and J. Bresson, "Composing Structured Music Generation Processes with Creative Agents," *2nd Joint Conference on AI Music Creativity (AIMC), Graz, Austria*, 2021.

[88] J. Nika, "Guiding human-computer music improvisation: introducing authoring and control with temporal scenarios," *PhD Thesis, Université Pierre et Marie Curie - Paris VI*, 2016.

[89] M. Malt and M. Fiorini, "News on Somax." [Online]. Available: https://forum.ircam.fr/article/detail/news-on-somax-mikhail-malt-marco-fiorini/