



**THÈSE DE DOCTORAT DE
L'UNIVERSITÉ PIERRE ET MARIE CURIE**

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Benjamin LÉVY

Pour obtenir le grade de

DOCTEUR de L'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

**Principes et architectures pour un système interactif et
agnostique dédié à l'improvisation musicale**

Tim BLACKWELL	PR Goldsmiths University of London	Rapporteur
Andrew BROWN	PR Griffith University	Rapporteur

soutenue le 16 décembre 2013 devant le jury composé de :

Gérard ASSAYAG	DR Ircam	Directeur de thèse
Tim BLACKWELL	PR Goldsmiths University of London	Rapporteur
Carlos AGON	PR UPMC	Examineur
Clément CANONNE	MdC Université de Bourgogne	Examineur
Shlomo DUBNOV	PR U.C. San Diego	Examineur
Robert ROWE	PR New York University	Examineur
David WESSEL	PR U.C. Berkeley	Examineur

Principles and Architectures
for an Interactive and Agnostic Music Improvisation System

by
Benjamin Lévy

Thesis submitted in satisfaction for the degree of
Doctor of Philosophy
in
Computer Science
from
University Pierre and Marie Curie

Gérard ASSAYAG	Ircam	Supervisor
Tim BLACKWELL	Goldsmiths, University of London	Reporter
Andrew BROWN	Griffith University	Reporter
Carlos AGON	UPMC	Examiner
Clément CANONNE	Université de Bourgogne	Examiner
Shlomo DUBNOV	UC San Diego	Examiner
Robert ROWE	New York University	Examiner
David WESSEL	UC Berkeley	Examiner

Abstract

Since the beginnings of computer science, computers have been used fruitfully for the generation of new music. In the last decades, their utilization to create generative systems moved from the implementation of tools for composition towards the invention of reactive programs aimed at participating in the much more unpredictable and challenging situation of musical improvisation. The work presented in this thesis focuses on the conception and realization of such a software, capable of pertinent interaction with acoustic musicians in a collective free improvisation, that is an improvisation without any predetermined knowledge of structures, rules or style. It is extended at the end of our work with considerations on emerging properties such as pulse or a broad notion of harmony. The OMax project proposes to approach the problem of non-idiomatic improvisation by learning and mimicking the style of a musician with an agnostic and incremental knowledge model. We take this computer system as our work basis and examine carefully three aspects: the conceptual principles of the system, the software architectures for effective implementations and the *real-life* usage of this system in numerous testing and concerts situations.

Besides a thorough study of all the conceptual elements of the system based on anthropomorphic decomposition of its parts, our main contribution is the design and realization of several variations of the OMax system. We first propose to use dual structures to store the literal information extracted from the input stream of a musician and to hold the knowledge model built on this information. We illustrate this architecture with a novel real-time visualization of the model. We claim that duplicating all the processes that lead up to the building of the knowledge model enables the computer system to listen at once to several aspects of the ongoing music with their own temporal logics captured in the different model instances. Running this way a multiple descriptions and multiple inputs modeling of the on going musical content greatly improves the pertinence of the computers response. The study of the generation mechanisms of the system enables us to put forward a new database oriented approach to the collection of these models. Our work has been also strongly coupled with the testing of our prototypes with several leading musicians. The musical feedback gathered along these numerous musical experiences lead the work presented in this thesis and opens up many directions for further research on the system, notably the exploration of automatic decisions based on the pertinence of the different descriptions at a given moment of the improvisation or a larger use of prepared material to a “composed improvisation” perspective.

Résumé

Depuis les débuts de l'informatique, les ordinateurs ont été utilisés fructueusement pour générer de nouvelles musiques. Au cours des dernières décennies, l'utilisation de systèmes génératifs s'est progressivement tournée des outils pour la composition vers l'invention de programmes réactifs pouvant participer à une improvisation musicale, situation bien plus imprévisible et stimulante. Le travail présenté dans cette thèse se concentre sur la conception et la réalisation d'un tel système informatique, capable d'interagir musicalement et pertinemment avec des musiciens acoustiques dans le cadre de l'improvisation libre collective, c'est à dire de l'improvisation détachée de toute structures, règles ou style prédéfinis. Nous étendrons ce cadre à la fin de notre travail en y intégrant l'utilisation de propriétés émergentes telles que la pulsation ou une notion large d'harmonie. Le projet OMax propose d'aborder le problème de l'improvisation non-idiomatique par l'apprentissage et l'imitation à la volée du style d'un musicien à l'aide d'un modèle de connaissance agnostique. Ce système sert de base à notre travail et nous en examinons attentivement trois aspects : les principes conceptuels du système, les architectures logicielles permettant une implémentation efficace, et l'usage réel du système dans de nombreux tests et concerts.

Outre une étude fouillée de tous les éléments théoriques du système suivant une décomposition anthropomorphique de ses différentes parties, les contributions principales du travail présenté dans cette thèse sont la conception et la réalisation de plusieurs nouvelles versions du système OMax. Nous proposons en premier lieu l'utilisation de structures duales pour contenir d'une part les informations extraites du flux d'entrée d'un musicien et d'autre part le modèle de connaissance construit sur ces informations. Nous illustrons cette architecture avec une nouvelle visualisation en temps-réel du modèle de connaissance. Nous prétendons que la multiplication de tous les processus menant à la construction du modèle de connaissance permet au système d'écouter en parallèle plusieurs aspects de la musique se déroulant. Chacun de ces aspects possédant sa propre logique temporelle mène à la construction d'une instance différente du modèle. Nous obtenons ainsi une modélisation du discours musical basée sur plusieurs descriptions de plusieurs entrées ce qui permet d'augmenter considérablement la pertinence de la réponse de l'ordinateur. L'étude menée sur le mécanisme de génération du système nous permet de proposer une nouvelle approche de la collection de modèles de connaissance vue comme une base de donnée. Notre travail a été fortement associé à des tests réguliers des prototypes du système avec de nombreux musiciens de premier plan. Les réactions collectées au cours de ces nombreuses expériences musicales ont fortement orienté le travail présenté dans cette thèse et ont ouvert de nouvelles directions de recherche autour de ce système, notamment l'exploration de la prise automatique de décisions éclairée par la pertinence des différentes descriptions du flux musical au cours de l'improvisation ou une plus grande utilisation de matériau musical préparé dans le cas d'*improvisations composées*.

Contents

Abstract	i
Résumé	iii
Contents	v
List of Figures	ix
1 Introduction	1
1.1 Thesis Focus	1
1.2 Motivations	2
1.3 Thesis Organisation	2
I Generative Systems	5
2 Operational Approach	9
2.1 The <i>Time-Scale</i> Axis	9
2.2 The <i>Learning</i> Axis	11
2.3 The <i>Control</i> Axis	13
2.4 Operational Plot	14
3 Behavioral Reflexions	21
3.1 Improvisation Systems	21
3.2 Evaluation	24
II Principles of an Improvisation System	27
4 General Principles	29
4.1 History	29
4.2 OMax Principles	30
4.2.1 Phenomenological Description	30
4.2.2 Operational Hypothesis	31

4.2.3	Functional Overview	32
4.3	Cognitive Views	33
5	Perception	37
5.1	Listening	37
5.1.1	Inputs Anatomy	38
5.1.2	Extraction	42
5.2	Analysis	46
5.2.1	Segmentation	46
5.2.2	Timing Considerations	55
6	Knowledge	59
6.1	Alphabets	59
6.1.1	Predefined Alphabet	60
6.1.2	Evolutionary Labelling	61
6.1.3	Summary	64
6.2	Memory Model	66
6.2.1	Factor Oracle	66
6.2.2	Linear Skeleton	67
6.2.3	Suffix Links	67
6.2.4	Forward Links	69
6.3	Enriching Memory	70
6.3.1	Annotations	70
6.3.2	Virtual Pasts	73
6.3.3	Higher Structure Clustering	74
7	Performance	77
7.1	Production	77
7.1.1	Generation Principles	77
7.1.2	Rendering	80
7.2	Conducting	82
7.2.1	Voices	82
7.2.2	Content Shaping	84
III	Software Architecture	87
	Software Environment	89
8	Fluid Data Access	91
8.1	Data & Graph Structures	91
8.1.1	Low-Level Structures	92
8.1.2	Programming Interfaces	93
8.2	High Level Application: Visualization	94

9	Multiple Processing	99
9.1	Multiple Perceptions	100
9.1.1	Parallel Chains	100
9.1.2	Visualization	102
9.1.3	Duplication of Inputs	104
9.1.4	Centralization	104
9.2	Multiple Outputs	104
9.2.1	Generation Architecture	105
9.2.2	Polyphonic Extension	107
10	Database Approach	113
10.1	Knowledge Query	113
10.1.1	Independence of the Structures	114
10.1.2	Prominence of Medium	116
10.1.3	Knowledge Models as Database	117
10.2	Filtering & Weighting	119
10.2.1	Data Flow	119
10.2.2	Filtering	121
10.2.3	Weighting	122
11	Musical Experiences	125
11.1	Duets	125
11.2	Collective Free Improvisation	127
11.3	Jazz Band	128
11.4	Prepared Improvisations	130
11.5	Speech Improvisations	131
11.6	PedagOMax	133
12	Conclusion & Future Works	135
12.1	Conclusion	135
12.2	Perspectives	136
	List of Musical Experiences	139
	Bibliography	151
	Appendices	161
A	Algorithms	III
A.1	Mel Frequency Cepstrum Coefficients Computation	III
A.2	Factor Oracle Incremental Algorithm	IV

A.3	Factor Oracle Clustering Algorithm	V
A.4	Virtual Fundamental Computation Code	VII
A.5	C++ Classes	VIII
A.5.1	Data Structure	VIII
A.5.2	Graph Structure	VIII
B	Publications	XI
B.1	NIME 2012	XI
B.2	In Progress	XVI
C	Documentation of OMax Public Software	XXV

List of Figures

1	Plot of Generative Systems	15
2	General Principles of the OMax System	32
3	Cognitive Modelisation	34
4	Example of Different Streams Configurations for an Input	40
5	Diagram Summarizing the Anatomy of an Input	41
6	Anatomy of an Input with Context Consideration	42
7	Slicing of Polyphonic MIDI Stream	48
8	First Stage of Pitch Grouping: Statistical Analysis	50
9	Examples of Euclidean Distance Weighting Profiles for MFCCs	52
10	Clustering Process for Vector Descriptors	54
11	Example of Mutation in Vector Clustering	55
12	Illustration of the Timing Considerations and Date Computation	57
13	Representation of Clusters in 2 Dimension with their Center and Label	63
14	Description Chains up to the Labeling per Streams and Descriptions	65
15	Skeleton of the Factor Oracle Graph of the Sequence abaabacba	67
16	Suffix Links of the Factor Oracle Graph of the Sequence abaabacba	68
17	Complete Factor Oracle Graph of the Sequence abaabacba	69
18	Crossing Primary Extractions Information to Enrich Memory	72
19	An Example of Higher Level Clustering	76
20	Map of Patterns Constituted by the Suffix Links of an Abstract Sequence	79
21	Trees of Patterns Constituted by the Suffix Links of an Abstract Sequence	80
22	The Initial Collection of OMax's Externals for Max/MSP	93
23	Visualization of the Sound with Higher Level Information	94
24	Visualization of the Suffix Links with Colorful Representations	95
25	Visualization of a Small Generated Sequence with its Original Material	96
26	Visualization of the Improvisations with Read Head and Next Jump	97
27	Dual Descriptions Input Chain	101
28	Dual Descriptions Input Patch	102

29	Dual Descriptions Visualization	103
30	Generic Architecture of the Improvisation Logic	105
31	Specialization of the Improvisation Logic for Pitch and Spectra Descriptions	106
32	Generation Loop of with Improvisation Logic and Player	108
33	Summary of the First Multiple Architecture of the System	110
34	Parallel Segmentations Leading to Unaligned Models	115
35	Renovation of the Collection of OMax's Externals for Max/MSP	116
36	Overall Architecture of the Database Approach of the System	118
37	Diagram of the Database Approach of the Generation	120
38	Interface of the New Generation Part of the System	124
39	Inheritance Diagram of the Label Class	VIII
40	Collaboration Diagram of the Factor Oracle Structure Class	IX

Chapter 1

Introduction

Since the middle of the twentieth century, computers have been used to generate new music. From then, computers' usages have broaden to various music purposes including sound synthesis, sound processing, composition, improvisation and more. Along the time, researchers, engineers, musicians, composers have invented numerous systems more and more interactive to explore the relations between machine and music creation. Based on various principles, these system may produce sketches, assemble whole pieces, play unheard sounds or propose tools for real-time performance. Our research focuses on systems capable of direct interaction on the musical level, and we are especially interested in systems for musical improvisation. This field has been very active in the last years and several systems have been built and described to improvise with acoustic musicians. We base our research on this active background and examine the case of a system for improvisation with on-the-fly listening, learning and generating capacities. We studied and developed in our work the concepts and software architectures for the realization of such a system and took much care to test our research in *real-life* musical situations.

1.1 Thesis Focus

The starting point of our research is the stylistic reinjection principle extensively studied in the Musical Representations Team of IRCAM and associated to the OMax improvisation scheme. In the frame of this research field, we try to explore the global problem of creating an agnostic system for generative real-time improvisation. More than an in depth research on one of the numerous elements needed to build such a system, we adopt here a more distant view over the problem and attempt to follow through the extension of the conceptual principles needed for this agnostic, interactive improvising environment. We lead this research through cognitively-inspired conceptual architecture down to the very design and implementation of fully operational software and its thorough testing in challenging real-life concert situations involving interaction with world-class improvising musicians. The main musical objective we pursue in this work is to create a convincing musical partner for improvisation with rich propositions' capacities and much

musical *a propos*. We explore several conceptual and design approaches along this work. Some of these explorations have lead to deep renewal in the views and uses of the inner model for stylistic reinjection as well as considerations and rethinking of all the elements needed around this core to build up a functional system. Our work thus constitutes a decisive contribution to the establishment of this reinjection paradigm and to the software architectures allowing its existence in real-life concert situation.

1.2 Motivations

When discovering stylistic reinjection and the OMax early prototypes at the beginning of my PhD, I was amazed by its potential to engage a musical dialog with very various musicians and adapt to different styles. Stylistic reinjection seemed to me as a very powerful principle to question musical improvisation itself in its intimate balance between imitation and self-renewal. As I remembered my own experience in free improvisation as a cellist, I felt that the bending of pre-existing musical material is an important mechanism of collective free creation which is filled with personal reinterpretation of the others musical discourse fed into the common music matter. I decided then to give to this principles a strong conceptual foundation and pursue the achieving of a computer music system making use of this mechanism and capable of developing a pertinent musical discourse in a dense collective performance context as well as actively participating to an uncluttered improvisation in a duet with an acoustic musician. The playing with such a system as an interactive instrument — an active instrument capable of suggesting and developing musical directions by itself but also flexible and malleable enough to be conducted and played — has also been a very strong motivation in my work. The utopian goal of being capable to improvise in any musical context with this original instrument drove several of the explorations described in this thesis.

1.3 Thesis Organisation

We start this work with the presentation of three axes as a structure for the review of several generative systems which constitute the large framework of our research. We characterize these systems depending on the time-scale of the units they manipulate, their learning capabilities and their need for human control. Then we place our system in this landscape and discuss a few behavioral questions raised when trying to build a computer system dedicated to improvisation. Our goal is to build an interactive improvisation system with on the fly listening, learning and generative capabilities that makes a controllable generative instrument to improvise along with acoustic musicians.

Starting from the core principle of stylistic reinjection and the early prototypes of the OMax scheme, we thoroughly study all the conceptual aspects of such a system. We adopt a global and anthropomorphic view on the system and detail its perception, knowledge and performance parts. The perception of our system is based on the listening of melodic,

timbral, harmonic and rhythmical aspects of the input streams corresponding to the musical discourse of an acoustic musician. After the extraction of low-level information, we propose several segmentations to aggregate meaningful units for each description of the input. The graph we use to model the patterns of those units requires a strict classification thus we analyze the influence of this *alphabet* and propose several possibilities for such *symbolic labeling*. Then we present the Factor Oracle graph: it is an acyclic automaton built incrementally to recognize all the repeated patterns of a symbolic sequence. We detail the structure of this graph which serves as the core of our knowledge model and in particular the forest of suffix trees connecting all patterns. We propose several enriching of this knowledge model to gather in the memory of our system a powerful multi-descriptions representation of the musical material. Finally we explain how we navigate in this rich representation to generate new variations mimicking the style of the original material but which introduces novelty, thanks to pertinent recombinations. We also show how we render and can conduct musically these variations to create a consistent and controllable musical discourse.

The third part of this thesis focuses on the effective design of the system. We propose new low-level structures to build and hold this knowledge and illustrate the efficiency of these structures with a novel visualization of the knowledge model. Then we detail two software architectures we designed for this improvisation system. The first architecture places side by side the different descriptions learnt in the system and enables the parallel usage of these descriptions to develop variations on separated aspects of the original material. The second architecture adopts a novel view, oriented towards a database approach of the collection of descriptions and graphs constituting the knowledge of the system. With this architecture, we benefit from all the descriptions and model instances at once to enable the generation of hybrid variations, that is variations on several aspects (melodic, timbral, harmonic) of the original material in the same generated discourse. Through a filtering and weighting process, this architecture also enables to take advantage of several complementary information extracted from the original material (pulse, register etc.) and refine the musical conducting of the computer based generation.

Finally, we present in the last chapter of this thesis various situations in which we played with the system. We show how these numerous musical contexts help test and enhance the application of our research. This permanent confrontation with real-life music improvisation along leading musicians in studio sessions or concerts has been one of the most important drive of the work presented in this thesis.

Part I

Generative Systems

The huge number of computer systems imagined and used to generate music makes very long and tedious an exhaustive review of all of them. The system often mentioned as the first historical example of generative systems is the one which generated or *composed* the Illiac Suite in 1956. However, in this review we choose to focus on the (already numerous) generative systems destined to produce sound material, excluding thus the softwares composing music scores. In the rest of the thesis, we will refer to them simply as *generative systems* without recalling this limitation. We will try in this part to reference the most notable systems which have been subject of publications and that we encountered along our research. They mark out an impressive evolution and a great variety from the first uses of algorithms (like formalized in [Chadabe 84] for example) to the present, complex and powerful systems that we can find in the various conferences around computer music. It is important to notice that numerous *ad-hoc* or confidential systems are developed almost everyday by sound and computer engineers, composers, musicians... who do not feel the need of writing and publishing anything about them and are happy to use them with successful results. In these cases, it is very difficult to investigate their approach and functioning.

The organization of our review is mainly inspired by the pioneer work and reflexion on this work of R. Rowe in *Interactive Music Systems, Machine Listening and Composing* [Rowe 92] and the more recent article of T. Blackwell, O. Bown and M. Young, *Live Algorithms: Towards Autonomous Computer Improvisers in Computer and Creativity* [Blackwell 12]. Important reflexions about the types of generative systems may also be found in [Cope 05], [Papadopoulos 99] and [Biles 13].

These works present various approaches and propose the characterizations of many examples of generative music systems. Main criteria are technical or musical, and aim to discriminate among the abundance of existing systems. However these different criteria do not inform on the same aspect of the different systems. In particular, when [Rowe 92] and [Cope 05] detail the learning capabilities of the systems, or the type of generative algorithm used, they place their classification on the operational aspects. That is, they discriminate their numerous examples on *how they work*. Whereas in the beginning of the article [Blackwell 12] and in [Biles 13], criteria as *spontaneity*, *participation* or *novelty* denote behavioral aspects of the systems, that is *how they interact* with their musical context.

Our work has been mainly focused on the operational approach. Thus we will present in chapter 2 a new set of three operational criteria over which we propose to place the notable examples of the literature. Though, we will also consider in the second chapter of this part (chapter 3) behavioral reflexions around the musical properties and the musical evaluation criteria of such systems. Both the operational and the behavioral aspects help conceptualize our own approach and characterize the computer system which constitute the center of this thesis.

Chapter 2

Operational Approach

Most of the generative systems which are subject of publication are described through scientific research papers, conferences or books. This channel of publication focuses *de facto* the description of such systems around their functioning and their scientific and/or technical innovations. Even though this angle may be open to criticism when working on system intended to produce music, this operational approach is also the one we mainly take in this work.

Rather than utilizing strict categorization of the different systems of the literature, we propose in this chapter to use three operational axis. We will place a selection of systems in this 3-dimensions space depending on their characteristics along the criteria defining each axis. We figured out these three discriminative axis without going deep into the classification of the technical aspects of the systems like the algorithms used or the functional architecture. Our goal is to define continuous and basic aspects usable to characterize any generative system depending on their general principles. Doing so, we draw a map of generative systems which will enable to position our work in the large area covered by all these computer systems. The resulting graph is presented [Figure 1](#).

2.1 The *Time-Scale* Axis

As the first of these axis, we take the *time-scale* of the elements manipulated to produce a musical discourse. Actually, the (temporal) size of the elements hides the real interest of this axis: the level of abstraction of these elements. In the vast majority of musical systems, one of the basic processes — after the irreducible sampling due to digital nature of computer systems — is *segmentation* to obtain longer elements (in time) and avoid the problem of manipulating a continuous and high rate flow of data. The main goal of segmentation of music material is to extract consistent and useful entities that can be meaningfully exploited and manipulated afterwards. Once coherent segments have been delimited, their significance — often expressed in a condensed, *symbolic* form — can replace the actual content of the unit.

Expressed nevertheless in term of size, this axis goes theoretically from the tiniest element in computer music: the sample of a digital signal, to an almost infinite magnitude of symbolic structure — we could imagine a system manipulating music pieces to organize the overall program of a concert, even though we couldn’t find any articles describing such a system. One of the highest level we encountered is a real-time *meta-score*, Soliloque by Fabien Lévy [Lévy 07] which uses fragments of a concert to recompose a piece in form of a musical *mosaic*.

In a purely acoustical parallel, this axis would be illustrated at one end by a *continuous* instrument like the violin or the clarinet¹ in which the tiniest elements of the rendering of the sound are to be taken into account. At the other end would be a CD² or any forward and linear playback technology which allows access only to whole pieces — or sometimes sections of the piece — and have no other time segmentation.

Along this continuous axis, we can place a few markers to help assemble groups of systems. No generative systems actually work at the audio sample level (commonly 44100 samples a second) i.e. with no higher level elements than the continuous flow of 16 or 24bits encoded points of the *waveform* of the sound. Most of the documented systems gather at least small windows of samples — typically around a few thousand, which correspond to the usual size of a Fast Fourier Transform (FFT) — which serve as a starting point to compute some kind of description of the content. These elements have been sometimes named “grains” and especially studied in [Roads 04]’s Microsound. The size of these elements also correspond to the shortest event our ear is able to isolate and fast concatenation of these result in what we call *granular synthesis*. An example of system using units of such size is given in [Schwarz 06]’s CataRT system. It computes one or several descriptions for each unit then builds a powerful organization of these descriptions in a huge database. To generate a new musical discourse, it navigates knowledgeably through this database and concatenates a stream of grains.

The second *scale* marker we can place would be around the duration of a note or a chord. As our axis is not meant to be graduated in actual durations but is more an axis of *symbolic* level, this second marker indicates a very important transition between what can be called the *subsymbolic* level and the *symbolic* level as discussed in [Rowe 09]. A note or a chord is — in the music notation domain — a unit which has a significant *meaning* independently from the reality of its signal-level embodiment, a musical *representation*. It belongs therefore clearly to the symbolic domain whereas the grains previously mentioned are actual sound elements irreplaceable by symbols. System such as the one described in [Dannenberg 93] makes clear this transition with the direct manipulation of what is called *events* in this article and which represents directly the symbolic content of the musical material used to generate. These events are to be placed right next to our second

¹by opposition to the Piano or other keyboard or plucked instruments in which the sound of a note is almost totally defined by the attack

²which is obviously not purely acoustical as it is a digital format but compact cassette and magnetic tape are quite old fashioned now and vinyles are not a univocal example because of their use in DJing and other music creation processes

marker and a huge collection of generative systems of various principle operate at this level as [Miranda 01]’s CAMUS cellular automata for example.

An even clearer expression of this transition is given with the massive use of MIDI protocol thoroughly analyzed in [Loy 85]. Its standardization and extensive use — even nowadays — in the music industry shifted this protocol to be used as a usual and acknowledged description of music content for solos, chords, jazz standards for example and much more applications. Numerous and various generating computer systems work directly at that level or make use of MIDI-like description, even very recent ones like [Pachet 13]’s chord grid classification. MIDI events being defined with the succession of note-ons and note-offs with their pitches and velocity — and channel information, sometimes complemented with more continuous controls like bend or after-touch — they do not encode the audio content or *realization* of it. Thus they denote a symbolic representation of the musical content often directly convertible to notes of the traditional notation system.

The third important marker along our *time-scale* axis may be the *phrase* level. Typically, a phrase is defined either between two sizable silences or with a pre-determined number of beats, chords, bars or measures. Despite the possible variation in the duration of phrases, this level of segmentation is also commonly used in generation systems. [Biles 94] for example, uses a genetic algorithm on *phrases populations* to breed novel phrases. Another noticeable example is given in [Carey 12] where the author constitutes a database of phrases described with the mean and standard deviation of various descriptors over each of them. As a comparison, a looper pedal from a guitarist’s typical gear works also at this phrase level — in this case, a phrase is defined by the duration of the first loop.

Finally, we can continue this axis up to the music piece duration (typically several minutes). We could add the song’s verse/chorus duration which would be around the phrase marker and the classical piece’s section or *movement* which would be in between the phrase and the piece duration. These last marker are relevant in term of musical content but far less discriminative in computer systems.

2.2 The *Learning* Axis

To draw our operational map of generative computer softwares, we use a second axis representing the *learning* characteristics of the systems. *Learning* is defined for the common sense in The American Heritage Desk Dictionary by the *gaining of knowledge, comprehension, or mastery of, through experience or study*; in the case of computer software, *to learn* describes the ability of a system to improve its performances with the knowledge of some data. This task can be supervised or unsupervised, meaning that there may or may not be definitions, specifications or documented examples of the properties to learn. More specifically in the domain of music generation, we mean by *learning*, the ability for a generative system to *retain* and improve from any kind of information from input data, either coming from training experiments, from externally defined corpus of

data or from on-the-fly discovery. An notable remark is that *retaining* implies some kind of memory in the system either literal (the system records the data as it was presented to it) or abstracted (the system hold some information extracted from the data it was presented to). However, recording or keeping track on data in a memory only is not sufficient for a system to be considered as *learning*. Learning is also characterized by some kind of influence or modification of the functioning of the system itself thanks to the data. For example the training of a neural network as used in [Young 08] modifies its subsequent behavior thanks to training input data. Even though these data are neither recorded literally nor reused we do consider that Young’s NN Music system is indeed learning, at least during the training phase.

The *learning* axis typically goes from purely rule-based systems — which can be considered as *not learning* at all, rules being predetermined — to totally *agnostic* systems which have to discover everything from their input. One of the first and almost purely rule-based example of generation systems is [Holtzman 81] which uses language grammar or rewrite rules to analyze existing repertoire and generate pieces of composition. Cellular automata systems such as those of [Miranda 01], already mentioned, expresses the evolution of a system as a set of predefined transition rules and are pre-determined by the initial state of the system and the application of these rules. On the other end of this axis, a system learning absolutely *everything* from its input without any prior hypothesis certainly does not strictly exist as there should be, for now in computer systems, some kind of definition of the information to look for or extract from the input data, or at least a procedural description of how to handle the input.

In the analogy with classical instruments and effects, any acoustic instrument is definitely not supposed to learn anything from its player and would be placed at one end of the axis with rule-based systems. As previously mentioned, no technical system wether analog or digital is actually capable of learning *everything* from scratch.

An important remark has to be done on this definition of the *learning* axis: whether the learning is done in real-time, along the process of acquiring material for an input as in [Collins 11]’s LL system, or on a corpus or database of pre-recorded material as done in [Collins 12]’s Autocousmatic system to generate new acousmatic pieces, is not a criterion on this axis. Then we can summarize this axis as: how much *a priori* (either implicit or explicit) does the system contains before analyzing any material. The more *a priori*, the less we consider it as *learning*, and conversely, the less *a priori* information or hypothesis the system needs, the more it has to learn from the material to build a consistent discourse. For example, the typical bass/chords/melody division of idiomatic Jazz that we can find in [Thom 00], [Pachet 13] and others, is considered as an *a priori* and moves these systems in our map towards rule-based systems. The roles of each musician is indeed not learnt by system and may not be changed. On the other hand, the *cumulative rehearsal database* of [Carey 12]’s *_derivation* system or the user’s *judgement* influencing the evolution of [Biles 94]’s genetic algorithm stress their *learning* capacities. This characterization concerns equally the analysis and generative part in generative

systems — most of those systems can usually be divided rather clearly in an analysis sub-system and a generation sub-system.

2.3 The *Control* Axis

The third axis we propose to analyze how generative systems work, describes their need for human *control*. This is mainly inspired from the discussion of [Rowe 92] about the instrument vs. player paradigm. R. Rowe defines the two directions of our axis like this:

Instrument paradigm systems are concerned with constructing an extended musical instrument: performance gestures from a human player are analyzed by the computer and guide an elaborated output exceeding normal instrumental response. Imagining such a system being played by a single performer, the musical result would be thought of as a solo.

Systems following a player paradigm try to construct an artificial player, a musical presence with a personality and behavior of its own, though it may vary in the degree to which it follows the lead of a human partner. A player paradigm system played by a single human would produce an output more like a duet.

In the context of a musical group or ensemble, these paradigms characterizes the dependency of the computer system on a human to play and interact musically with the other musicians. In other words we discriminate along this axis the role of the computer in the overall setup. Is the computer system *played* by a human like a new maybe smart or reactive instrument? Or is the computer a musical companion or partner on its own?

In practical terms, this axis is linked (but not restricted) to the *technical* manipulations needed on the computer i.e. the direct interaction — with traditional computer interfaces like mouse, triggers, knobs etc. or many more paradigms and novel interfaces as discussed in [Wessel 02] — required between a human performer and the system to generate an engaging musical discourse. This practical approach of the control is also considered in [Biles 13]. The same extreme examples as given for the previous axis can help defining the end of this axis: a CD or a predefined MIDI sequence do not require any control. Once started, they will unwind their unchangeable content independently from any human — unless or until a “stop” button is pushed, action which can be considered as the simplest and even minimalistic interaction with a machine. On the other end of this axis, an acoustic violin is a useless instrument without the violin player, no sound will ever get out of the instrument if nobody actions continuously its mechanisms (except perhaps in the case of open string *pizzicati*). This way, a violin requires permanent control over all the elements of the performance, from the tiniest bits of sound to the overall form and musical direction of a piece.

Naturally, several generative systems present a user interface which allows to control them but it is still up to the *supervisor* of the computer to decide whether he or she

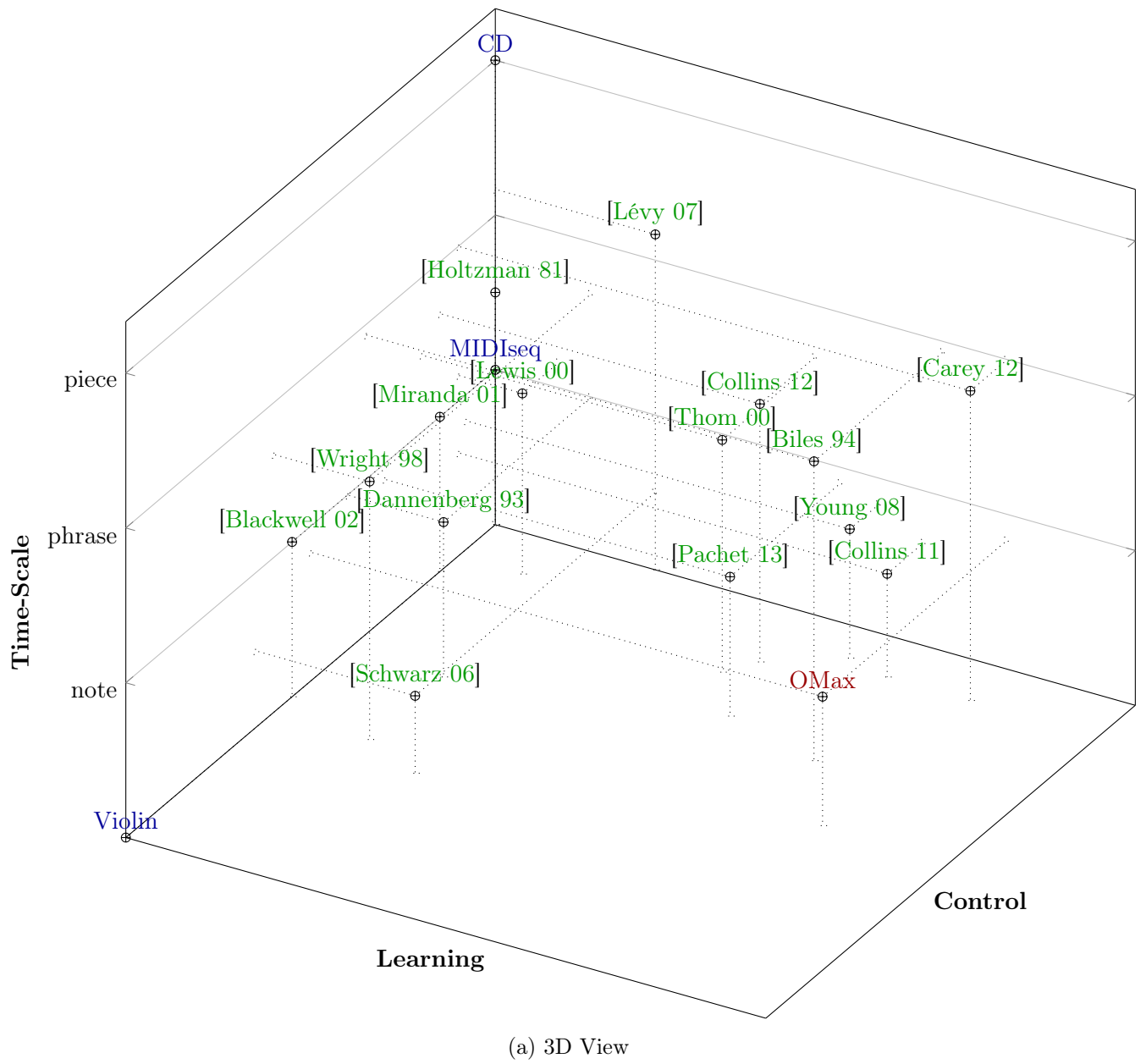
wants to intervene or not. And the same system can be *played* very differently by two *supervisors* depending, for example, on how much other things they have to manage or how much confidence they have on the computer system to react pertinently during the performance. George Lewis, when playing his *Voyager* system [Lewis 00] has much to do already with his acoustic trombone and will not spend too much time tuning the computer during the performance, even though *Voyager*'s interface gives access to numerous parameters for fine tuning. This interaction can obviously be seen the other way around: the conception of [Lewis 00]'s or [Thom 00]'s systems is oriented towards a certain *autonomy* to allow direct interaction with a musician without the need for a human dedicated to the computer's *supervision* while M. Wright & D. Wessel's system to improvise indian *Tal* is clearly designed to be controlled by a *computer musician* [Wright 98]. It is precisely this orientation towards the dependance or the independence from the human control we characterize on this axis.

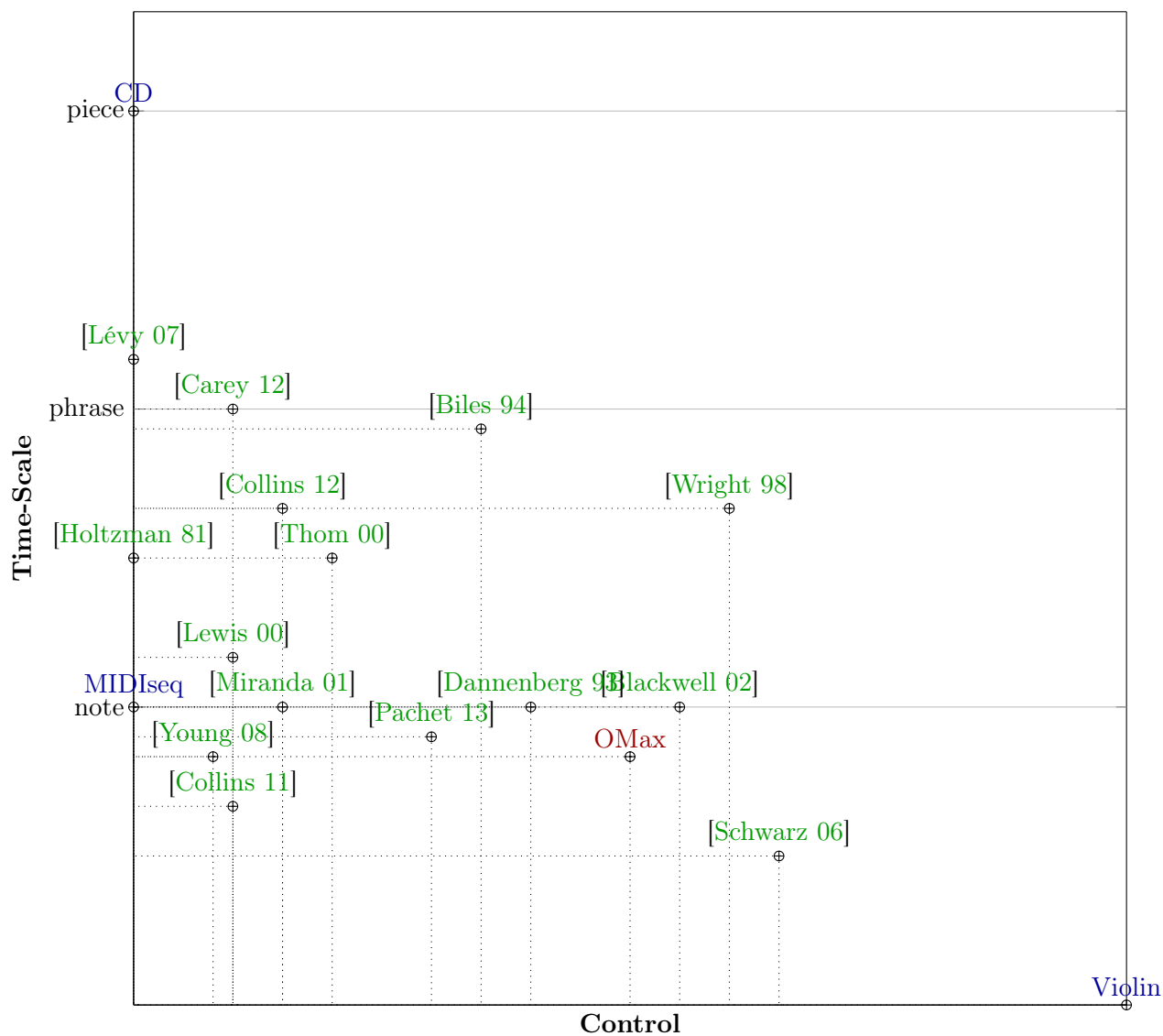
System's like [Lévy 07] or [Holtzman 81] are positioned at one end of this axis, they are almost totally autonomous systems which require almost no control. They are oriented towards composition and only need a set of predefined rules to unwind their piece, fixed once and for all in the case of [Lévy 07]'s Soliloque, adjusted for several generations (including a musicology example to *describing a Schoenberg trio*) in the case of [Holtzman 81]'s grammar. At the other end of the axis, [Schwarz 06]'s *CataRT*'s or [Blackwell 02]'s *Swarmusic* need for a running *target* guiding the path in the grain database (in the case of *CataRT*) or the attraction of the particles (in the case of *Swarmusic*) pushes towards a continuous control (manual or scripted) of the computer based generation. Naturally, this continuous control can be replaced by the following of a real-time input or the running of a pre-defined script.

2.4 Operational Plot

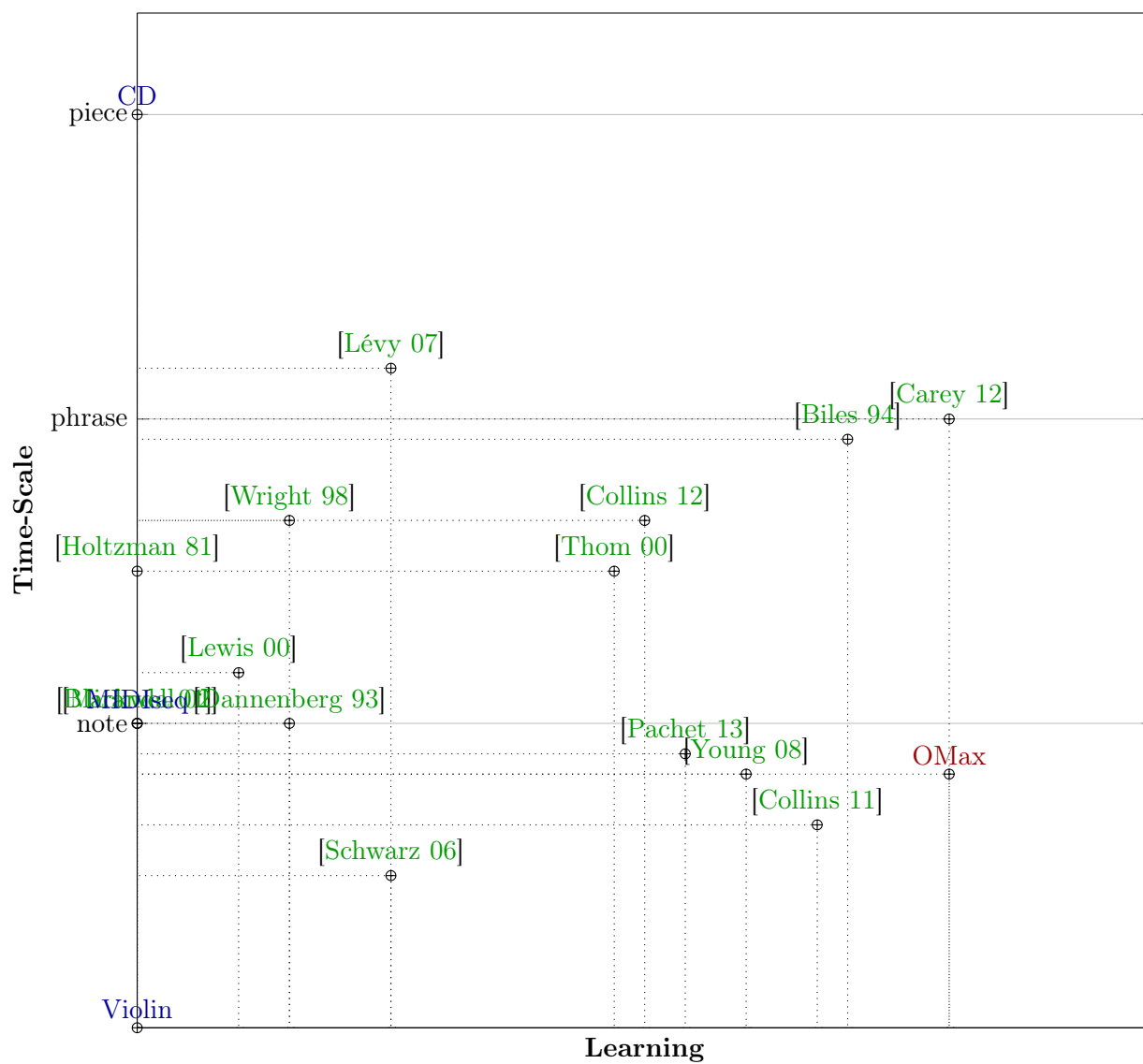
We defined in the three previous sections, three axis to describe operational aspects of generative systems and help us place our own work in this review map. We gathered in the following plot (Figure 1) all the references we mentioned earlier in this review and gave them coordinates on the three operational axis. To the list of these systems, we added our work around OMax (in red) and the analog examples we gave as illustrations along our description of the three axis (in blue). When deciding the coordinates, we tried to respect the specific ideas and functioning of each system but compare it to the others in the terms of each axis definition to achieve a coherent map. Naturally we may have under- or overestimated certain aspects of one or several systems. After a 3-dimension view of this plot, we give the three 2-dimensional projections of this map to enable the reader to examine more clearly each axis and each system.

Figure 1: Plot of Generative Systems

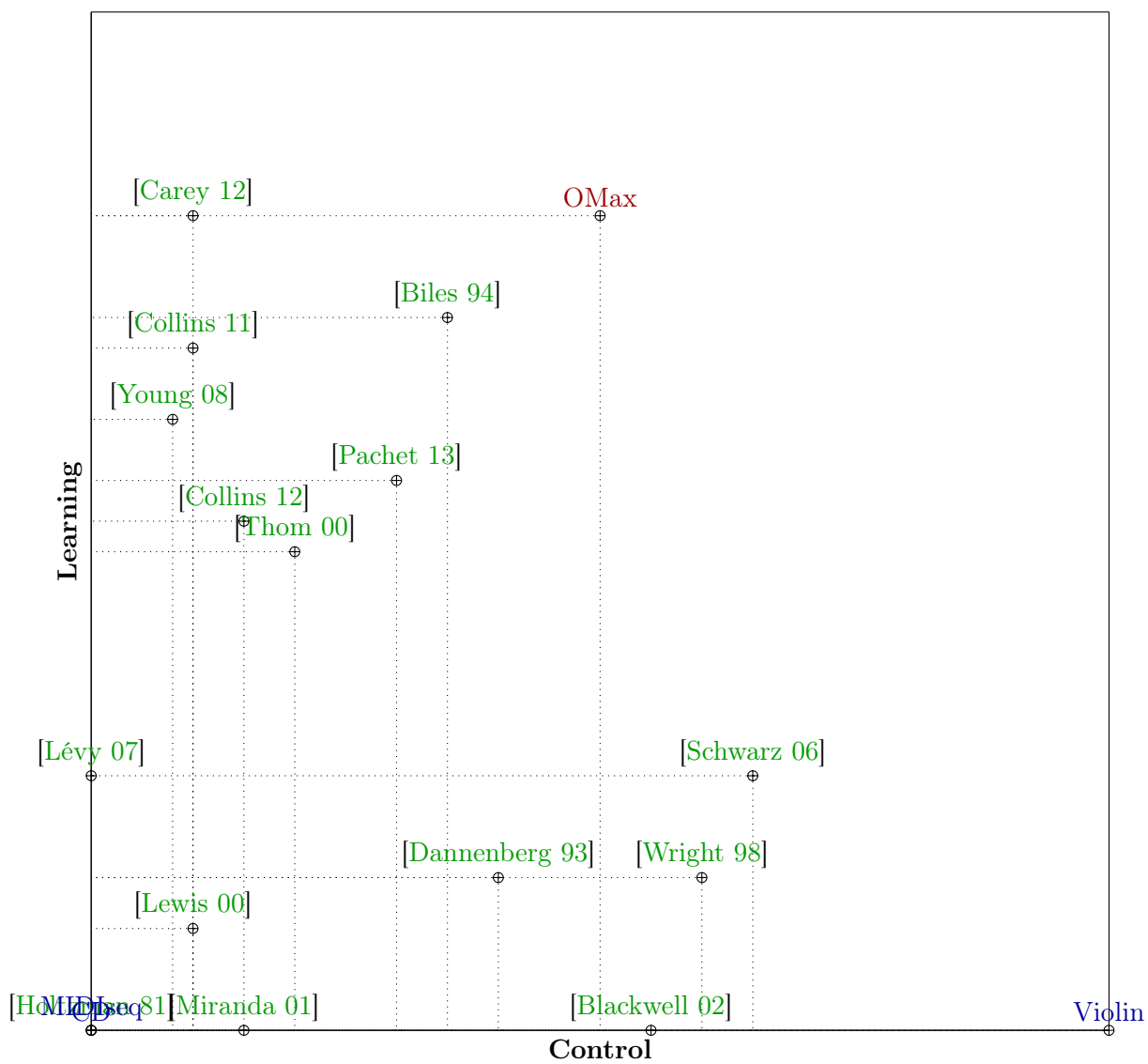




(b) *Control/Time-Scale* Projection



(c) *Learning/Time-Scale* Projection



(d) *Control/Learning* Projection

From this graph and its three projections, we can draw a few conclusions and explain the position of our work. On the *Control/Time-Scale* projection, we observe that most systems are naturally placed along a diagonal: the longer (and higher in symbolic level) the elements are, the less control the system requires. This is especially illustrated by the position of [Lévy 07], [Carey 12], [Thom 00], [Dannenberg 93] and [Schwarz 06]. This distribution suggests that a higher view on the structure of music enables a system to reproduce more autonomously such structured discourse. Though, many systems are placed below this diagonal and notably towards the systems with less human control which shows a tendency to invent systems capable of taking on (or trying to take on) the role of a musician. We discuss this autonomy in the next chapter (section 3.1). In our work, we propose a system which takes place along the diagonal but takes on the role of what we could name a *generative instrument*, that is, a system capable of generating new musical discourse on its own but enabling a human *supervisor* to bend and conduct efficiently the content with high level musical decisions. It implies that our system manipulates elements long enough to model efficiently the musical structure and generate new coherent discourse but is versatile and controllable to enable the musical conducting of the human supervisor. On the *Learning/Time-Scale* projection, systems are rather well spread and we can see that the learning ability is not directly related to the level of the musical units. [Lévy 07] shows a natural example of the predetermined arrangement (composition) of long units while the concatenation of very small grain in [Schwarz 06] do not exhibit more learning capacities. In this domain, our work takes the opposite direction of [Carey 12] by learning the most information possible from the organization of rather small units. Finally, we expected from the *Control/Learning* projection to backup the assumption that the more a system learns, the less control it requires. But apart from [Schwarz 06] which concatenation mechanism requires important control, learning systems such as [Biles 94] do not seem to imply less control than [Dannenberg 93]’s interactive composition system. Again, the overall distribution illustrates the tendency to conceive more and more autonomous computer systems. We challenge this tendency in our work and pushed our research towards a controllable agnostic system.

Chapter 3

Behavioral Reflexions

The operational approach corresponds to many historical analysis of generative systems. In the previous chapter, we proposed a new set of three continuous characterization of generative systems which enable us to place several examples in a 3-dimension space illustrating *how* these system work. However in more recent work as in [Blackwell 12] or [Biles 13] the behavioral aspects of the systems are being justly studied. With different approaches, these discussions and characterizations focus on the interaction of a computer system with its musical context. While T. Blackwell examines mainly the comportment of the system itself on a musical level, J. A. Biles also includes in his reflexion the interaction with the musician “responsible” for the system on stage and the appeal for the audience. In turn, we have to undertake reflexions about the behavior of generative systems. This approach will give us an appropriate angle to discriminate systems oriented towards improvisation from the larger group of generative system (section 3.1) and to broach the sensitive problem of the evaluations of such systems (section 3.2).

3.1 Improvisation Systems

Participation In the previous chapter, we voluntarily reviewed a selection of notable *generative* systems which are built with very different musical purposes for historical, technical or aesthetic reasons. They fit together in the vast domain of computer systems for music generation as would also several commercial and/or general public tools, softwares, hardware gear or electronic instruments. What differentiates the former systems studied in the previous chapter from the latter only mentioned here as a comparison is the *creative* role i.e. the *active* role of the computer which allows us to properly define the *generative systems* group. Such systems not only help expressing a musical discourse but also shape itself the musical matter. With this characterization, we can not differentiate composition systems for improvisation system but we can discriminate systems which *assist* composition or improvisation — which can be expert systems as [OpenMusic] or pedagogical softwares — from systems which effectively generate new musical material, that is systems not only suggesting ideas or directions but also capable of participating

to the musical result. Running these programs — working with swarms, grammars, reinjection mechanisms, live coding principles or any other principles — the computer system exhibits some *participation*: it does not only execute a succession of rules to get the result but suggests some new directions. Naturally, depending on the system, it may suggest these directions to the *supervisor* interacting with the computer or directly to the musicians participating. Each system is also biased in its suggestions and directions by the angle of its core and programming principles. This influence of the system on the musical interaction and result system has been more thoroughly studied in [Bown 09]. But this *participation* is a characteristic shared between the generative improvisation systems and the generative music composition systems that we reviewed.

The three axis we defined to position the generative systems that we reviewed in our operational approach do not discriminate *what* musical situation(s) the computer system is capable of taking a role in. For example, we reviewed generative systems which explain and build composed music pieces like [Holtzman 81]’s system or idiomatic electroacoustic pieces like [Collins 12]’s Autocousmatic and systems generating a continuous stream of sound in a musical improvisation, like [Blackwell 02]’s Swarmusic or [Schwarz 06]’s CataRT. The musical situations these systems can participate to are obviously not the same: Collins’ Autocousmatic can not be used to improvise with a musician, at least not as it is described in the article. Conversely, it requires a *compositional* work to make Blackwell’s Swarmusic participate to a composition — for example with a simple script like it is described in [Blackwell 02] in the case of a “solo” improvisation. And in this latter case, it is not clear which part of the composing process the system is taking on.

Adaptation Among many definitions and discussions about the nature of improvisation, T. Bachir-Loopuyt, C. Canonne et al. characterize the improvised work this way (in French) in [Bachir-Loopuyt 10]:

Contrairement aux œuvres de la tradition classique qui sont des types abstraits, les improvisations sont des événements. Il existe néanmoins une continuité entre ces deux régimes ontologiques, que fait apparaître le contextualisme de Levinson. Dans sa théorie, l’identité des œuvres musicales classiques est déjà dépendante du contexte de leur création. [...] Cette dépendance contextuelle est rendue manifeste dans le cas de l’œuvre improvisée dont l’identité est exhaustivement déterminée par le contexte de sa création : c’est le fait d’avoir été exécutée à telle date par telles personnes qui caractérise in fine l’identité de l’œuvre improvisée.

We can translate it literally this way:

Unlike the works of classical tradition which are abstract types, improvisations are events. There exists however a continuity between those two ontological schemes revealed in the contextualism of Levinson. In his theory, classical music works’ identity is already dependent from the context of their creation.

[...] *This contextual dependency is manifest in the case of the improvised work which identity is exhaustively determined by the context of its creation: it is precisely its execution at a given time by specific persons which characterizes in fine the identity of the improvised work.*

We find in this description of improvised work a very deep relation with context and time and more specifically with the context and time of the unwinding of the improvisation. This point seems to be the essential difference between generative composition systems and improvisation system. It is particularly obvious in the notable work of D. Borgo, *Sync or Swarm, Improvising Music in a Complex Age* [Borgo 05]. This permanent need for improvisers and improvisation systems to *adapt* their internal state, directions and choices to the running context of their musical discourse contrasts with the reflexive and iterative processes of composition — which are well illustrated in [Collins 12]’s Autocousmatic system. A fine attempt of modeling these adaptation processes in the context of Collective Free Improvisation has been done by C. Canonne through a formalism coming from dynamic systems’ mechanics with very eloquent results [Canonne 11]. In this article, from a purely theoretical signal x , C. Cannone and N. Garnier model some higher level information like intention, objective thanks to equations of the dynamic systems’ mechanics. Then they resolve this set of equations to observe the dynamic behavior of each musician in the group depending on their number, virtuosity and other musical qualities. When pushing this characterization of improvisation, we could say that composition is a process which allows back and forth examinations and changes of the result while improvisation is an inflexible unrolling of the result without any possibilities to change what has been played. Naturally, several musical processes — perhaps almost all the actual musical processes — take place with many subtle balances in between these two extreme views. But this characterization has strong consequences on the design of computer generative systems oriented towards one or the other musical situation. In particular, the real-time consideration of the system for its context and the *adaptation* to this context is a strong requirement for an improvisation system as also claimed in [Young 08]. In the extreme case of Live Coding for example even though we have to consider the *team* including the computer and its supervisor as *the system*, this need for adaptation has strong consequences on the design of the computer system otherwise any programming language or environment would be suitable for musical Live Coding — which is very far for reality! Blackwell’s other “*basic set of capacities in order to participate appropriately*” [to a collaborative improvisational context] in particular *novelty*, and *participation* [Blackwell 12] seems to be consequences or ensuing characterizations of the assumed *adaptation* capacity of the system.

Autonomy Finally, we could try to differentiate *improvisation* systems from *improvising* systems. We used indiscriminately both terms up to this point of the thesis — and we will continue doing so in the rest of our work. But as with the extreme opposition of composition and improvisation it seems to us that this difference in the usage of the system

does not rely only on the question of *control* or *autonomy* addressed through our third axis. The control can indeed be eluded for example through the usage of a real-time input to define a *target* — as done in the *Audio-Controlled Synthesis* scenario of [Schwarz 06] or the *capture algorithm* of [Blackwell 02] — or with the usage of pre-defined scenarios implemented in scripts or presets of parameters. Moreover, the control, or the autonomy of the computer system is only relevant if we consider the sole machine. This view of an *autonomous* machine pursues the dream of a computer system which would generate pertinent material without any direct human supervision. Our view on the contrary includes the *supervisor* of the computer and considers rather the behavior of a (sometimes virtual) entity composed by the generative system and its supervisor. This view is partly inspired by the feedback of our experiences with acoustic musicians. In most of the real-life musical situations, the acoustic musicians feel perfectly well that our system is not strictly autonomous — and will probably never be — and they are sensitive to the role of the supervisor or of the *automatic* behavior pre-define by scripts. This view also allows us to include in our behavioral reflexions Live Coding situations for example, as the one described in [Brown 09]. In this situation, the role of the supervisor is tremendously amplified but the influence of the coding framework (or language) is not to be neglected nevertheless.

From a generative computer *improvisation* system, we expect at least to suggest or produce some innovative musical material. But the choices, evaluation and decisions concerning the usage of this material may belong to a *computer musician* — the supervisor of the computer. He or she is effectively responsible for most of the musical choices even though his or her system slants drastically these choices. On the other hand, some hope to be able to implement in a not so distant future a completely autonomous *improvising* system capable of playing on its own and interacting musically with other human or computer musicians. The reality of today's computer systems is widely spread in between those two illustrative cases. And the (philosophical) question we may ask for most real cases is: who is improvising most? Is it the computer system or its supervisor? In practice, the answer is not trivial and we leave it open for the moment.

3.2 Evaluation

The general evaluation of computer systems aimed to generate music is a very thorny problem. The main bias comes from aesthetic considerations inherent the musical aspects of such systems especially in the situation of improvisation. When we distinguished at the beginning of the previous section (3.1) systems *to assist* improvisation from generative systems *for* improvisation, we implied that the first goal of the latter systems is to directly make music. Many computer systems from software instruments to improvising softwares embrace this goal. As such, this goal is impossible to evaluate — or always fulfilled as soon as some sound is output by the system — without going into subjective perspectives as developed in [Ariza 09], one of the rare references on the subject. However this may not

be the only intention when designing such a system. Indeed, musical and improvisational constraints imply a strong software architecture research. Designing the proper architecture to implement efficient and pertinent interaction between the different parts of the system and the outside musical world is a goal *per se*. And it constitutes the main issue of the work presented in this thesis. The comparison of musical output of different versions of a music generation system is the approach of [Hsu 09]. W. Hsu and M. Sosnick make use in this work of questionnaires filled by both the musicians playing with the system and the audience and tries through the answers to these questionnaires to compare different versions of the computer system in terms of the musical behavior and result. Doing so, they explicitly try to avoid the *subjectivity* denounced in [Ariza 09] by this comparison of two versions of the same system. However C. Ariza's article thoroughly deciphers several tests imagined and used in the music field and show that none evacuates the *aesthetical* bias of the listener(s). Our own view on the versatility of improvisation explained later in this section reinforce this point. C. Ariza concludes with:

Until machines achieve autonomy, it is likely that humans will continue to form, shape, and manipulate machine output to satisfy their own aesthetic demands, taking personal, human responsibility for machine output

Improvisation Mechanisms However, besides music and software concerns, there are some *human* interest in developing system aimed at improvising. Whatever core mechanism they rely on, the testing of generative softwares naturally includes multiple (musical) confrontation with human musicians. The human improvisator takes here the role of both a judge and a enlightener of the systems qualities or drawbacks. In this mirroring, it seems to us that there is not only evaluation pointers about the system but also some knowledge to gain about the human musician engaging a dialog with the system. Even if the human cognition — studies in several other aspects and with very different methods — is certainly not the model for all the generative system, we feel that being capable of designing a credible improvising computer systems also informs us on *how humans improvise*. In other words, the usage of techniques in form here of computer programs reveals and may give important clues on mechanisms of human behaviors.

Versatility Musicians though have themselves very various ways of engaging a musical dialog. Especially in the case of improvisation, the result is of almost infinite variety even with the same musicians in the same place and conditions, depending on their feeling, their mood and hundreds of slight differences. The same variety will appear in the testing of one setup involving the computer system with various musicians. In this conditions of extreme versatility, how would the examination of improvisations' results be of any information about the computer system participating to the improvisation? [Hsu 09]'s work seems to circumvent this problem with the statistical significance of a big number of questionnaires filled in by four musicians and a consistent number of persons of the audience. In this

case, the variety in result and the inevitable *risk* of improvisation is replaced by the number of subjective opinions on one result¹. We did not adopt this approach coming from the Human-computer Interaction field mainly oriented towards the research of new interfaces, devices or tools as intermediaries to control computer systems.

Getting back to the music objective of an improvisation system, in the work presented in this thesis, we choose to estimate the value of the system presented and its different versions through the diversity of musical situations it could take place in. We enjoyed very much playing with the system and putting it at risk with fresh prototypes and/or newly met musicians in several situations. These situations were diverse as *impromptu* public improvisation in a barn during a summer festival; formal, rehearsed and well organized concert in front of an audience used to contemporary written music; participation with a formed jazz band to straight-ahead Jazz festivals; construction of a vocal tale in a theater piece. The numerous experiences lived with the system along this four year research enabled us to gather several type of feedback. We had the opportunity of playing with numerous and very various musicians in open workshops gathering this way several different opinions of the same version of the system. We participated as well in long term construction of regular musical productions with the same musicians but with the successive versions of the system. All these experiences are succinctly gathered in the *List of Musical Experiences* at the end of this thesis and we detail some of them in [chapter 11](#). This recurring but always renewed interest of musicians to our system gave us confidence if not directly in its constructional or technical value, at least in the musical pertinence its design enables.

¹Hsu's approach is actually more precise than depicted here and includes rehearsals and differential questionnaire for a comparison of two systems.

Part II

Principles of an Improvisation System

Chapter 4

General Principles

In this chapter, we introduce the background principles for a system dedicated to musical improvisation. We based our research on the concept of *stylistic reinjection* and on the earlier experimentations on the OMax environment that we have completely reconsidered conceptually (Part II of this thesis), redesigned and provided with a concert-ready implementation in C and in Max (Part III of this thesis). We first present in this chapter these background foundations of our work through a short history of style simulation and reinjection (section 4.1) leading to a short description of OMax principles and objectives, both on phenomenological and operational aspects (section 4.2). A very brief overview of the functional organization of the system is also given section 4.2.3 in order to give to the reader a general notion of the different elements which will be referred to and detailed further in this thesis. Then, as an introduction to the thorough work achieved on the conceptualization of such an improvisation system and detailed chapter 5, 6 and 7, we explain the cognitive insights which helped organize our research (section 4.3). A musical counterpart of these cognitive insights will be presented in chapter 11 where we will develop the musical experiences and accumulated along this work.

4.1 History

The work presented in this thesis finds its roots in *style simulation* i.e. in the capturing of specificities of a musician or a composer which define his *style*. One of the first example of a style simulation can be found in the software M by J. Chadabe and D. Zicarelli [Zicarelli 87] where a Markov model of an incoming MIDI sequence is built and walked to later output a modified version of this sequence. By this functioning: the direct modification of an input on the symbolic level (thanks to the MIDI protocol), this system already acts as a capture and remodeling of the style of its input even though the *style* was not the center of this work. From 1998 thorough style modeling studies have been undertaken [Dubnov 98, Assayag 99] relying previous research on universal compression and statistical algorithms like Lempel-Ziv compression [Williams 91] or Incremental Parsing [Ziv 78]. These algorithms which usually encode arbitrary sequences

through Markovian principles are used in the case of music to exhibit properties of the sequence(s) they encode. Then, most of the time, thanks to the prediction aspects of Markovian theories, they enable to generate new sequences showing the same (or close) musical properties. One of the main interest in such modeling technics is their agnostic characteristic: no musical *a priori* either with rules or implicit with music theory frameworks is needed except the assumption of the sequential nature of the input.

The stylistic study of already existing music material have always been coupled in this project (and most of others) with the generation of new material both as a proof of correctness and reproducibility of the processes developed and, as discussed in [Conklin 01], as an important characteristic of any music theory, either explicit or underlying statistical models. Along the years, the study on style modeling and the investigation of new material generation lead to the definition and thorougher exploration of *stylistic reinjection* [Assayag 06a, Assayag 10, Surges 13] as a ground principle for musical interaction and improvisation.

Alongside, the OMax software has been developed [Assayag 05, Assayag 06c] to illustrate the theoretical explorations and to try out these new musical interactions with acoustic musicians both in the frame of improvisation and composition [Bloch 08, Robin 09].

4.2 OMax Principles

As previously explained, the work presented in this thesis took as starting point, the state of the research around stylistic reinjection and the OMax software as of 2009. To understand the formalizations, descriptions and advances described along this thesis, a brief sketch of OMax principles is needed. We will therefore outline here in a few words the global musical behavior, the rough functioning and the basic architecture of this stylistic reinjection system.

4.2.1 Phenomenological Description

Reinjection as presented in [Assayag 06a] relies on the hypothesis that new musical discourse can be formed from inference and recombination of existing material. Provided that the learnt material is vast enough, the new material generated by recombination of it constitute a real source of novelty exactly as human improvisator draw their innovation capacity in the recombination of material they have heard, trained on and/or memorized. And if the material learnt is consistent in style, then the generation of new material from this memory constitute a credible stylistic simulation.

To apply stylistic reinjection to music, the Musical Representation team of IRCAM with other researchers implemented a system capable of listening to a musician, learning his or her musical discourse using a knowledge model and generating new variations of this discourse thanks to the model. The system acts on the sound as a distorting mirror. It

records the raw data of the musician — its natural sound output in the case of an acoustic instrument or its MIDI output in the case of digital instrument — and replays small parts (patterns) of this sound chained one to another in a very different order than the original discourse. Naturally, many parameters influence the recombination of the musical portions such that numerous musical results can be obtained: from a close replication with very few variations from the original music, to a superabundance of recombinations making the link with the original material difficult to hear or apprehend.

One notable specificity of such a system applying strictly the stylistic reinjection principle is the conspicuous mimicking nature of the result. The recorded material is used to render the output of the system so that the musician feeding the system and listening to the output is confronted to a disturbing mixture of his or her own sound with his or her personal articulations, style or mannerisms recombined with the abstract computer's grasp on this musical discourse. This situation creates at the same time a very intimate relation between the input and the output of the system and a confusing distance between what the musician is feeding to the system and how the system uses and replays parts of it.

4.2.2 Operational Hypothesis

To concisely describe how a stylistic reinjection system works, we can use the three axis developed [chapter 2](#) to analyse generative and improvisation systems. Here are the three operational hypothesis on which the OMax system is based:

- *time-scale*: the modeling in this system is done at the symbolic level. An abstract memory model is used which is based on a graph. This graph uses univocal “letters” (symbols) and recognizes repeated patterns (sub-sequences) of these letters. This knowledge is then used to generate new musical discourse mimicking the style of the material learnt.
- *learning*: even though the knowledge model can be prepared with pre-analyzed material (see [6.1.3.1](#) and [6.3.2](#)), the typical usage of this system starts with an empty memory, which means that the system has to learn everything on-the-fly from the acoustic musician playing along. As a consequence, we have to avoid any hypothesis regarding the style and the content of the input material. That is: the system needs to be as agnostic as possible to handle the most various musical situations and learn the most information possible from its input.
- *control*: the system is supposed to be driven and supervised by a human. In the typical improvisation situation, he or she can be either the acoustic musician which also provides the input material or a *computer musician* playing with the system (as the only *software instrument* or as part of a larger setup). However, we intend to provide *musical* parameters to control the computer system. This goes along with

high level control which can also be pre-determined, composed or automated to give more autonomy and/or behavioral instructions to the computer.

4.2.3 Functional Overview

To describe in a nutshell the computer structure of such an improvisation system based on stylistic reinjection, we can organize its functions into six distinct elements illustrated Figure 2. The input is used in two ways: it is recorded to be replayed (by rather small

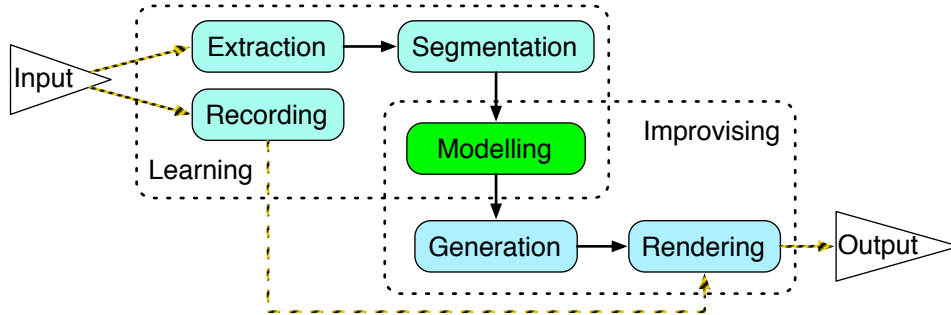


Figure 2: General Principles of the OMax System

portions) later on and in parallel, it is processed to extract information. In order to construct the statistical modeling which assumes only the sequential nature of the input, consistent and consecutive elements have to be precisely defined on the information flow i.e. a segmentation of the continuous flow of information is needed to constitute the elements of the sequence. These elements are then learnt in the model. Concurrently to this learning part, the improvising part of the system uses the model to generate new paths corresponding to new organization of the musical patterns and render these variations by chaining the corresponding portions of the recording as the output of the system.

An important remark follows this brief functional description of the system: there exist in this system a decorrelation between the *learning* process and the *improvising* process. They refer to the same knowledge model but are not coupled neither in time, nor in causality. The only requisite is for the improvising process to have a none-empty knowledge model. In other words, as soon as the system has learnt one element, it can improvise a new discourse with this element. But no direct connection is done between the learning and the improvising processes. This observation frees the system from strict *real-time* constraints. Because of the independence of the two processes, the learning part is not required to work faster than the commonly admitted barrier of a few milliseconds (20 at most) which characterizes a *hard real-time* environment. On the contrary, latency is not an issue in the learning process provided that it is capable of supplying an efficient model on the fly (by opposition to batch processing) which excludes however non-causal processing commonly used on music analysis, data base indexing or information retrieval. We can name this constraint *soft real-time*.

4.3 Cognitive Views

Improvisation among the larger field of human *creative activities* have been an important research subject in many domains of sciences. From philosophy to computer science and particularly in cognitive science. Very various situations of human *improvisation* are studied as can be seen from the numerous articles published on the subject in management, team work or decision making for example. The musical improvisation as a typical and easily isolable example of *creativity* in arts is a real attractive center of the studies especially — but not only — around Jazz (choruses and harmonic grid). To study musical improvisation in general, philosophical point of view as in [Bachir-Loopuyt 10] for example have been adopted along with complex modeling of interactions as in [Canonne 11] or attempts have been made to simulate neurons and networks of those in actual music producing systems like [Young 08].

The main underground motivation for studying and trying to realize a computer system capable of musical improvisation is arguably the clues, information or models it suggests about the human realization of this same task. In a certain extend (only), the underlying hypothesis is that if a computer system *is indeed* capable of improvising *naturally* with one or several musician(s) ie if this system captures and reproduces enough intimate characteristics to be credible in the interaction with humans, then, when creating this system, we have understood (at least partially and in a computer oriented form) what *is*, and *how* human do musical improvisation. This understanding of the human capacity of improvisation may be a good motive for an anthropomorphic view of computer systems for musical improvisation. Other mimetisms are naturally possible for example with animal flocks or swarms as in [Blackwell 02] and the research on computer systems for musical improvisation could simply ignore or refuse any external models. In our research we take on a certain proportion of anthropomorphic conception of the system and even use this view to structure the formal presentation of the principle of this system in this first part of this thesis. In contrast, the second part of the thesis (Part III) will be dedicated to the software architecture of the system, i.e. pure computer oriented views.

We base our anthropomorphic model on classical cognitive physiology or neuropsychology models of human behavior in the situation of improvisation. As summarized in [Pressing 88] it includes three main components:

- *input (sense organs),*
- *processing and decision-making (central nervous system)*
- *and motor output (muscle systems and glands)*

This model is easily transposable to computer systems and paradoxically, this transposition is done in a rather simple and fruitful manner in the $P Q f$ architecture of [Blackwell 12]:

The modules are: P (listening/analysis), Q (performing/synthesis) and f (patterning, reasoning or even intuiting). [...]

Naively speaking, P is to ears as f is to brains as Q is to voice, but in humans, these compartments are themselves conceptually ambiguous.

We carry on with this anthropomorphic model and its computer application and we detail the internal elements of these three parts. This is summarized [Figure 3](#) and serves as the structure of the rest of this first part of the thesis studying the principle of our computer system for improvisation. We named *Perception* the P module of [Blackwell 12], ie the

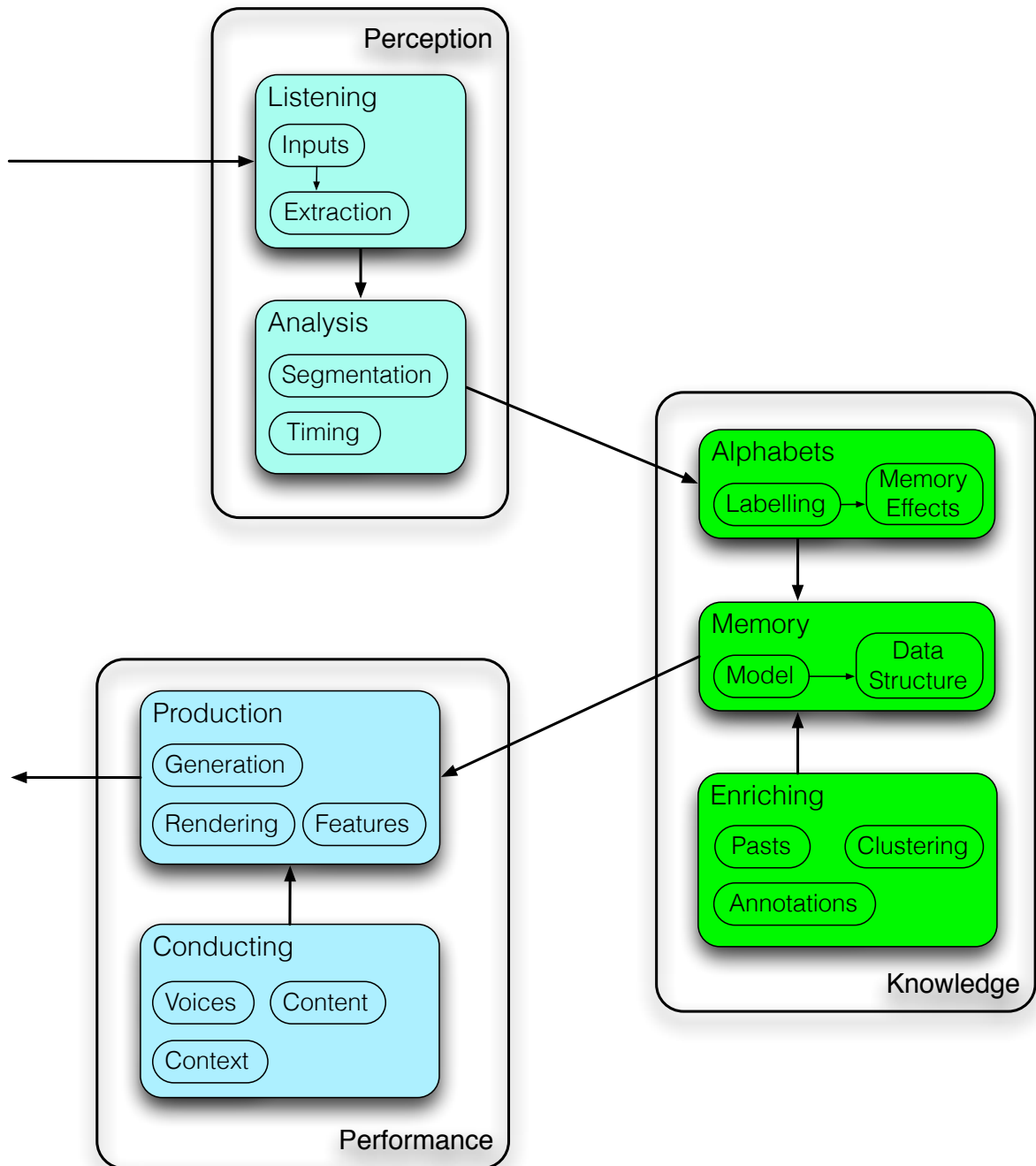


Figure 3: Cognitive Modelisation

input of [Pressing 88] and argue that it contains both the *listening* and the *analysis* of the input(s). We explain [chapter 5](#) how it can be achieved. In the case of our system based on a memory model for sequences, we denote the function f with the term *knowledge* and we divide it into the *alphabetization* — which correspond to the transition from the concrete elements describing the signal to symbols encoding those elements —, the *memory* which is the location for abstract structure to hold the knowledge, and the *enrichment* of this memory through various mechanisms. This module is presented [chapter 6](#). The Q module ie the motor output in [Pressing 88] is translated into the *Performance* part of our model and includes both the effective *production* of the sound and the processes to musically *conduct* this realization based both on the knowledge and on external musical choices (either automatically or with a human *supervisor*). This formalization based on actual cognitive reflexions gives us a strong conceptual framework to present our research on the global conception — that is the study of every elements — of a computer system dedicated to musical improvisation. The *perception* part of the system is presented [chapter 5](#), the *alphabetization* and knowledge model are explained [chapter 6](#) while the *performance* part is described [chapter 7](#).

Chapter 5

Perception

We adopt in this thesis an anthropomorphic view of a music improvisation software. This view helps us define the first stage of the system as the *perception* entity. By perception we mean every process which has to take place before putting any information in any kind of memory, i.e. before *organizing* in time or in any kind of structure what the system perceives. Indeed organizing elements usually implies that the elements are already, if not meaningful, at least consistent and well defined and delimited — except in very specific domains as *information geometry* as in [Cont 11] for example which try to define elements through their meaning. To build such elements, we have first to examine *what* the system listens to, then *how* it listens to it and form coherent elements from it.

The what is presented in section 5.1, we call it *listening*. It deals with the *streams* that the computer will get. We will explain what we intend by streams, what are their nature and types. We will discuss situations with one or several streams and different use of them as well as cases of synchronized or correlated streams and how we group them to define a musical *input*. We also present the low level extractions of information we can execute on these streams and the adequacy of the extraction to the kind of stream. The how will be explained section 5.2, *analysis*, which explains how from low level descriptions of the incoming streams, we form consistent units that will be passed on to higher level models.

5.1 Listening

As we have seen in the first chapter, creating a computer-based improvisation system which is able to react and interact with other acoustic musicians certainly implies *listening* to what these latter play. Before studying the analysis of information in section 5.2, we first have to decide what the system is able to listen to and *how* we make the computer listen. For our system we will take the *sound* output(s) of (a) musician(s) as **input** stream(s) of the computer.

Some researches as in [Gillian 11] use gesture or motion capture input to learn from a musician. These inputs can be very informative and give relevant clues about the

interactions between several musicians, in particular in the case of collective improvisation. However it implies studying deeply the links between the gestural and the sound realms in musician's performances which is largely beyond the scope of our research. We focused our research on the usage of the output of what a musician consider as its *instrument*. In the case of a purely acoustic instrument, that will be the sound result. In the case of a digital instrument, we may have access to already digitalized and possibly structured information as we will see in this chapter.

We first examine in section 5.1.1, **Inputs Anatomy**, what are the different streams we have access to. Then we discuss what can be their different roles in an improvising system before presenting section 5.1.2 the types of low level information **extraction** we achieve on these streams.

5.1.1 Inputs Anatomy

The scope of our research around music improvisation is not to use secondary or external information as would be the usage of gestural capture of the musician's movement or the usage of specific modified instrument. We rather attach importance to the seamless integration of the computer system to the usual setup of the musician(s). That means that we want also to limit the intrusion of sensors or other accessories on the acoustic instrument and prefer utilizing a conventional microphone which is nowadays very much customary in concerts. This background choice does not however limit the number of audio streams we can feed to the computer, especially in the case of collective improvisation.

5.1.1.1 Streams

A group of musicians improvising in a perfectly dark room could be a purely theoretical acoustic analogy for our listening situation: each musician can refer to and interact with the others only in the sound realm. In real situations we make use of two kinds of input *streams*: we use mainly *audio* streams captured from acoustic instruments through microphones of different types. In the computer domain, that means an analogical signal usually converted to a flow of digital samples of 16 or 24 bits at a sampling rate of 44.1 kHz in an audio interface. When available we can also benefit from musicians playing on digital instruments like keyboards and receive the MIDI output of their instruments as a digital stream fed to the computer. These two kinds of input streams: Audio and MIDI will also be the types of the output of our system.

MIDI Stream In the case of musician playing digital instruments which output a MIDI stream, we can run a purely MIDI version of the system. Digital instruments like MIDI keyboard output a clean and unambiguous digital signal what ever the practical conditions of the playing are (in studio, on stage, solo or with a band...). We will see in the next chapter that in this case, the extraction of information is already done by nature of the MIDI protocol. But the MIDI case will serve as an excellent enlightener for segmentation

issues. Since the beginning of the OMax project, numerous prototypes and ideas have been tested with this purely MIDI setup with Bernard Lubat¹ who mostly plays keyboard and synthesizers.

Audio Streams On the other hand, the audio capture of any instrument may vary very much in quality depending on the microphone, the distance to the instrument, the background, the acoustic of the room... The typical improvisation situation does not allow the use of artificial conditions like sound-proofed rooms or isolation cabins to obtain an optimal capture of the acoustic instrument. Worse, we intend to study and design a computer system enabling to play also on stage which may be the worse acoustic situation but a very likely one for a system oriented toward live improvisation. As explained the brief description of our computer system in the previous part, we extract from the audio stream, information on the musician's playing and musical meaning of these. To do so, we need a focused capture and to avoid getting the sound of other instrumentalists in the same audio stream. Otherwise, we may extract inaccurate information. It typically implies, whatever the acoustic instrument is, that a distant aerial capturing would get too much sound from the background to be able to properly isolate the desired information. Conversely, as we record the sound of the musician, not only to analyze it but also to recombine several parts of it, very close capture may be of poor quality when replayed through the speakers. We explored to main ways to deal with this problem:

- splitting the capturing of the acoustic instrument into two separated audio streams, using two different microphones: in most cases a contact microphone will be a very good device to get an extremely focused stream and its quality may be enough to extract relevant information. We will use in this cases a second aerial and farther microphone to record the sound stream that we will use for playback. This solution is illustrated [Figure 4b](#)
- using a focused aerial microphone (hypercardioid for example) as a compromise and put it close to the acoustic instrument. With this solution we have to accept the irreducible background capturing and a lesser recording quality especially in low frequencies but we the analysis results are still good and it is a much more adaptable setup.

From these reflexion on input streams, we can sketch the simpler and most typical situation for our improvisation system: a duet between a musician and a (supervised) computer. This is the situation we have mainly tested along our research, in particular with several saxophonists for testing sessions, demos, presentations and concerts (see [chapter 11](#)). This setup is the easiest possible with our system — one microphone, a computer, an audio interface and one or two speakers. Naturally we tried with various instruments: clarinet, flute, voice, bassoon, percussions.

¹famous French pianist (and drummer) especially fond of free improvisation with the system for several years

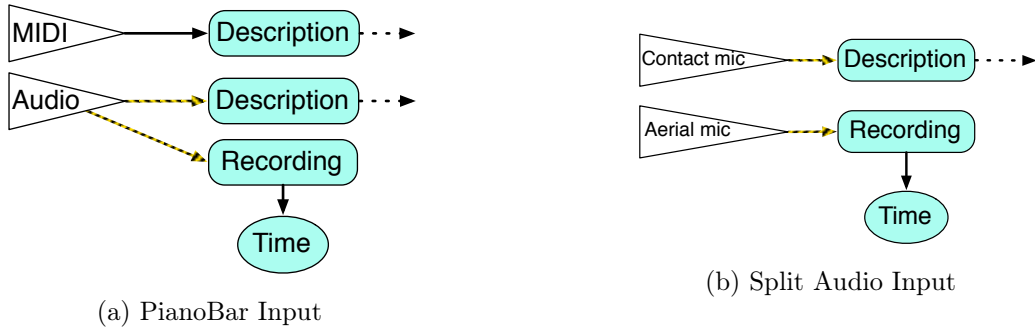


Figure 4: Example of Different Streams Configurations for an Input

Hybrid Input Usually, audio and MIDI streams are used separately because they correspond to different instruments. Either we can capture the sound of an acoustic instrument or we get the MIDI flow of a digital one. One exception is however to mention: we make often use of the PianoBar system developed 2003 by [Moog] when we work with the Piano. This device which can be put on the keyboard of any 88 keys acoustic piano places an optical sensor over each key and detects its moving. It then converts the information to a MIDI format. This way, it creates, as Yamaha MIDIPiano, an hybrid instruments with the full sound possibilities and MIDI data output. In this situation we get two synchronous audio and MIDI streams which both correspond to the same musical discourse as illustrated Figure 4a.

5.1.1.2 Medium

As shown in the previous section about streams, two streams whether audio or MIDI can correspond to the same instrument and captured in parallel. We then have handle them together and the correlation of them may enrich the information that we can extract. They do not constitute two separated musical inputs even though they may have different natures in the case of hybrid inputs. To characterize what constitutes a musical input in the system, we consider its *medium* i.e. the unique, incompressible material that we record, independently from the analysis or extractions being made. In the case of the hybrid input with the PianoBar device, the audio stream of the piano's sound is recorded. This audio recording will be our medium and the MIDI information from the PianoBar extension will be considered as a description on this medium. A similar situation arises in the case of two audio streams captured with two microphones, one is used as the *medium* ie for the recording and the other will be exploited to extract musical information describing the recording. Beyond a vocabulary distinction, it has a practical and important consequence: the common and ground reference of an input, especially constituted with the conjunction of several streams, is constituted by the unambiguous time stamping of the recording.

We can considerably broaden this vision of a musical input constituted by several and distinct streams in the case of collective improvisation. When enabling our system to listen at once to several musicians playing together we reach the sensitive question

of what *playing together* means. When defining *freely improvised music*, [Blackwell 02] states that it relies on:

1. *a distinctive musical language for each improviser, and*
2. *an ability to couple this with the other contributions arising from the ensemble.*

We can see in this definition the duality between the individual musical discourses and the correlation of those to shape the collective result. For our listening computer system, we can reformulate this question into: if we want the system to listen to two musicians playing together, do we consider them as two separated inputs or are they two stream participating in one global musical discourse that we can consider as global input ? This question emphasizes again the importance of the distinction between our *inputs*, *streams* and *mediums*. While the streams correspond to actual flow of raw data arriving to the computer, their correlation depends on whether they share a common medium or if they correspond to two distinct inputs. We will relate in [chapter 11](#) musical situations in which this question has been a crucial concern to enable the machine to participate to the collective improvisation and we will detail the practical solutions we adopted. [Figure 5](#) summarizes our anatomy of an input up to here. One real and usual implementation of

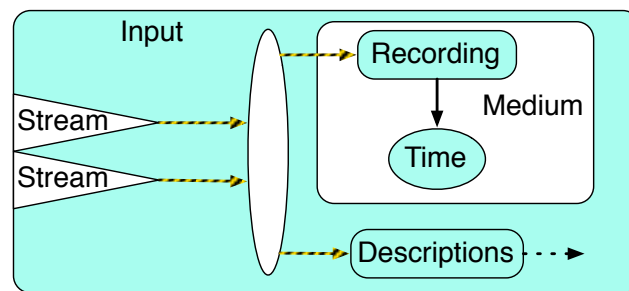


Figure 5: Diagram Summarizing the Anatomy of an Input

this anatomy has been already presented in [Figure 4b](#). In this case, there are two streams coming from an aerial and a contact microphones and the mapping of the streams is direct: the aerial microphone's stream is used for the recording while the contact microphone stream is used for a pitch extraction for example. An even simpler implementation is commonly achieved with only one microphone used both for the medium and the descriptions.

5.1.1.3 Context

The question we reached in the last paragraph can also be expressed in terms of *context*. What is the context of a particular musical discourse? Or in other words, what was going on at the same time. In our anatomy of an input, we showed that correlated streams may be considered together and share a common medium and time and concurrently will be described and used as the material for a new computer generated discourse. There

are however streams that will not serve as the material for a computer generation. It is the case of the rhythmic section of a Jazz band for example. We do not need to extract the same kind of information from these streams. Neither do we need to record their whole sound. However, they give very useful information about the musical *context* of a solo for example. The use of this kind of complementary information will be described in [chapter 6](#). Instead of recording in parallel all the instruments in a massively multitrack input, we take on the hierarchy and roles of each stream and divide them into two kinds: the main streams we described earlier, constituting inputs and which will provide the main musical material to analyze and generate and additional streams which are not mandatory but provide complementary information.

Additional streams are mainly secondary audio streams of various quality, specifically adapted to extract definite information. They may or may not be present in the setup, their absence will not stop OMax to function normally however, their presence will importantly improve the aptness of the computer based improvisation to its context. And their information may be fruitfully shared by several inputs. For example, contact microphones on the cymbals of the drum kit will allow to gather information about the current pulse (see [subsection 5.1.2](#)). Voicings and harmonizations done on the piano during a Jazz chorus enable the same way to extract harmonic context of a chorus. [Figure 6](#) completes the previous anatomy of an input with this context considerations. In our nomenclature,

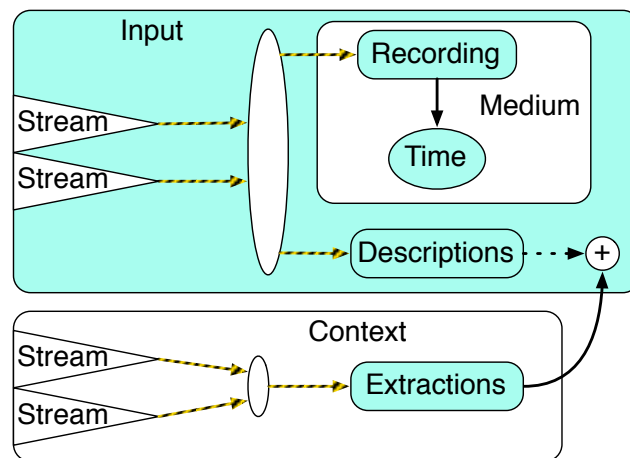


Figure 6: Anatomy of an Input with Context Consideration

we do not consider these streams as part of any input, they may be fruitfully shared by several inputs and the information extracted from them are used both to enrich the knowledge built in inputs and the generation as we will explain in [chapter 6](#) and [7](#).

5.1.2 Extraction

Once we have decided (see [5.1.1](#)) what sound we listen to, we need to decide what aspect of this sound we focus on. The human listening can be global and is certainly able to switch very quickly from one aspect to another depending on several parameters that

we cannot model. However, we can mimic that behavior by extracting several types of information in parallel and relying on the power of the computer to examine all of these angles alongside.

As we explained in section 4.2.3, because the output of our system is not temporally correlated to the modeling part of our system, we have weaker constraints on the time and latency than critical real time applications. Indeed, from the inputs, we extract information to build units (section 5.2) then construct a knowledge model (chapter 6) that we use to improvise in a parallel process. We can thus tolerate latency in the low level extraction of information since we will have to wait, at least, for the end of the current high level unit before being able to add it to our knowledge model. Extraction of information from the additional streams is more critical because we use some of this information to adapt our computer based generation to the current musical context (see chapter 7). If we want the generation part of the system to be a reactive system, this adaptation has to take place as fast as possible. In both cases we have nevertheless a causal constraint that we cannot violate; we have to extract information along the playing of the acoustic musician and do not have access to the future of the sound. Because a revision of the low level information would also mean a revision of all the higher level analysis and modeling build on top, we try to avoid multiple-pass techniques.

We present in this chapter four types of extraction for our system. The first two types, *Pitch* and *Spectra* are base extractions fed to a higher level analysis (see section 5.2) while the last, *Pulse*, is used as contextual information extracted from additional streams (see section 5.1.1.3). The third type, *Chromagrams* serves as a low level extraction for both higher level description and contextual information.

5.1.2.1 Pitch

The first type of low level extraction of our system is pitch analysis. We enable this way our system to have a *melodic* listening. The polyphonic extraction of pitch in music has been a very active field of research especially in the Music Information Retrieval community [Bay 09]. However this community being oriented towards off-line analysis of huge databases of songs and pieces, very few algorithms proposed in this field are causal and implementable in real-time with low latency (medium latency would be enough in our case, see the discussion section 4.2.3) and are versatile enough to work on any kind of instrument without prior training. Although a few papers as [Dessein 10] suggests efficient methods for both off-line and on-line polyphonic transcription, these methods have not lead yet to the publication of an effective and ready-to-use real-time algorithm which prevents us unfortunately to benefit from these research. On the other hand, several techniques are long available for real-time (i.e. low latency) monophonic pitch estimation. In the case of our system, we will firstly listen to musicians separately so can suppose our input is mainly monophonic and work with these kind of algorithms. We make intensive use of the Yin algorithm proposed in [De Cheveigné 02] which is very robust and have

successfully been implemented and used in many applications. This implementation has convenient features for our use. In particular, it outputs a *quality factor* which can be seen as a *confidence indicator* we use (with a threshold) to indicate non-harmonic portions of the material which are not relevant for the pitch description. A parameter of the algorithm also enables to specify the minimal frequency (in Hz) acceptable for the detection, which is a very practical way to quickly adapt to the instrument.

The third parameter of Yin’s implementation by Norbert Schnell is the output period in millisecond. It means that we have control over the rate of the output of this low level description. We typically choose to get a vector of three floating point values every millisecond or every other millisecond, representing the estimated pitch (in Hz), the signal amplitude (between 0. and 1.) and the quality factor of the estimation. We will see [section 5.2](#) how we handle this flow of raw pitches information.

5.1.2.2 Spectra

The second type of audio analysis in OMax uses spectral descriptors to focus the listening on the timbre aspect of the sound. They allow the system to listen and play with wider types of sound including noises, percussions or to discriminate several instrumental playing modes. There have been several studies comparing different types of descriptors for various tasks and in particular for instrument recognition as in [\[Eronen 01\]](#) or [\[Essid 04\]](#). Numerous work on speech recognition make also use of Mel Frequency Cepstral Coefficients [\[Zolnay 05\]](#) and thanks to significant improvements and studies about their implementation [\[Feig 92, Jensen 06\]](#), they have been proven (for example in [\[Brown 01\]](#)) to be easily computable, very compact and reliable for recognition tasks both for speech and instruments. As our main focus is not the feature extraction itself, we decided to use these relatively simple but readily available MFCC descriptors. A first attempt of including these along with linear predictive coding in OMax had been made in [\[Bloch 08\]](#) but the rather complicated implementation used and moreover their very rough quantization made this attempt rather inefficient to foster our knowledge model (explained [section 6.2](#)). Our main contribution around spectral description is developed in [section 5.2: Analysis](#) and has been published briefly in [\[? \]](#).

MFCCs are vectors of floating-point, multi-scale coefficients describing the spectrum — or timbre in musical terms — of small windows of the audio stream. For each vector, the first of these coefficients is a relevant estimation of the overall energy of the slice of audio and the successive coefficients give more details about the shape of its spectrum. We use M. Malt and E. Jourdan’s implementation of MFCCs in the *zsa.descriptors* library for Max/MSP [\[Malt 08, Jourdan \]](#). In this implementation, the size (in samples) of each window is settable as well as the outputting rate of these. We typically choose a window of 4096 sample and an overlap of 4 which means a vector every 1024 samples (our global sampling frequency is classically 44100 samples per second). Converted into milliseconds,

we get vectors calculated over windows around 93ms long every 23ms which is a much slower rate than the pitch extraction.

The number of coefficients for each vector is the number of perceptive bands (mel scale) dividing the spectrum. Speech applications typically use 20 to 30 bands to obtain a very precise description of the spectral content. More bands, and thus more coefficients, allows also to make sound synthesis from this description. We do not synthesize any sound in our processing chain and favor a much lower number of coefficients — usually 10 — to simplify the computational complexity of our next stage analysis (see [section 5.2](#)).

5.1.2.3 Chromagrams

We have recently added the extraction of chromagrams as one of the first stage extraction of OMax. Chromagrams are useful to efficiently approximate a harmonic signature such as in L. Bonnasse-Gahot work to obtain a versatile adaptation of the harmonic context of a melodic event [unpublished]. In our case, instead of using chromagrams only as annotation, we first looked at chromagrams the same way we treated MFCCs for spectra description: as a low level description that may lead to a higher level model ([chapter 6](#)).

Chromagrams, also named *Pitch Class Profiles* [[Fujishima 99](#)] or *Harmonic Pitch Class Profiles* [[Gómez 04](#)], depending of their implementation, have been extensively used to automatically extract chords for labeling and classification — examples are found in [[Harte 05](#)] or [[Gómez 06](#)] among several others, see [[Papadopoulos 07](#)] for a thorougher study — or to extract other various tonal information as in [[Peeters 06](#)]. As MFCCs, it has been implemented for real time extraction in a Max/MSP *external* of the *ircam.descriptors* library, allowing us to use it very easily.

Our chromagrams extraction module outputs vectors of 12 coefficients taking floating point values between 0. and 1. which represent the contribution of each semi-tone of the tempered western occidental scale in the audio signal. They are voluntarily computed over a rather large window — typically of 8384 samples, that is 190 milliseconds at 44100Hz — compared to the spectral description, to avoid the (harmonically) undesirable effects of transients and attacks. As MFCC vectors, Chromagram vectors are overlapping and typically output every 46 milliseconds (every 2048 samples). More precise chromagram vectors of 24 or 36 coefficients exist to consider quarter tones or finer tunings. But as with MFCC vectors we prefer a smaller number of coefficients to reduce the complexity of the next stage ([Analysis](#)).

5.1.2.4 Pulse

Pulse extraction is a whole field of research on its own, far beyond our scope, especially in the context of improvised music which implies causality and real-time discovery. It has been studied for years with various approach either directly from the audio signal [[Goto 01](#)] or with MIDI-like events as in [[Nouno 08](#)]. Thanks to the work of E. W. Large [[Large 95](#), [Large 94](#)] and implementations of it, done by L. Bonnasse-Gahot [[Bonnasse-Gahot 10](#)]

and J. Chao [Chao 11], we favor an easy and straight forward approach to include pulse in OMax.

In Large’s work, pulse is represented through an oscillator which is characterized by a period — the tempo in musical terms expressed in time or in beat per minute (bpm) — and a running phase — which correspond to the musical synchronization. We use a simple attack detector [Puckette 98] and feed the onsets detected into the entertained oscillator which is then able to adapt its period and phase. At the rate of its refreshing — typically every 5 or 10 milliseconds — the pulse extraction module outputs its running phase (periodic between 0. and 1.) and its current tempo.

Our setup for pulse extraction is not robust enough yet to handle correctly non-percussive or very irregular audio streams. But we tested its usage in easy situations, as a Jazz band for example, extracting pulse on the sole *ride* cymbal captured through contact microphones. This way we could focus our research more on the usage of such kind of additional information in the knowledge model and generation and did not try to improve pulse extraction techniques for themselves. Pulse extraction is a typical case of contextual information extracted from a additional stream and enriching the information on an input (see section 5.1.1.3 for the formalization of these cases).

In this chapter we gave a definition of an *input* in OMax, attached to a medium and timeline and we examine the different kinds of *streams* we can handle with the system and which may be associated to form an input or be analyzed separately to from an additional source of contextual information. Then we explained the four kinds of low level information we extract from audio streams: notes, MFCCs, Chromagrams and Pulse. As our first main task in OMax is to model the musical discourse of a musician, we need now to assemble these low level informations to form meaningful units, that is the topic of the next chapter.

5.2 Analysis

We have presented section 5.1 the different kind of low level information extraction we use in our system. To complete these different *listenings* and model a musician’s improvisation, we need to gather the micro-units coming from the different extractions and group them to build consistent macro-units. Deeply tight with temporal succession, these macro-units are contiguous chunks of information which raises meaning when correctly delimited. This is the role of **segmentation** presented as the first part of this chapter (subsection 5.2.1). We will see how we delimit such units depending on the type of stream and extraction. Then we expose a few important **timing considerations** necessary in a real-time system such as ours (subsection 5.2.2).

5.2.1 Segmentation

Segmentation of audio signal, either *on-line* (i.e. with real-time causal algorithms) or *off-line* is again a whole subject of research on its own and has several communities attached to it depending on the main material (multimedia files or purely audio) and the main goal which can be classification (of instruments or speaker) for example or building of *summaries* of files. In particular, a rather large section of the MIR community explores several level of segmentation based on various isolated or combined descriptions of the sound as in [Lu 02] for example. Research as [Foote 00] or [Cont 11], mostly based on change detection also developed methods to combine segmentation and classification in one process. However, to our knowledge, none of these methods lead to a causal implementation without training or prior internal model such as a priori templates of the elements or a score in the case of written music which in our case of improvisation would be irrelevant. Therefore, in our research we kept working on simpler formalism and well-tried segmentation principles. We first segment and timestamp the information flow before considering (in the next part) the *symbolic* labeling and modeling.

Depending on the type of stream and the type of low level extraction performed on these stream, the segmentation stage is necessarily very different. It may consist of a slicing (5.2.1.1) in the case of MIDI streams, which was already implemented for a long time in OMax (and has been ported in our work) but also a more subtle statistical grouping (5.2.1.2) in the case of a monophonic pitch extraction (see 5.1.2.1) which required a new implementation or the setting up of a very new clustering (5.2.1.3) system for the case of vector data as provided by the spectral and harmonic extractions (described 5.1.2.2 & 5.1.2.3).

5.2.1.1 MIDI Slicing

Segmenting monophonic non overlapping MIDI notes is obvious: each note is a unit already described and properly delimited by the MIDI note-on and note-offs. But segmenting Polyphonic MIDI stream requires more work. Indeed, even the very simple case of a legato melody, where one note overlaps with the following, asks the question of where to place the boundary to obtain consistent segments. Is the overlap an interesting segment per se? Do we only consider the onsets as limits of segments and discard the legato? These questions are much amplified in the case of a very polyphonic MIDI stream like the output of a MIDI keyboard. In this case, numerous notes overlap to form the multiple layers of the musical discourse and articulations like legato or grace notes may entangle the musical texture.

In our research we tried to unify the monophonic and non overlapping MIDI case and the polyphonic general MIDI case with one consistent vertical slicing. The question of the legato is actually present in both cases and will be examined section 6.1.2.1. We kept the original representation with *slices* roughly exposed in [Dubnov 03]. Each MIDI onset is a point for vertical segmentation defining the beginning of a slice and the end of the

previous one. To make this mechanism compatible with monophonic MIDI representation, we have to add an exception for silences. Note-offs from the last note sounding have to trigger the closing of the current slice ensuring that if the musician stops playing, his/her last note does not hang in an unclosed slice. However, it is not sufficient to do so and close every slice the moment the last note is released; that poses a problem in the case of a musician playing staccato: by definition of staccato, each note will be followed by an *empty* slice which has no *rest* meaning. In other words, the silence following a staccato note does not have the same musical meaning as the (longer) silence following the end of a phrase or of a piece. Therefore we need refinement again.

Thanks to a time threshold, we will only consider the case of a very long silence (usually above 2.5 seconds), significative of the end of the musician’s playing. In this case only, the slice is artificially closed. For all other cases, we will associate a possible rest following a slice with this slice. This doing presents two advantages for our system: firstly, it allows us to identify two musical patterns in our model even if the playing mode (legato/staccato) is different (this will become clearer with [section 6.1](#) and [chapter 6](#) when we describe our model). Secondly, as we will replay small pieces of our recorded input to create new variations ([chapter 7](#)), we will also replay the silence following any musical element when this was the case in the recording. It is a musically desirable behavior to enable the computer based generation to respect the different playing modes and phrasing of the original material. For example, reinjecting a staccato section when generating, will be also rendered as a staccato passage. We also apply this functioning to the monophonic MIDI and pitch extraction.

We need to take care of another “detail” when slicing specifically a polyphonic MIDI stream: bad pianists and actually even good pianist never attack all the notes of a chord at once precisely at the same instant. On the contrary, micro timing of attacks in the keyboard technique is used by very good musicians to reveal or stress particular notes in the chord for musical reasons. To be able to model the different musical elements of the discourse later on, we do not want clusters of micro slices at the beginning of every chord. Therefore, we need a *tolerance* parameter (in milliseconds) to consider all the note-ons of a chord as one attack only (the first note attacked will then serve as the precise opening time of the slice). We usually set this parameter to 50 milliseconds. [Figure 7](#) summarizes the different cases we described with an illustration using the classical “MIDI roll” representation. Once delimited, these slices will constitute the longest and unbreakable unit that we will consider in our knowledge model ([chapter 6](#)).

5.2.1.2 Pitch Grouping

The second type of extraction introduced in OMax in [[Assayag 06b](#)] and presented [subsubsection 5.1.2.1](#) it pitch extraction based on the low-level Yin algorithm [[De Cheveigné 02](#)]. It outputs a flow of triplets [*pitch*, *amplitude*, *quality*] usually at a rate of 500 or 1000 Hz. We need to gather these rapidly changing numbers into contiguous overall stable section that we will be able to call *notes* and later integrate to our high level model [chapter 6](#).

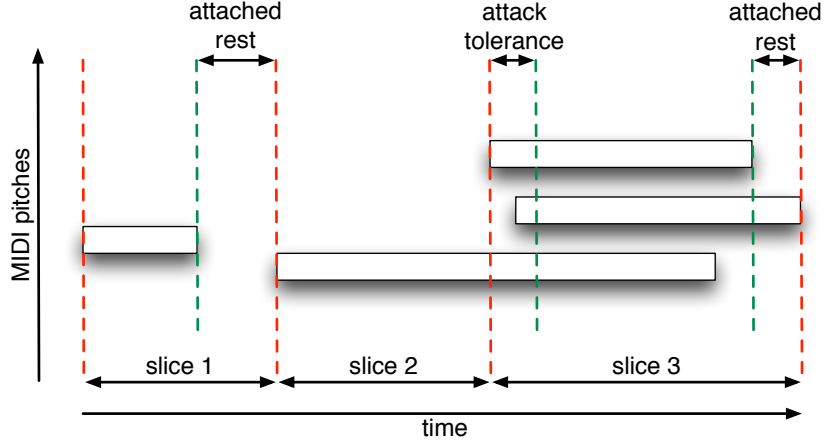


Figure 7: Slicing of Polyphonic MIDI Stream

Our system, based on the C optimization of a patch originally designed by G. Bloch achieves this with two stages. The main idea of the first stage is to compute statistics of incoming pitches over small overlapping time windows and have a minimal appearance ratio (between 0. and 1.) as criterion for what we consider as a *stable* pitch as explained in [Assayag 06b]. Along our work, we had to reprogram in a low level language (C) this mechanism to make it more efficient and enable a handy integration in our modules.

In more details, this first stage works as follows: triplets $[pitch, amplitude, quality]$ are first selected depending on their *quality*. Below a certain quality threshold, the pitch information is considered as inconsistent — musically, it denotes the lack of pitch information in the stream or a transient or attack part — this characterization replaces the *pitch* information by a 0. value to still be considered in the following statistics. Each new *pitch* value arriving opens a small statistical window around 55ms long (adjustable parameter) which will gather every following *pitch* and *amplitude* pairs. For every pair gathered in this time window, if the *pitch* is the same as the one which opened the window, it is added to the participation ratio of this pitch over the window and the amplitude is summed to the other amplitudes. If the *pitch* is different, then the vector is just added to the overall counter of pairs arrived during the window and its content is discarded. At the end of the window, the ratio of pairs of *pitch* equal to the first one divided by the total number of pairs arrived is output (between 0. and 1.) as a *pitch probability* if it is above a *confidence* threshold, and the mean amplitude is also output, naturally. If the *probability* is below the threshold, nothing is output. This mechanism has two major consequences: firstly, as triplets with low *quality* values are replaced with pairs of *pitch* 0., even a relatively small succession of low *quality* information will output a *statistically meaningful* pair $[0., irrelevant\ amplitude]$ and thus denote an off-set information — musically, either the pitch is not stable enough to be considered as a note or we arrived at the end of the previous note. Secondly, as every new *pitch* value opens a new statistical window, we have parallel and concurrent windows gathering the same $[pitch, amplitude]$ pairs but which

participation to each window’s internal statistics is different. Musically, it means that during the attack — the short period during which the note is not stable yet — we have several hypothesis on which note will actually be played and whenever one of them is starting to be statistically prominent, the system outputs it. The time reference for the new note will be the beginning of the window which *won* the statistics. Figure 8 illustrate this mechanism:

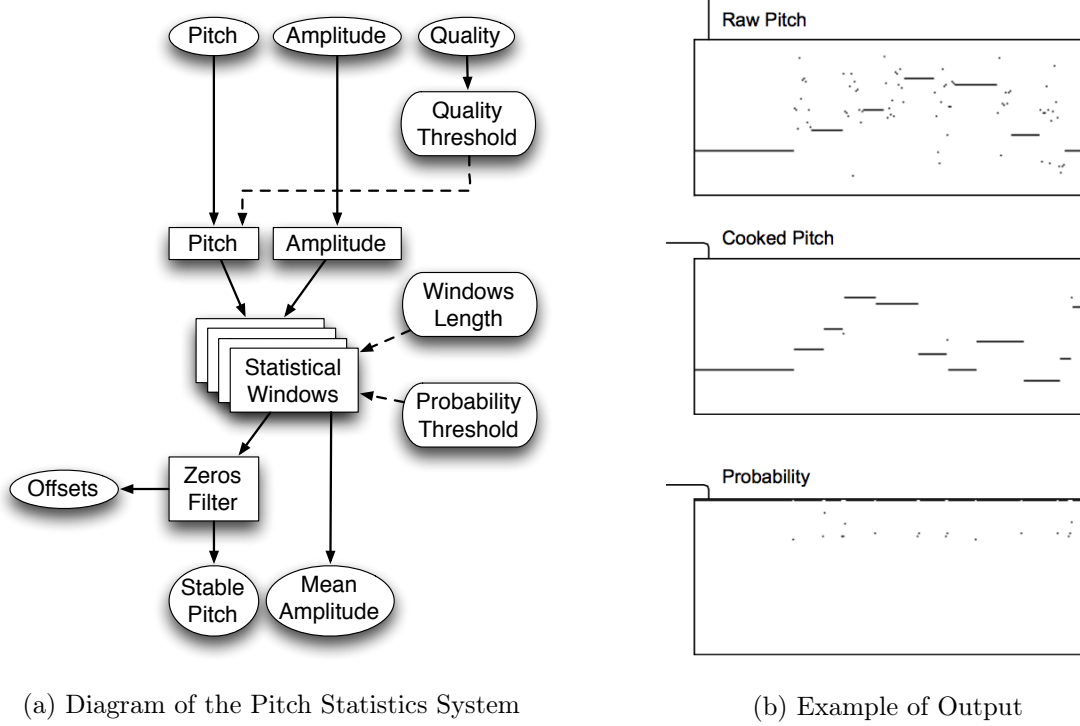


Figure 8: First Stage of Pitch Grouping: Statistical Analysis

The second stage of pitch analysis is done simply by gathering all the successive windows of the same *stable* pitch to form a consistent *note*. The first statistical window output (thus meeting the probability threshold) triggers an onset of the corresponding pitch and the first statistical window with a different pitch or the first offset occurred triggers the offset of this note. Naturally, consecutive offsets are discarded. As a result, the output of this whole pitch analysis is a MIDI-like sequence of non-overlapping onsets with pitches and amplitudes and offsets.

5.2.1.3 Vector Clustering

Vector describing timbral aspect of an audio stream have been tried in our system since [Bloch 08]. At the time, attempts were made with MFCCs and LPC vectors. However raw extraction outputs floating point value vectors from which we need to assemble bigger and meaningful units. In [Bloch 08]’s work, it is achieved with very severe quantification of the coefficients. This raises important problems for both kinds of vectors: MFCCs are multiscale representation of spectra and inside one vector coefficients have

neither the same variation range nor the same importance. Quantizing all them at once (with the same quantization step) as it was done do not create a spectrally *meaningful* classification. The quantization step is either adapted to the first coefficients (which have a much larger variation range) and too big for the latter coefficients or too small to obtain an efficient quantization of the first coefficient. On the other hand, even if LPC coefficients are of similar importance and range inside one vector, their calculation rely on source/filter model which is suitable for voice but may not be as accurate for instruments. More importantly, LPC coefficients, without any other transformation, encode filter coefficients which are very sensitive to variation. Any small modification of one of them may lead to a very different or even unstable filter. Therefore we revised the entire spectral analysis and based it on distance and incremental clustering. Chromagrams which were introduced later along our work also outputs 12-dimensional vectors describing the harmonic content as explained 5.1.2.3. We applied the same analysis principles to classify them.

Distances To aggregate descriptors, instead of quantifying their values, we decided to consider them as real multidimensional vector representation of the timbre and use distance measure between them and clustering algorithm to classify them. For MFCCs vector, we use Euclidean distance while for chromagrams we preferred Cosine similarity. As previously mentioned, MFCCs vectors are a multiscale representation in which the first coefficients represent the overall shape of the spectrum and have a larger variation range than the last coefficients which represent the tiniest details of the spectrum and have much smaller range of variation. Euclidean distance can discriminate between spectra in most cases thanks to this natural property.

However some instruments or some musician’s style sometimes require refined differentiation than the one already imbedded in the MFCC algorithm. For example Michel Doneda’s playing of the saxophone is almost exclusively based on breathing and continuous noise colored through the instrument. There are almost no articulated notes. Therefore the overall spectral shape of his sound is roughly always the same, rich of all frequencies and only subtle differences in narrow bands give his art to hear. We had thus to enhance our distance with weights to discriminate pertinently among incoming vectors. Multiplicative weighting are applied to the vectors before computing their euclidean distance: if $x = [x_1, x_2, \dots, x_M]$ and $y = [y_1, y_2, \dots, y_M]$ are the vectors to compare and $w = [w_1, w_1, \dots, w_M]$ the weights, the weighted Euclidean distance is given by:

$$d(x, y) = \sqrt{\sum_{m=1}^M w_m (x_m - y_m)^2}$$

Preset weighting profiles have been adjusted along our experience with several musicians and instruments. We present a few examples Figure 9. Profile 9a is the default, flat profile which is equivalent to no weighting ($w = [1, 1, \dots, 1]$). Profile 9b ($w = [1, 2, 3, 2, 1, 1, 0.5, 0.33, 0.2]$) has been design especially for Michel Doneda’s playing

i.e. variations of noises and breath sound as mentioned earlier. It enhances the second, third and fourth MFCC coefficients to discriminate among very large band spectra. We also decreased the importance of the last coefficients to eliminate the influence of too tiny fluctuations and thus concentrate our classification on the musical “color” of the large band noise. Profile 9c has been adjusted for low register instruments as Double Bass, Bass

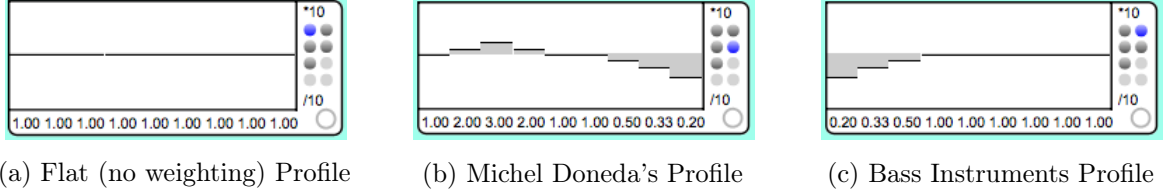


Figure 9: Examples of Euclidean Distance Weighting Profiles for MFCCs

Clarinet or Bassoon which have naturally very distinct spectra switch between playing modes which let sound their low frequencies (normal bowing or blowing for example) and other playing modes which emphasize the higher harmonic frequencies (*sul ponticello* for the Double Bass for example). Therefore we can decrease the influence of the first coefficients which already have large variations and let the higher order coefficients discriminate smaller differences in between close playing modes or timbres.

For chromagram vector, we use Cosine distance given by:

$$c(x, y) = \frac{\sum_{m=1}^M x_m * y_m}{\sqrt{\sum_{m=1}^M x_m^2} * \sqrt{\sum_{m=1}^M y_m^2}}$$

Because of the denominator, Cosine distance includes normalization which is why we use it for harmonic content: the same chord should be correctly identified whether played *forte* or *piano*. Cosine distance also identifies better chords or clusters of notes than Euclidean distance. Here is a example in which we simplify by considering only binary coefficients: a note is either present (value = 1) or not (value 0): if we have the following incoming vector [100010000000] which can represent a major third up from C or a C major chord without the fifth and we compare it to each of the vectors representing isolated notes (ie [100000000000], [010000000000] etc) and to the C major chord vector: [100010010000]. Euclidean distance will declare our input vector equidistant from C, E note and major C vectors (distance = 1.) while the cosine similarity will measure it closer to the C major chord (distance = $\frac{1}{\sqrt{2}} \approx 0.707106$ from C and E notes, distance = $\frac{1}{\sqrt{2}\sqrt{3}} \approx 0.816496$ form C major chord. Higher is closer). This latter result is a better neighboring than the euclidean outcome, the interval being more prominent than individual notes when dealing with harmonic content.

Clustering Once we have decided which distance we use with our descriptors, we have to define how to build our clusters which will in term serve as classification of our spectra. As reviewed in [Jain 99] or [Xu 05], there are many types of clustering with advantages and weakness. In our system we have very strong constraints for the clustering:

1. naturally, we do not know *a priori* what the musician will play.
2. we do not know (or fix *a priori*) the number of clusters. And *a fortiori* we do not have any predefined dictionary of timbres.
3. we need an *on-line* classification to incrementally build our model i.e. we cannot wait until all the vectors have arrived.
4. each vector has to be classified in one and only one cluster.
5. because of the sequential nature of our model we cannot modify the classification of previous vectors. We can only add points to an existing cluster or add new clusters. Otherwise, as we will see [section 6.2](#), we need to rebuild the whole model.

Point 4. excludes hierarchical clustering as well as fuzzy clustering techniques and point 3. necessitate an incremental clustering. Point 1. and the improvisatory and ready-to-use nature of our system does not allow the artificial neural network approach for clustering (like self-organizing feature map or adaptive resonance theory) because of the importance of their training phase. Finally, point 2. keeps us away from most *k-means* clustering algorithms which require the *a priori* definition of k and point 5. forbids existing adaptations of *k-means* with splitting or merging of clusters determining k along the process. Rather than researching in the vast domain of clustering for a rare variation of a complex technique which would suit our needs, we favored a much simpler and older approach: nearest neighbor clustering which has been used successfully for many applications as in [\[Fu 77\]](#) for example.

The nearest neighbor clustering algorithm is expressed in our case by the following algorithm. Naming v_t the incoming vectors, $d(v_i, v_j)$ the distance between any vectors i and j and χ_l the current clusters (initially zero):

1. initialize with cluster χ_0 of center v_0 ²
2. for each incoming vector v_t :
 - (a) let $k = \arg \min_l d(v_t, v_l)$ for all centers v_l of existing clusters χ_l
 - (b) if $d(v_t, v_k) < T$ then add v_t to cluster χ_k
 else $[d(v_t, v_k) \geq T]$ define new cluster χ_l of center $v_l = v_t$

Threshold T is a settable parameter which changes drastically the number, size and consistency of our clusters. In our system, this parameter is left undefined until the very last moments before starting the listening because it may depend on the style of the musician playing or on several aspect of the setup as the background present while

²This initialization depends on the distance used. $v_0 = [0., 0., \dots, 0.]$ is the easiest and correct initialization for Euclidean distance. For harmonic vectors and Cosine similarity, we initialize with 12 predefined clusters corresponding to the 12 binary vectors $[100000000000]$, $[010000000000]$ etc. representing the 12 isolated notes.

capturing. It is also very much affected by the distance weights profiles presented [Figure 9](#) for MFCCs. The proper value of T for a given situation is usually found empirically. An interesting mechanism to set automatically such threshold in a similar system has been developed in G. Surges & S. Dubnov’s PyOracle [[Surges 13](#)].

Finally, once the vectors are classified in clusters, we adopt the same second stage grouping as for pitches ([5.2.1.2](#)): contiguous vectors classified in the same cluster are accumulated to build a longer unit considered as one stable spectra or harmonic content. [Figure 10](#) summarize the processing of floating point vectors in our system. MFCCs processing is presented [Figure 10a](#). Chromagrams vectors processing presented [Figure 10b](#) differs by the lack of weighting and the use of Cosine similarity instead of Euclidean distance. This last differences also inverts the comparison with the threshold T as proximity is maximal when Cosine similarity is higher (closer to 1.).

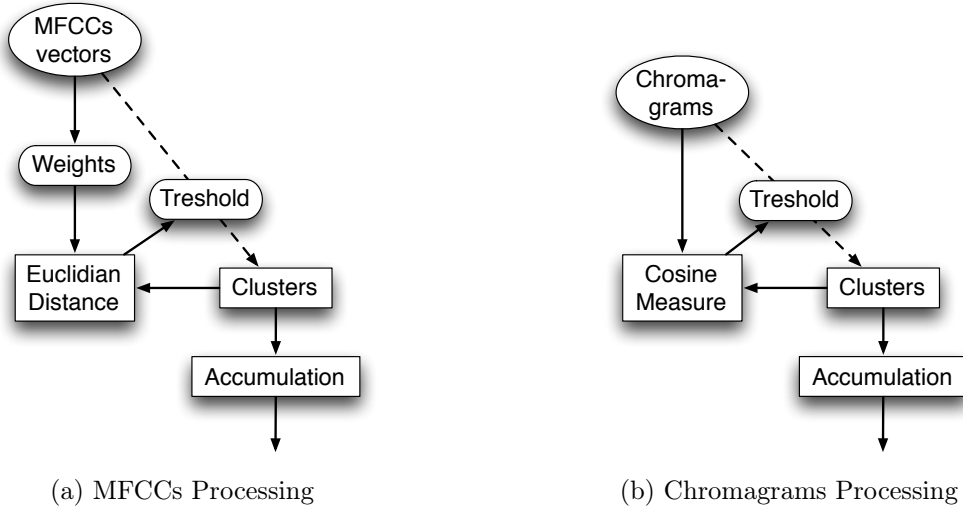


Figure 10: Clustering Process for Vector Descriptors

For each type of stream and extraction, we described our method to build consistent, stable macro-units that we will be able to feed to our system. Before doing so we have to consider the link of these units with time.

Mutations Another behavior of the mechanism for on-the-fly clustering and labeling of vector description is the possibility of mutations according to the past of the learning. Depending on the musical material already encountered and known to the system — which means in our processes, depending on the clusters and labels already defined — the same (or close) vectors can be identified differently. An example of such a mutation is given [Figure 11](#) in 2D with the Euclidean distance representation. Vectors framed in red, although very close, are recognize differently depending on the moment they appear. The vector appearing in [Figure 11a](#) (circled in red) is associated with the “blue” cluster χ_1 because it’s center v_1 is the closest and their distance is under the threshold T . The second

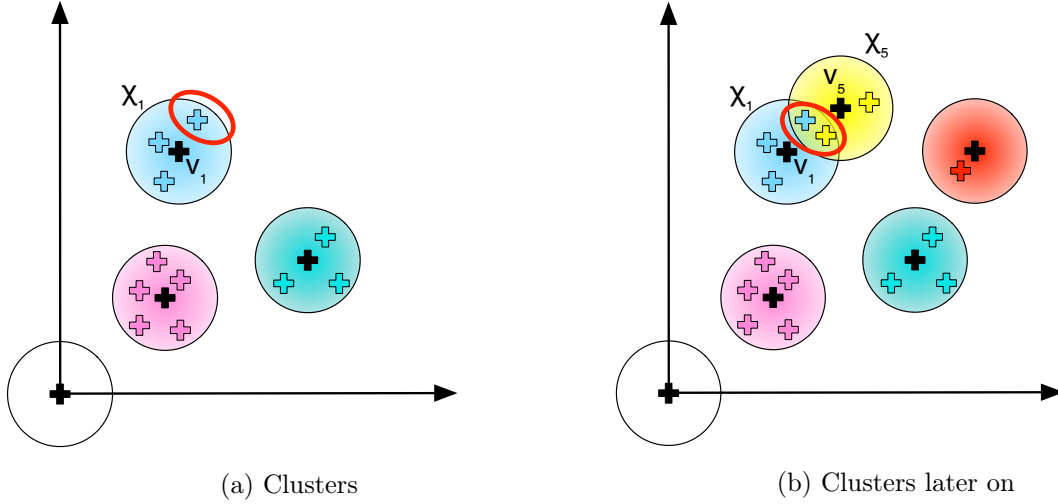


Figure 11: Example of Mutation in Vector Clustering

vector, appearing later on [Figure 11b](#) (circled in red) is closer to the centre v_5 of a more recently defined “yellow” cluster χ_5 . The appearance of the “yellow” cluster χ_5 did not change the class (label) of any of the previously encountered vectors (as specified as one of our very strong constraint on the clustering presented [5.2.1.3](#)) but it modifies the labeling of forthcoming material. This *mutation* mechanism is possible because the measures (Euclidean distance and Cosine similarity) and subsequent clusters and labels we use do not define a univocal *partition* of the vector space along the time (this space usually has 10 dimensions for spectral/MFCCs and 12 dimensions for harmony/Chromagram). The combination of a maximal size for each cluster defined by the threshold T and the nearest neighbor computation allows the “volume” of existing clusters to be reduced (without unlabeled previous elements) by the apparition of new ones bordering them. As the number of vectors extracted for the audio and considered in the system grows, so does the number of clusters and their associated center/label. The overall partitioning of the space thus changes over time with the filling of *empty* regions or interstices. In other words, because the theoretical clusters defined by the “radius” T may overlap, the increasing number of neighboring clusters refines the definition of each of them. This effect has a musical meaning: a specific playing mode can be considered as an accident and identified as an already heard mode (cluster) if encountered only once along the learning. But the same playing mode, if developed more thoroughly by the musician may be rightfully creating one or more new cluster(s) — and letter(s) in our vector alphabet — to describe (and classify) the finer musical differences.

5.2.2 Timing Considerations

In this section, we gather two main considerations which are very different in cause but concern both time handling. The first one, [Time Stamping](#) is inherent to the nature of music which is very tight with time implying that we date our macro-units. The

second remark comes from implementation considerations and finds its cause in computer scheduling and memory usage.

5.2.2.1 Time Stamping

The macro-unit we built from [subsection 5.1.2](#) and [subsection 5.2.1](#) are linked with a timed medium (see [subsubsection 5.1.1.2](#)), most of the time an on-going audio recording. Therefore they encode dated elements along the timeline of medium and we have to include this time reference into the information describing these units. This time stamping or time tagging will also serve, as previously explained, as a common absolute reference. Some of the process described to form the units can be considered as instantaneous as the vector clustering for example while others do necessitate time as the low level extraction or the statistical pitch analysis. The former can be ignored when considering the time-stamping but the latter have to be included in the computation of time references. The date we want to imbed in our macro-units is always the date of the beginning of this unit. Consequently, to compute the proper date, we have to include the different latencies induced by the time-consuming steps. Low level extraction always induce irreducible latency. Pitch, MFCCs and Chromagram extractions are all based on time-frequencies representation processing which requires an observation window, often expressed in samples. The bigger the window is, the more precise the information will be but the more latency we get. To set the pitch extraction window, the lower pitch we need to extract will be the main criterion. For MFCCs extraction, we will favor smaller Fourier Transform (FFT) windows to have faster information. For Chromagrams extraction, we will choose much longer windows to avoid the interference of transients and attacks as explained [5.1.2](#).

In the case of the pitch processing, the statistical analysis described [5.2.1.2](#) gathers information over small time windows, which adds also the windows duration latency to the initial extraction latency. And for all the different extractions, we always group successive micro-units at the end of the processing chain when their content description is stable to get the longest possible macro-units. This grouping has to be also considered in the computation of the beginning date of each unit. All these delays add up to an inherent overall late from the real-time medium unwinding.

5.2.2.2 Recording Management

For obvious memory reason, we do not want to record silence parts i.e. long periods during which the musician we are listening to is not playing. We have therefore a very straight forward silence detector: whenever no data from any of the extraction's processing arrives during a long enough time — typically around 3 or more seconds — we consider that the musician stopped playing and it worth stopping the recording and the learning. We save this way memory and CPU consumption.

However, *starting* the recording with the output of the analysis raises a major problem: because the processing implies latency — as would any processing chain do — triggering

the real-time recording with the output of the analysis will start the recording *after* the arrival of the first event detected and the recorded medium will be missing the audio of the very beginning of the musician’s playing. To avoid this problem and make the detection of activity arrive early enough to trigger the recording without any loss, we artificially delay all incoming audio streams on the recording chain — but not on the analysis chain — so that even with the overall latency, the first element analyzed comes out of the processing at least 100ms before it arrives to the *late* recording. The delayed timeline serves as the common absolute reference of the input. Consequently, when computing the date of a unit for time-stamping we have to subtract the latency of the analysis chain from the artificial delay of the recording, the result being an overall advance on the timeline of the artificially late medium. This computation is illustrated [Figure 12](#).

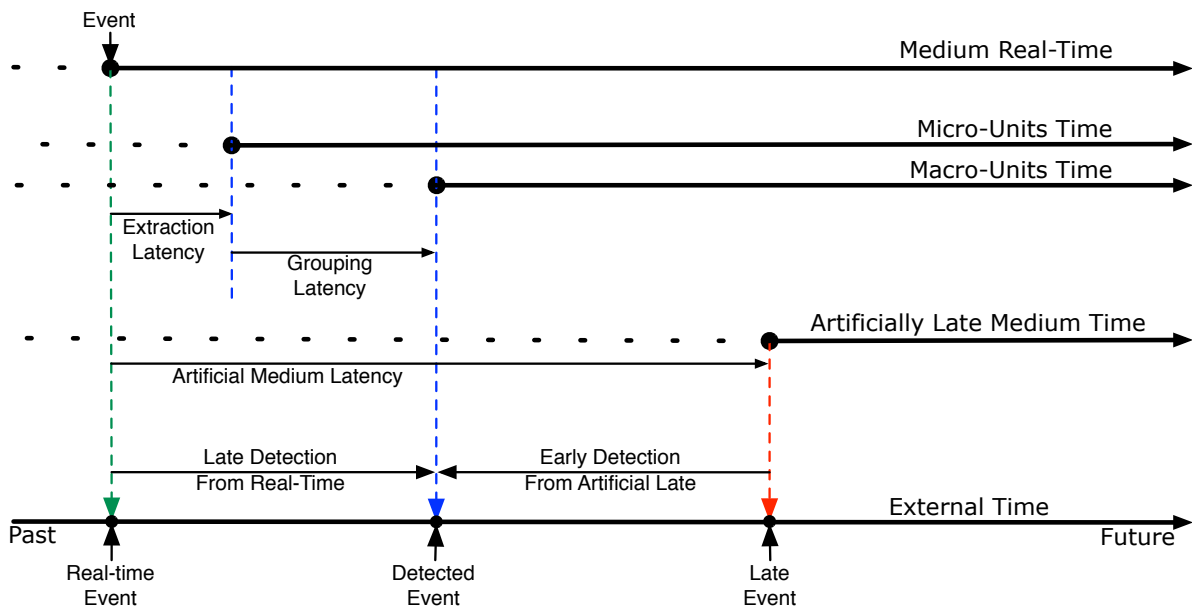


Figure 12: Illustration of the Timing Considerations and Date Computation

For each type of stream and extraction we have presented our process to build consistent macro-units describing the content of the input. Monophonic MIDI stream is constituted of a sequence of self described MIDI notes while the slicing of MIDI stream outputs vertical segment containing multiple MIDI pitches. Pitch extraction is analyzed to form a monophonic MIDI-like sequence of notes and MFCCs and Chromagrams vectors are gathered into coherent multidimensional clusters through a simple on-line closest neighbor algorithm based on Euclidean or Cosine proximity. All of these units arrive properly delimited and dated with reference to the input’s artificially delayed timeline. This overall and fixed delay is no obstacle to our “real-time” objective as the sound output of our system is not directly connected to this learning thread as explained in [chapter 4](#).

Chapter 6

Knowledge

In the brief cognitive considerations of section 4.3, we considered a *processing and decision making* entity named f in [Blackwell 12]’s anatomy of *Live Algorithms*. In our system, this entity is mainly constituted of the graph we use to model music sequences. We present this model in section 6.2 of this chapter. However, this graph, coming from text analysis, requires a strict and unambiguous classification of the elements to model named *alphabet*. We have thus to build this alphabet before building the model. We explain this process in the first section of this chapter (section 6.1). Finally, in the last section of this chapter (section 6.3) we propose new ways to enrich the information contained in the sole graph and thus enhance the whole knowledge of our system.

6.1 Alphabets

Our knowledge model, presented in the next section (6.2) has been invented in the text community and requires an alphabet, usually noted Σ , over which the sequence of symbols is taken. That implies a univocal classification and labeling of our units before learning them in this model. Considering our units, extracted from audio content and even clustered on-the-fly in the case of vector data, this labeling is not trivial and has important consequences on the pertinence and usability of our knowledge model which recognizes repeated patterns of symbols.

Nevertheless our knowledge model do not require defining *a priori* all our symbols, so we could use and test along our work two very different types of alphabets: predefined and evolutive alphabets. Predefined alphabets are a kind of labeling in which we can list all the possible symbols before getting any incoming unit. On the other hand, evolutive alphabets are built incrementally along the apparition of segmented units coming from the analysis chain described in the previous chapter. In this case, we discover the labels on-the-fly as well and constitute our alphabet along the learning.

6.1.1 Predefined Alphabet

Two main descriptions let us use predefined alphabets: monophonic MIDI stream and pitch extraction from monophonic audio stream. In both case, even though we do not know which symbol will indeed appear in the sequence, the “letters” (symbols) of the alphabet are univocally defined beforehand. However, as we will see, this does not mean that there is only one possible alphabet. On the contrary several predefined alphabets are possible to label these two kinds of units.

6.1.1.1 MIDI Pitch

In the case of a monophonic MIDI stream, high level units are directly described by their pitch and velocity, as specified by the MIDI protocol. Therefore, in this particular case, we use the pitch information as pre-defined symbols, close to the classical music score notation. The MIDI note number (encoded through integers) will be our main alphabet. The relation with the frequency (f) of the note is given in the MIDI standard by

$$d = 69 + 12 \log_2\left(\frac{f}{440}\right)$$

and goes theoretically from 0 to 127 . However, it is possible to fold this multiple octave alphabet onto one octave only in this case, we have only 12 pitch classes in our alphabet. We will discuss in [subsubsection 6.1.2.3](#) how the size of the alphabet influences our model.

6.1.1.2 Notes

The case of monophonic pitch extraction give us more opportunity to refine and use different labeling. The first and natural alphabet we used is the same as the monophonic MIDI alphabet: semi-tones encoded for example by the same integers (note number) as in the MIDI standard. We will see in the last part ([Performance](#)) that the usage of this alphabet in the case of monophonic pitch extraction adds the possibility of generating a MIDI output even when our input was an audio stream. As with monophonic MIDI, we can also fold this MIDI note number alphabet on one octave and obtain a 12 pitches classification.

More interestingly, as developed thanks to Steve Lehman, saxophonist from the contemporary New-York Jazz scene who came to work with us during the summer 2011, we can refine our audio notes alphabet by dividing it into quarter-tones which are very much used in contemporary music composition and improvisation. This multiplies our number of pitch classes by two and allows very precisely tuned patterns appreciated by several musicians. Again, this alphabet can be folded onto one octave and reduced to 24 pitch classes. Steve Lehman, along our collaboration, even refined this alphabet to fit very accurately his saxophone fingerings which do not actually allow exact quarter-tones.

We defined this way four different labeling for pitch extraction on monophonic audio stream: semi-tones, semi-tones on one octave, quarter-tones, quarter-tones on one octave.

6.1.1.3 Silences

As explained in 5.2.1.1 about the slicing of polyphonic MIDI streams, we do not want to consider very short silences as autonomous macro-units because they may be part of the instrument's playing mode and may avoid symbolic patterns to be properly recognized. However, longer silences which can correspond to the end of a musical phrase may be considered as notable musical elements that should be learnt in the sequence model. To do so, we need to define at least one *silence symbol* which will encode each resting period long enough to be considered as a musical entity but short enough not to trigger the stopping mechanism described 5.2.2. This way we can add one symbol to our predefined alphabets and enable the recognition of phrase endings and beginnings.

However using only one common silence symbol may cause false or unwanted identifications of patterns. For example — using the anglo-saxon alphabetical notation of pitches and adding an 'S' letter for silences — sequences like A-B-C-S-A which denotes a phrase ending by A-B-C, then the beginning of a new phrase by A, will be identified with sequences like C-S-A-B-C denoting a phrase finishing by C and the starting a new phrase by A-B-C. This identification is musically arguable because the end a phrase is a strong musical element which disconnects two parts of the musical discourse. The duration of the silence is not taken into account and the phrase beginning A-B-C of the second sequence may have almost nothing in common with the beginning of the phrase in the first sequence (we only know it starts with an A). Therefore, following a reflexion with L. Bonnasse-Gahot, we implemented a second refined representation of silences: to be able to consider the strong *caesura* role of rests in our model which is strongly related to succession, we encode each silence with a unique symbol. Thus sequences like B-C-S-A-B and B-C-S'-A-B are not represented identically ($S \neq S'$) and we preserve the separation role of silences in the sequence model.

The unforeseeable number of silences, each represented with a unique symbol leads us towards our second type of alphabet: evolutive alphabets in which the labels are defined along the segmentation of macro-units.

6.1.2 Evolutive Labelling

As our knowledge model does not need the *a priori* definition of the equivalence classes, instead of predetermining them, we can label our units along their discovery in the audio. The first example of such evolutive alphabet is given by the polyphonic MIDI slices labeling. Then we use extensively this on-line labeling for the vector clusters.

6.1.2.1 Virtual Fundamental

As previously explained 5.2.1.1, the segmentation of polyphonic MIDI stream outputs vertical slices as macro-units which contains a collection of MIDI pitches. To classify (strictly) these slices, we need to reduce these collections of pitches to a single label.

Several chord classifications are in use in different domains: classical harmony treaty with tonal functions, Jazz chords classifications with additional bass note, mathematical music analysis with full unambiguous classification. They are often incompatible with one another and sometimes ambiguous. But more importantly they suppose the *a priori* knowledge of rules and information about the context (main tonality, tonal role of the chord etc.). As we discussed in the [Introduction](#), we do not want such an *a priori* in our system because we assume nothing about the musician’s style. Therefore we oriented our research towards purely content-based algorithms to classify the MIDI polyphonic slices. We first ported a recursive algorithm from OpenMusic [[OpenMusic](#)] which computes the approximate *greatest common divisor* of all the present frequencies of the slice. We assumes this frequency is the bass note or *fundamental* of the chord — possibly missing in the chord — and we use this frequency value (rounded to the nearest multiple of an adjustable *step* parameter) as *ad-hoc* symbol for our alphabet.

Later on, thanks to G. Bloch’s inspiration on MIDI slices description [unpublished], we adopted a more refined 3-steps algorithm. Coming from a classical background, G. Bloch postulates the following: if there is a fifth interval in a chord, then the fundamental note of the chord is most certainly the lower note of the fifth, conversely, if there is a fourth interval in the chord, then it probably correspond to an upside down fifth, and the fundamental note of the chord is supposed to be the higher note of the fourth. We implemented in our system his mechanism to find fifth, then fourth interval (if no fifth is found) and default to the virtual fundamental algorithm previously describe if there are no fifth or fourth.

In this second variation, we reduce polyphonic MIDI slices either to one of the note supposed to be prominent in the chord or to a virtual note which may not be present in the chord but is related to all other notes of the chords thanks to the greatest common divisor algorithm. As the notes of the chord are already encoded through MIDI pitches (see [6.1.1.1](#)), we also round the output of our virtual fundamental algorithm to the nearest MIDI pitch. Thus instead of having a real evolutive alphabet, we fall back on the predefined MIDI notes alphabet to encode the description of each slice. And we also have the same possibility to reduce our alphabet to one octave and get a 12 semi-tones alphabet.

6.1.2.2 Cluster Centers

As presented [5.2.1.3](#) the two vector descriptions — spectral description with MFCCs and harmonic description with Chromagrams — are aggregated with a nearest neighbor clustering algorithm to form macro-units. While doing this incremental clustering, new clusters χ_l are always associated to the first vector v_l which distance to all the other clusters is above (or below, depending on the type of distance used) the threshold T . As our clustering algorithm does not use mobile centers opposite to *k-means* algorithms for example (by definition of *k-means*), this first vector v_l will always remain the center

of the cluster χ_l . Therefore we can use the index l as abstract symbol for our labeling. Each forthcoming vector classified in the cluster χ_l will be labeled by l in the alphabet. As a consequence and for very practical reasons, what we actually store and grow as symbolic alphabet of vector is a dictionary of all the coordinates of the centers v_l associated with their index l (for easy retrieval). This center’s dictionary is also used for distance computation and we output at once, the distance to the nearest neighbor and the label (index l) of the corresponding cluster. Figure 13 illustrates in two dimensions the vector clustering and labeling, with centers and classified vectors for both Euclidean distance, best represented with discs in the plan (13a) and Cosine similarity best visualized with circular sectors (13b).

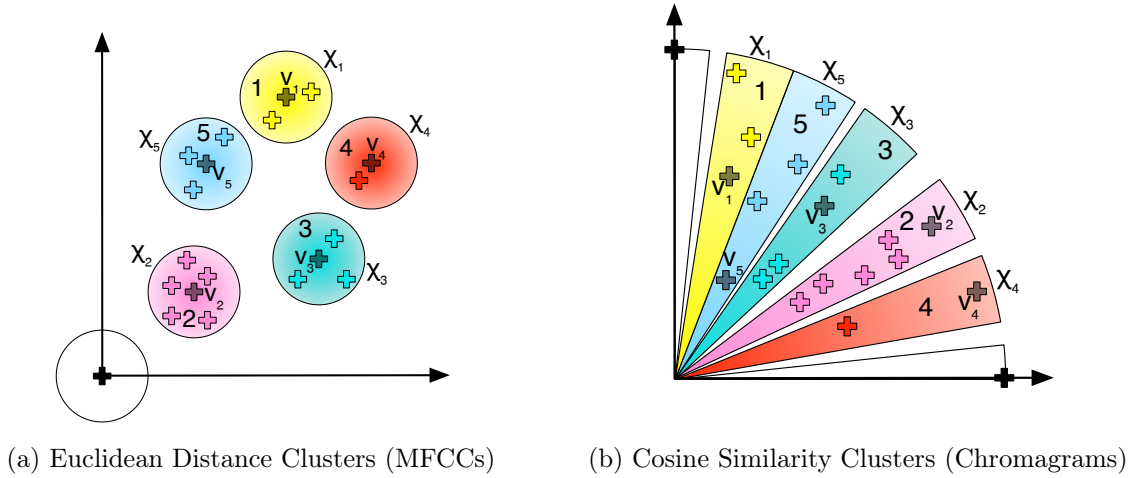


Figure 13: Representation of Clusters in 2 Dimension with their Center and Label

NB.: for Cosine similarity, the “center” of the cluster is the central axis of the circular sector (in 2D or cone in 3D).

6.1.2.3 Alphabet Size

We have just exposed several ways to label our macro units and get a sequence of symbols or “letters” to feed to the knowledge model. For each type of streams and descriptions, we show that there are several possible labeling, either predefined or incrementally discovered. When choosing what labeling we are about to use and the parameters of this labeling (possibly related to the parameters of the clustering as well), like the octave folding or the threshold T for example, we directly or indirectly but strongly influence the number of symbols in the alphabet used in the model. The knowledge model we use ([Factor Oracle](#)) is based on *repeated* pattern recognition in a sequence of symbols. Therefore, there is a very important trade-off between the number of symbols in the alphabet and the number and length of identical patterns potentially appearing in the sequence and recognized by the model. Indeed, the more “letters” we have in the alphabet, the more precise the symbolic description is, but the less likely patterns are to appear more than one time identically in the whole sequence. The extreme situation in this direction is an overly

large alphabet in which each symbol describes very precisely and uniquely each musical element appearing this way only once in the whole musical material. In this case, no pattern is ever repeated — each pattern appear only once — and our memory model is useless. Conversely, if our labeling do not discriminate musical elements enough — that is, if very different musical elements are labeled with the same symbol — we will get very recurrent patterns recognized in our model which actually correspond to very distinct musical content and our model will be too general to enable smooth and sensible musical variations.

This balance between the size of the alphabet and the richness and pertinence of the patterns recognized in the knowledge model is our main empirical criterion to adjust the spectral profiles and the threshold T of the vector clustering presented 5.2.1.3. Along our numerous musical experiences with the system (see chapter 11), we determined that keeping the size of the vector description’s alphabet between 300 and 600 “letters” is a good compromise to get both fine description and musically meaningful repetitions of patterns in the model.

6.1.3 Summary

Before describing in more detail our knowledge model, let’s summarize the whole chain from the input stream to the sequence of labels. Figure 14 recapitulates for each type of streams and each step the different characteristics and parameters.

The listening part of our system described 5.1 includes the framing and extraction of micro-units which is relevant only in the case of audio stream. On the streams, we have three types of extraction: pitch extraction, spectral description and harmonic content characterization. These three types of listening start with a Fast Fourier Transform (FFT) defining the smallest observable element. Then known algorithms are applied: Yin for pitch extraction [De Cheveigné 02], Mel-Frequency Cepstrum Coefficients for timbre description [Fang 01] and Chromagrams for harmonic content characterization [Fujishima 99]. The micro-units containing these descriptions are aggregated into bigger segments considered as stable thanks to statistical analysis processing: straightforward statistics for pitch information, nearest neighbor clustering for vector (spectral and harmonic) descriptions. For MIDI streams which are event based, note-ons and note-offs events serve as boundaries to delimit the stable segments. Then symbols are computed to label the content of the macro-units. MIDI note numbers are primarily used to label monophonic MIDI and pitch extraction. It can be refined by using a quarter-tone scale in the case of note extraction but also reduced to one octave only to decrease the number of “letters” in the symbolic alphabet. The consideration of the most harmonically prominent note in polyphonic MIDI slices is complemented, if needed, by a recursive greatest common divisor algorithm to reduce each polyphonic MIDI slice to its virtual fundamental. The centre of each vector cluster is used as label in the case of vector (spectral and harmonic) descriptions. Finally, considering the different latency of each step of the chain (in particular, statistical window and frame sizes), macro-units are time stamped to unambiguously link them to a musical

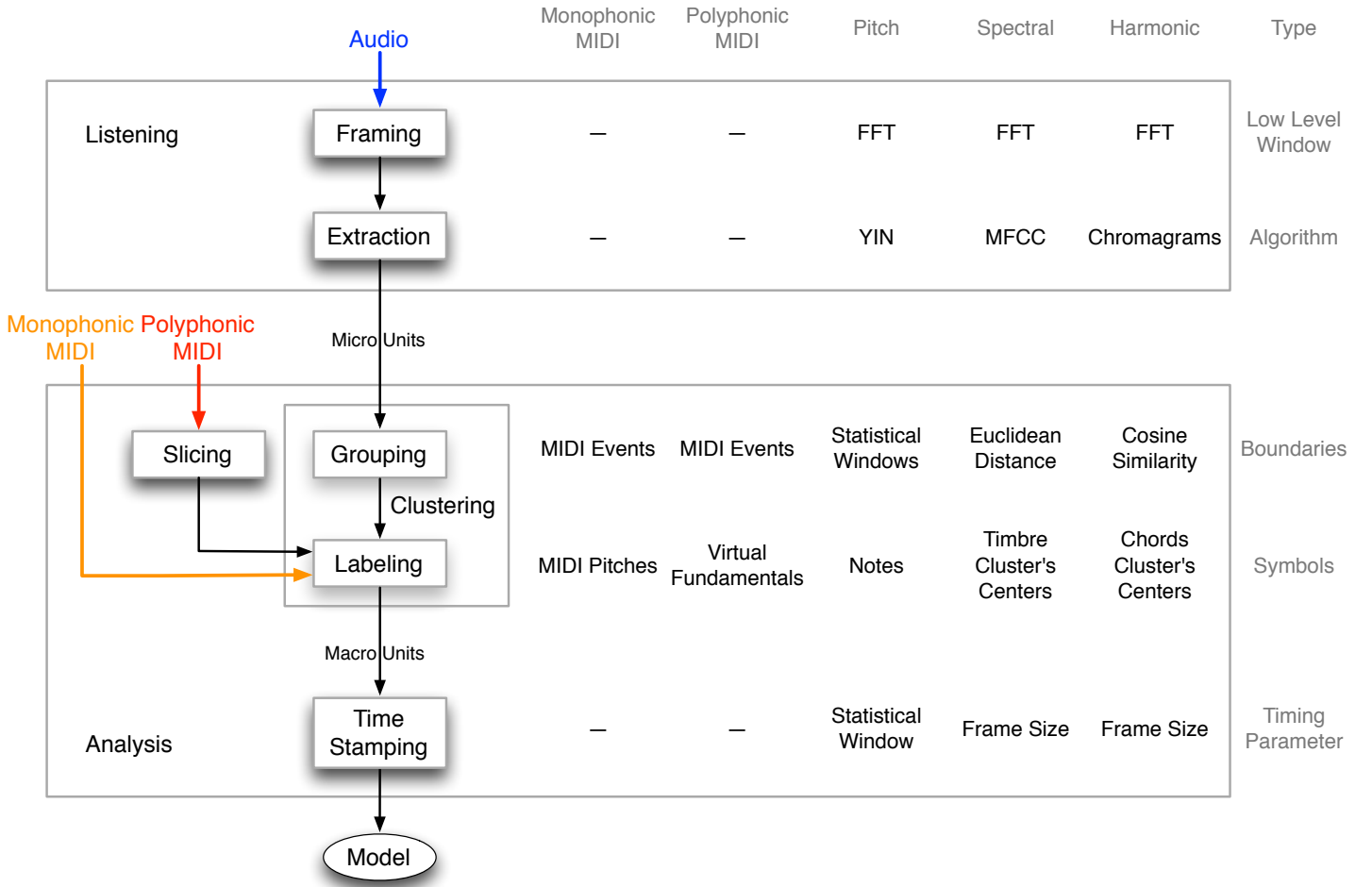


Figure 14: Description Chains up to the Labeling per Streams and Descriptions

element of the input. The sequence of those symbolic elements, linked with their musical content and description is ready to be learnt in our knowledge model.

6.1.3.1 External Analysis

In addition to the five types of description included in our system that we presented in the previous sections and summarized [Figure 14](#), we can also make use of external analysis processing provided that this processing segments and labels univocally a MIDI or an audio stream. In particular, thanks to the work of N. Obin [[Obin 11](#)] we used non real-time prosody analysis made on speech recordings. N. Obin's process to analyze speech material is done in two steps close to the extraction and analysis elements of our system. First, a segmentation is achieved to obtain proper syllables boundaries [[Obin 13b](#)], then a modeling of the syllables prosodic contour is done to classify them [[Obin 13a](#)]. This process outputs a text file containing, in a tabular form, the beginning and ending date of syllables and encodes with very condensed and univocal labels the prosodic contour of each syllables. It also includes silence labeling. Thus we can use this information

describing precisely speech material exactly as one of the description included in our system and previously described. We substitute this result of the prosodic analysis to the online analysis chain of our system and feed *offline* to the memory model with the syllables contours as labels. Our system is then capable of generating a new speech stream with respect to the prosodic characteristics of the original material.

6.2 Memory Model

As briefly introduced in [section 4.1](#) and thoroughly explained in [\[Assayag 04\]](#), G. Assayag and S. Dubnov founded their research around knowledge models for music on statistical modeling [\[Conklin 03\]](#) and tried several kinds of models from classical Markov Chains to Variable Markov Chains implemented in the form of Incremental Parsing (IP, derived from [\[Ziv 78\]](#)) and Probabilistic Suffix Tree (PST, suggested in [\[Ron 96\]](#)). They eventually began to show the optimality and efficiency of a formal model called Factor Oracle [\[Allauzen 99a\]](#) for sequence modeling in music. Extensively using this structure in our work as well as new formal developments added to Factor Oracle afterwards, we present in details this model in this Section.

6.2.1 Factor Oracle

Factor Oracle has been invented 1999 by C. Allauzen, M. Crochemore and M. Raffinot as a pattern matching structure competing with suffix trees , Directed Acyclic Word Graph (DAWGs), factor automata and other known structure [\[Allauzen 99a, Allauzen 99b\]](#). They provided from the first article a very efficient incremental algorithm to build this graph and carefully demonstrated its correctness and usage for string matching (with what they called Backward Oracle Matching or BOM). Naturally, this structure has been used in the text community for example in [\[Kato 03\]](#) and algorithms based on Factor Oracle have been improved and extended over the years (in [\[Cleophas 03\]](#) or [\[Faro 08\]](#) for example). In particular, a very recent and music oriented extension of Factor Oracle has been proposed in [\[Maniatakos 12\]](#). Interestingly, the algorithm has been used very efficiently also in the genetic community, especially for DNA sequencing and finding specific meaningful subsequences in proteins or other sequences. An example of such use named *FORRepeats* can be found in [\[Lefebvre 03\]](#).

Thorough explanations of the building algorithms and explicit examples of Factor Oracle construction can be found in several articles starting with [\[Allauzen 99a, Allauzen 99b\]](#) but also in [\[Assayag 04\]](#), [\[Assayag 07\]](#), [\[Dubnov 07\]](#)... so we will not recall these here. We will rather detail the structure and properties of the Factor Oracle graph itself (once built) along the following sections with specific focuses on properties that are important for our use in a music improvisation system. We will illustrate these properties on the classical letters sequence : abaabacba.

6.2.2 Linear Skeleton

The first two instructions of the main function presented in the incremental algorithm of [Allauzen 99a] to add a letter σ to the Oracle of $p = (p_1 p_2 \dots p_m)$ to get the Oracle of $(p\sigma)$ are the following:

1. Create a new state $m+1$
2. Create a new transition from m to $m+1$ labeled by σ

This construction means two important properties:

1. for a word (or input sequence) p of total length m , there are $m + 1$ states in the graph
2. the skeleton of the graph is made by m transitions from state $i - 1$ to i ($i \in \llbracket 1, m \rrbracket$) labeled by the actual letters of the sequence

In other words, the skeleton of the graph (states and basic transitions) exactly represents the original input sequence. The illustration of this skeleton is given Figure 15 for the letters sequence abaabacba. This is an important property when modeling musical

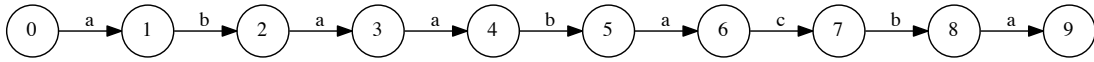


Figure 15: Skeleton of the Factor Oracle Graph of the Sequence abaabacba

improvisation sequences and learning the style from them because we do not destroy the time development of the musical material learnt opposite to several model like Markov Chains for example which does not enable to (re)generate surely the original sequence once the model is built. Our knowledge model includes as foundation the original discourse and we can choose to reproduce it literally if needed. This remark is also pertinent in regards with our principles (exposed section 4.2) in which we assume that the musical input is our ground truth, i.e. the choices of the musicians and his or her discourse is supposed to be always right so that the system has to learn from it and to analyze the implicit rules justifying it.

6.2.3 Suffix Links

In their article [Allauzen 99a] defining Factor Oracle, the authors define $\text{repet}_p(i)$ as *the longest suffix of $\text{pref}_p(i)$ that appear twice in $\text{pref}_p(i)$* . $\text{pref}_p(i)$ is defined earlier in the article as *the prefix of length i of p for $0 \leq i \leq |p|$* . Then along the on-line construction algorithm of Factor Oracle, they make use of a *supply function* $[S_p(i)]$, *that maps each state $i > 0$ of $\text{Oracle}(p)$ to state j in which the reading of $\text{repet}_p(i)$ ends*. They note about this function that:

$S_p(i)$ is well defined for every state i of $\text{Oracle}(p)$.
For any state i of $\text{Oracle}(p)$, $i > S_p(i)$.

After that, Lemma 7 of the article states the following property:

Lemma 7 *Let $k > 0$ be a state of $\text{Oracle}(p)$ such that $s = S_p(k)$ is strictly positive. We denote $w_k = \text{repet}_p(k)$ and $w_s = \text{repet}_p(s)$. Then w_s is a suffix of w_k .*

In a more natural language, this construction and properties means the following: each state i of Factor Oracle of the word p has a *backward* link $S_p(i)$ starting from state i and pointing to the first occurrence of the longest repeated suffix present at state i . This link is named *suffix link* and connects this way every pattern (sub-sequence) of the input sequence to the first occurrence of the longest suffix of this pattern previously encountered in the sequence.

In a second article by A. Lefebvre and T. Lacroq published a year later [Lefebvre 00] a novel information is studied and added to the Factor Oracle algorithm. With a smart backtracking of suffix paths (successions of suffix links), the authors show how to find with Factor Oracle the lowest common suffix “ancestor” of two arbitrary patterns (sub-sequences) of the initial word p and define the *length of repeated suffix* abbreviated in *lrs*. This new elements gives explicitly for each suffix link the length of the common suffix between i and $S_p(i)$ with two simple and incremental functions added to the original construction algorithm. It means that for each pattern of the original sequence, Factor Oracle graph with *lrs* improvement, gives both the places and the length of the first occurrence of the longest suffix of this pattern already encountered in the past of the sequence.

Two years later, A. Lefebvre and T. Lacroq with J. Alexandre improved their heuristic to compute the *lrs* and published the new algorithms in [Lefebvre 02]. We took care in our work to add this improvement to the version of the algorithms that we extensively use to build Factor Oracle on music data. On Figure 16 we added these suffix links labeled with their *lrs* to the skeleton of the same letter sequence as before: abaabacba.

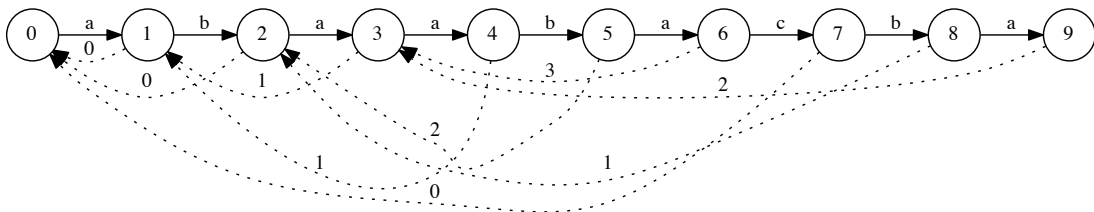


Figure 16: Suffix Links of the Factor Oracle Graph of the Sequence abaabacba

Even though there were thought at first as construction links, on the musical level (as well as in genetics as applied in [Lefebvre 00] and [Lefebvre 02]), the suffix links are very informative. Indeed they connect two repeated chunks of the material and the *lrs* gives the maximal length of the repeated portion. The ensemble of all the suffix links constitute thus an extended map of every repeated pattern in the musical discourse and allows us to model the succession and use of new or already heard material and let us study the way these patterns are connected one to another.

6.2.4 Forward Links

As a properly defined automaton (in which all the states are terminal), Factor Oracle of word $p = p_1p_2 \dots p_m$ on top of its skeleton and suffix links also consists of *forward transitions* labeled by letters of the original word. These transitions allows to generate all the factors of the input word p (patterns or sub-sequences of word, see [Allauzen 99a] for a rigorous definition). For our use, we differentiate the *forward transitions* from the skeleton with the fact that they do not connect two consecutive states. However it is to be noted (as done in [Allauzen 99a]) that every transition arriving to a state i , including the one from the skeleton is labeled by p_i i.e. the letter of the original word. Illustration of these forward links (added to the other links of the graph) is given Figure 17 for the same letter sequence as before: abaabacba.

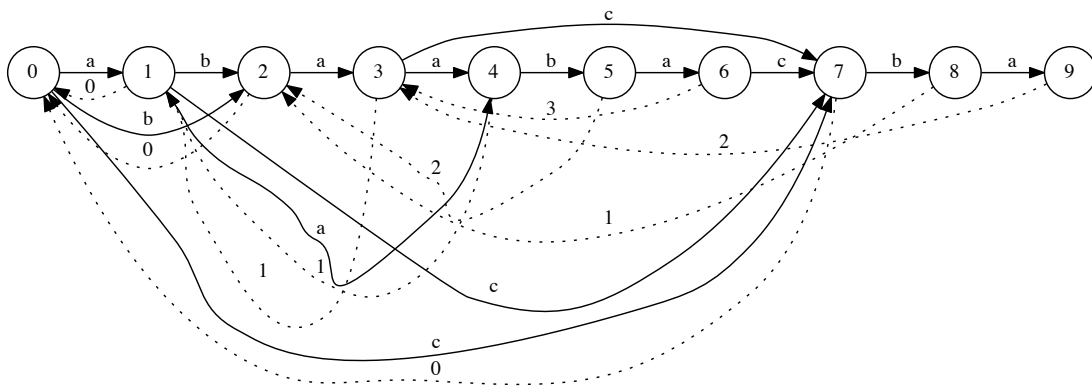


Figure 17: Complete Factor Oracle Graph of the Sequence abaabacba

As an automaton — thus characterized by the language it recognizes —, Factor Oracle is known to recognize at least all the factors of the input word. But it recognizes more patterns than only the factors existing in the input word. Examples are given in all the articles which explain its construction. Attempts have been made to analyze the language Factor Oracle recognizes as in [Mancheron 04] but no *simple* characterization has yet emerged.

Even though it could be used for music generation on the symbolic level, forward transitions (excluding the skeleton), on the contrary to suffix links which ensure a common pattern, do not embed any actual realization of patterns of the input sequence. In other words, navigating the Factor Oracle with usage of forward transitions allows to generate symbolic factors of the original sequence (and more) but do not correspond to actual chainings of the musical material learnt. They permit to find efficiently certain patterns though as originally and extensively developed in [Allauzen 99a]. Because our concern is the realization of smooth and pertinent variations on the musical level thanks to the underlying Factor Oracle model, we do not use forward transitions in our system. This has been however discussed in other musical contexts as in [Cont 07], [Dubnov 07], [Cont 08], [Assayag 09] and [Assayag 10] for example.

6.3 Enriching Memory

6.3.1 Annotations

The goal of our knowledge model is to provide consistent and smooth variations possibilities on a learnt material. The **Factor Oracle** graph provides a powerful structure for such variations, however the choice of the musically pertinent variations at a given time in the improvisation can not solely rely on the graph properties. Indeed once we have gathered a collection of acceptable variations with equivalent *smoothness* — thanks to the suffix links of the graph and their *lrs* (see 6.2.1) —, we still have to pick one of these connections to effectively play. For that we need additional information complementing the data already included in the **Factor Oracle** graph. These *annotations* can be extracted from the input stream(s) or from additional streams as explained 5.1.1.1. Instead of trying to extract more or different information from these stream — which would automatically increase the CPU consumption and the amount of information to handle and store, we can use the primary extractions we already described in Section 5.1.2 i.e. pitch, spectra, chromagrams or pulse. However to efficiently enrich the information about the possible variations, we need either to cross information types, that is to use as annotation an extraction type which has not been used to built the Factor Oracle on which we are currently improvising. This is explain in the next section, 6.3.1.1. Or we can derived from the same primary extraction, additional descriptors which did not serve in the analysis process leading to the Factor Oracle building. This mechanism is described section 6.3.1.2.

6.3.1.1 Information Crossing

Considering the information we already extract from the incoming streams, we have real-time descriptions of the melodic, timbral, harmonic and rhythmic aspects of the musical content. On each of these descriptions (except for pulse information) we can assemble raw information into macro-units labeled with a symbolic alphabet to build the knowledge model that allows us to make variations. This has already been describe in

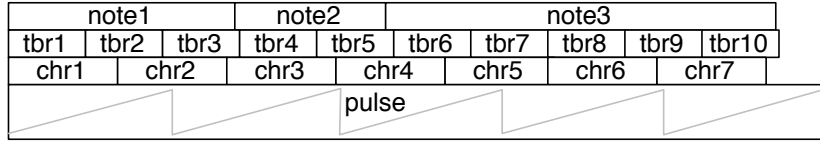
the previous chapters. However, because of alphabet size considerations (see 6.1.2.3) and important segmentation issues, we do not want to combine these descriptions into one multiple viewpoint model (such as in [Conklin 01] for example). The alphabet of such a multiple viewpoint system — for example achieved with the cartesian product of the alphabet of the primary descriptions — would be too large to allow recurrent patterns to emerge in the graph structure depriving us from variation possibilities. The segmentation issues are discussed at the end of this section.

However, we can still use the raw information of these four types of descriptions complementary one to another. The principle is the following: the knowledge model is built from the symbols provided by one of the descriptions, concurrently the extraction for the other description are also computed and raw data from these — that is data before the segmentation — are recorded alongside the *main* description as automatic *annotations*. The main description gives the segmentation, analysis, labeling and modeling of the stream and the annotations complement the information for each unit of this sequence. The knowledge model being based on suffixes (see subsection 6.2.1), we use for the annotations, raw descriptors value at the end of each macro-units of the main description. These annotations are fruitfully used when generating variations as we will see with more details chapter 7. The pulse description which does not lead to the building of a knowledge model is used indiscriminately to annotate the data from the other descriptions. Finally for the three audio extractions, we obtain three possible representation of our input:

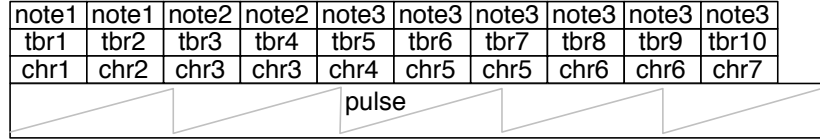
- A melody oriented representation using a knowledge model of notes. For each note, the harmonic context is given by the current chromagrams, the timbral context by the current MFCCs vector and the phase and tempo information is included as rhythmic information.
- A timbre oriented representation using a knowledge model built on MFCCs clusters. For each element of the sequence, considered as spectrally stable, a pitch information is added and an harmonic context is given by the chromagrams analysis. Phase and tempo values informs about the rhythmic context.
- An harmonic oriented representation which knowledge model is built on a sequence of stable chromagrams. The pitch information is added to inform about the *prominent* note in the harmony and the timbral aspects are given by the current MFCC analysis. The phase and tempo values from the pulse extraction allow rhythmic considerations.

It is important to note that these three representations are not equivalent to a unique multiple descriptors representation because of the segmentation. For each of these representations, the segmentation, ie the size of each element in the sequence is given by the *main* description — except for the pulse extraction which is considered as continuous. For example, melodic representation manipulates longer elements (notes) than harmonic representation which macro-units durations are between 50 and 250 milliseconds. The complementary information added to the *main* description follows the segmentation of

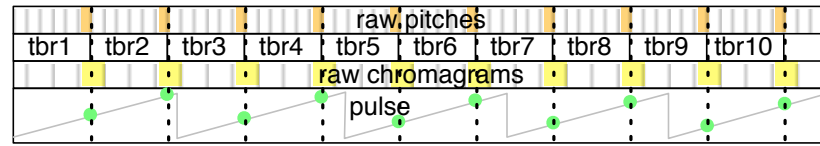
the *main* description. That is, for example in the melody oriented representation, one note will have one fixed information of its harmonic context. We illustrate this remark in Figure 18 with timbre description as main description (macro-units denoted with tbr#). Figure 18a presents all the possible *main* descriptions with their respective segmentation. Figure 18b shows what happens if we applied, for example, the timbral segmentation to all the other descriptions. Because our knowledge model is based on suffixes (see section 6.2), we take the values for the pitch and chromagrams descriptions at the end of each timbral element. Finally Figure 18c illustrates what we do in our system: for each unit of the timbral description we take the current value of the other description’s raw extraction.



(a) The Four Descriptions with Respective Segmentation



(b) One Segmentation for All the Descriptions



(c) Segmentation of the Main Description with Raw Annotations

Figure 18: Crossing Primary Extractions Information to Enrich Memory

This way, we take advantage of all the data extracted from the input stream(s) in all the representations whether leading to the building of knowledge model or to complement, in form of annotations another instance of the model.

6.3.1.2 Information Deriving

A second way to take advantage of already implemented processes, is to use partial data coming from the extractions that have not been used to form the macro-units as annotation. For example as described 5.1.2.1, the pitch algorithm also extracts a mean amplitude information corresponding roughly to the MIDI velocity i.e. to the musical dynamic of the note. This information is not used to aggregate units or produce labels. However it is a very important musical information thus considering it helps smoother variations and enables to adapt our generation to the present context (as we will see in chapter 7). In the case of the timbral description with MFCCs vectors, the first coefficient is systematically

dropped before the clustering because it corresponds to the overall energy of the slice which is not relevant to classify playing modes. We use this first coefficient as annotation as well, informing also on the musical dynamic of the slice.

Another and more subtle example of annotation have been imagined in the case of pitch description with a 12 semi-tones labeling (see 6.1.1.2). In this case the octave information is not used in the labeling since all the pitches are reduced to a one-octave classification. Considering the octave information as annotation indeed adds musical information in the generation process and allows to drastically avoid octave jumps when chaining patterns.

6.3.2 Virtual Pasts

A very strong research direction we have not thoroughly undertaken in this work is the possibility to apply the same processes (extraction, analysis, labeling, modeling, generating) to fixed recorded audio or MIDI material. Our choice on this research direction follows our strong orientation towards the study of an improvisation system with real-time learning capabilities. However, we still investigated the use of external musical material in the very specific case where this material would be complemented with real-time analysis of improvisations.

As our knowledge model is strongly sequential and incremental, it essentially depends on the order of the symbols of the sequence. Thus, complementing an already learnt material with a real-time improvisation fundamentally means prolonging the sequence built on the recorded material by adding symbols extracted from the real-time input at the (right) end of the model. That is, for music material, adding elements *after* the previously analyzed material. In other words, the pre-analyzed musical material is loaded into the model exactly as if it were a *virtual past* of the improvised sequence. And the improvised *complement* is added as a musical *following* of the recorded material. Naturally, this mechanism can be used to utilize material from rehearsal sessions for example.

But more interestingly, we used this technique to create for example hybrid improvisations between great recorded solos of the Jazz heritage and live improvisation. Our typical example has been, along these years, the use of a coda from *I want to talk about you* by John Coltrane (saxophone) complemented with an always renewed live improvisation of the saxophonist of the Nine Spirit Company, Raphaël Imbert, on the same theme. To do so, we analyzed the coda by John Coltrane with the exact same process as if it were a live audio stream and save both the audio, the analysis results and the knowledge model in computer files (respectively in `aiff`, `json` and `dot` file formats). The complementing of pitch is straight forward as the alphabet is fixed and thus the same for Coltrane's coda and Imbert's playing. However, for vector description, due to very different qualities of recording caused by the disparate date, conditions and technical equipment used, the MFCCs vector for example gave at first almost non-overlapping and separate clusters. This resulted in the knowledge model in almost unconnected portions of the graph with no, or very few, possibilities of hybridizations. Then we started to block the definition

of new clusters in the process of labeling (see 5.2.1.3) to force the live improvisation to be analyzed with the existing symbols defined on Coltrane’s coda. This method gave so much successful results that Raphaël Imbert integrated since this new “piece” around *I want to talk about you* in almost every concert he gave with our system.

6.3.3 Higher Structure Clustering

The Factor Oracle graph used as our knowledge model and described Section 6.2.1 is a powerful graph structure. However, because of its linear structure, it lacks direct access to hierarchical information, notably about the overall structure of the sequence which is a very important musical information. This problem of high level structure with usage of the Factor Oracle graph has been address with interesting results in [Schankler 11]. However, Schankler et al. have a global approach which includes the role of the computer-based improvisation in their study. On the other hand we are looking for higher level structure primarily in the input of our system without considering the role of the computer. Thanks to the visual observation of the Factor Oracle graph (see section 8.2) we noticed that Factor Oracle contains nevertheless some consistent high level information through the repartition of links along the sequence and especially the accumulation of suffix links in certain portions of the graph. From this observation we searched for an extraction algorithm. The extraction of such information can be summarized as a problem of graph partitioning or clustering. However one of the main interest of Factor Oracle is its incremental construction. Therefore we wanted to keep this advantage even for higher level information matter. This excludes usual clustering methods using similarity matrices, eigenvalues or expectation-maximization algorithms as those presented in [Azran 06] for example.

In the Factor Oracle construction algorithm we can observe a very strong reinforcement mechanism since links with high *lrs* (long patterns) can appear long after the first occurrence of the pattern and make connected portions of the graph emerge at once. An extraction algorithm needs to take advantage of this mechanism. Furthermore another very notable observation is the difference between “small” or *local* links i.e. links pointing to recently appeared patterns and wide or *global* links which represent the replaying of material presented long before. Both types of links, if they appear numerous permits to define consistent portions of the graph that we can call *regions*.

We proposed an algorithm (fully given section A.3) based on a weighting system which allows a reinforcement mechanism and gives, thanks to a threshold, the boundaries of consistent regions. It uses a list of weights (integers) associated with every state (and initialized to 0). It is linear in time and space. Three control parameters are required:

- a minimal context length, *minCtx*: suffix links with a *lrs* above this threshold are to be considered (usually around 6).
- a value (named *local*) to decide the local/global character of a link. This value is a criterion on the span of the link (number of graph state between both ends). It

is expressed as a multiplicative coefficient on the *lrs* and is superior to 1 (a value inferior to 1 would declare as local only a pattern overlapped with its own repetition, that is an exact loop. Usual values are from 2 to 15).

- a weight w to define a zone of “influence” of a link. This value is also a multiplicative coefficient applied on *lrs* and superior to 1 since the whole pattern must be included in the influence zone (usually between 1.5 and 2).

The output of the incremental clustering algorithm we propose (illustrated [Figure 19](#)) gives consistent portions of the graph based on the span and density of links in the graph. Their boundaries are given by contiguous states with a weight superior or equal to 1. However, to keep only strongly consistent regions we use a threshold (W_{min}). Changing the *local* parameters changes mainly the number of regions. With high values of *local* a rough clustering is done and we find the main large sections of the sequence. Lower values of the *local* parameter reveals a refined analysis with smaller and more numerous portions.

Considering the nature of a suffix link — a connection between identical patterns —, it is rather easy to add to our algorithm an identification mechanism for the portions delimited in the graph and thus be able to link a portion with another when they have the same musical content i.e. common patterns. Nevertheless, some situation are, even on a musicology point of view, difficult to determine. For instance, if we consider a piece with a first theme A and later on a second theme B at first clearly separated from A. In the development of the piece, A and B may be chained one to the other with a subtle musical overlap as we can find in many pieces. How can we define in these conditions a precise border between the second occurrence of A and the second occurrence of B ? Even though we are able to identify A and B both times, we can not draw a precise border between theme in the development section and two different musicology analysis will probably place it differently. Our identification mechanism encounter the same kind of problems. To decide nevertheless and simplify the algorithm we chose to take as reference the more prominent theme, that is the portion in the graph which has the highest weight in the algorithm.

This identification mechanism associating portions similar in content, allows to obtain a new definition of region. A region does not need to be contiguous and can gather several parts spread along the sequence. With this system the clustering goes beyond the time proximity and local consistency to recognize even fragmented regions. We can use the musically meaningful analysis given by the output of this clustering algorithm (both the boundaries and the identification of musical content) and add some hierarchical and form information to our knowledge model. An example of this clustering algorithm is given [Figure 19](#). The different fragmented regions appear as colored rectangles above the Factor Oracle representation. Colors are given by the identification mechanism described in the previous paragraph. Thus all the rectangles of the same color represent one fragmented region.

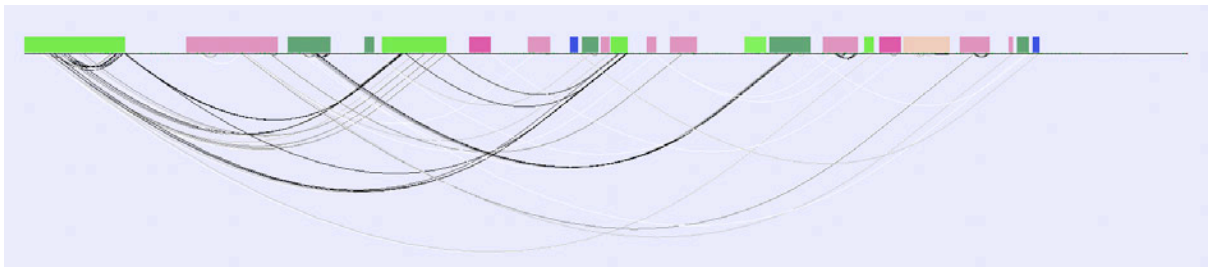


Figure 19: An Example of Higher Level Clustering

Chapter 7

Performance

In the $P\ Q\ f$ cognitive architecture of [Blackwell 12]’s *Live Algorithms*, the Q entity corresponds to the *Performing* part of the system. This part benefit from the *listening* and the *modeling* to generate a new musical discourse. We present in the chapter the part of our system corresponding to this effectuation. The knowledge model described in chapter 6 enable us to generate new paths in the memory to propose variations on the original material learnt. The principle to build such paths is describe in the first section of this chapter (section 7.1). Then, from these potential paths, we generate one or several musical output that we need to *conduct* — automatically or manually — to make a proper musical realization. We present the means we invented in our work for this *conducting* in the second section (7.2) of this chapter.

7.1 Production

In the previous chapters, we described how, from the listening of a musician’s playing, we build and enrich a knowledge model destined to analyze mainly the pattern repetitions in various aspects of the musical discourse. From this model we generate new variations mimicking the style of the material learnt i.e. *style reinjections*. We will see in the first section 7.1.1 of this chapter how our core principle for generation is tightly linked with the structure of our knowledge model and how we exploit this structure presented section 6.2.1 to generate new variations from the original material. Then we will present how we render an actual musical discourse to participate in the musical interaction (section 7.1.2). Doing so, we will finally present two features of this rendering, namely time-stretching and transposition and explain how these features need to be coupled with the generation mechanism.

7.1.1 Generation Principles

To generate new musical discourse thanks to our knowledge model, Factor Oracle, we rely on the alternation between two behaviors: continuity i.e. forward reading of portions of the model detailed Section 7.1.1.1 and jumps in the graph as will be explained Section 7.1.1.2.

These jumps rupture the continuity and chain patterns which were not consecutive in the original material and thus bring novelty. These behavior correspond to the combination of the principles (explained [chapter 4](#)) of *style reinjection* which relies on the reuse of learnt material to generate new discourse and *statistical modeling* which works mainly on probability drawing to generate symbolic progression as with Markov Chains for example. We name *improvisation logic* this fragmented walking of the graph of our knowledge model

7.1.1.1 Continuity

Thanks to the linear skeleton of Factor Oracle, the first and easiest principle to start generating new sequences from the musical material learnt in the system is the forward reading of portions of the original material. To do so, we navigate through consecutive state of the graph generating factors (sub-sequences) of the original sequence. They are incontrovertible patterns respecting the style of the original sequence but naturally call for another mechanism to add novelty.

Though, even this simple mechanism raises practical problems because of the very different types of symbols on which we can build our knowledge model (see [section 6.1](#)). The musical temporality underlying the different types of symbols questions the management of the size of the linear portions of the skeleton to read. Indeed, notes being (in average) much longer musical elements than MFCCs or Chromagrams, recalling the same number of successive states in the graph indiscriminately for all the symbolic descriptions will create very different durations when rendered in sound realm (see [7.1.2](#)). Keeping the link with the symbolic description and expressing the duration of these contiguous portions of the model in number of consecutive states has been a temporary solution. Depending on the description we had then a much higher *continuity* parameter for spectral description (typical value of 30 states, corresponding to 30 consecutive MFCC slices) than for notes (typical value of 7 consecutive states/notes). But as we did for *inputs* and *streams* (see [5.1](#)), we came back to the time anchorage of music when trying to unify our approach into a global system functioning indistinctly on several descriptions.

The *continuity* parameter, that is, the average size of the portions of the original material left unchanged when generating has acted from the very first versions of the system as a *variation ratio*. The shorter these portions are, the more the system uses the jumping mechanism which is the mechanism adding novelty to the generated discourse (see [7.1.1.2](#)) and the more the computer based generation is moved away, modified, recombined, compared to the original material. This parameter, expressed in later prototypes of the system directly in time (seconds or milliseconds), remains one of the main parameters left to the *supervisor* of the system to musically shape the musical result of the system. It can go from a few dozens of milliseconds to several seconds. The longer the consecutive portions are, the more the system will sound like a simple echo or delay machine. The shorter the portions are, the more the system will seem to dismantle the structure of the original material.

7.1.1.2 Jump Trees

The second and most important mechanism which enables the system to add novelty in the generation is what we call *jumps*. As we have seen in the presentation of our knowledge model (section 6.2), the Factor Oracle graph includes *suffix links* which connect repeated patterns of the original sequence. We illustrate this map on an abstract example Figure 20. The structure implicitly built by the collection of all these suffix links (when discarding the rest of the graph) is a tree connecting all the patterns from any length and presenting them (when traveling along the height of the tree) with respect to their longest common suffix. In other words, all the patterns of the sequence are connected in this tree thanks to their last symbols. Patterns with a long suffix in common are placed closely towards the leaves of the tree.

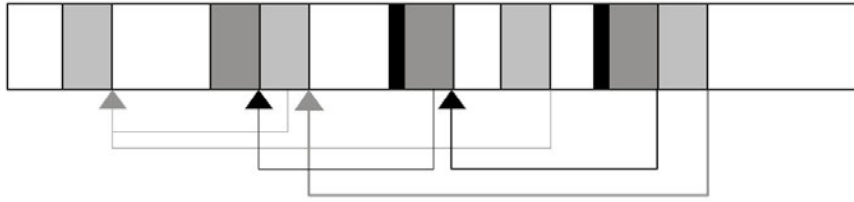


Figure 20: Map of Patterns Constituted by the Suffix Links of an Abstract Sequence

Naturally, taking the whole tree i.e. all the patterns implies that, at least at the root of the tree, patterns may have nothing in common ($lrs = 0$, see 6.2.3). But pruning this tree depending on the lrs which labels the edges of the tree i.e. the number of symbols in common between the ending of two different patterns, with a threshold named *minCtxt*, creates a forest of disconnected trees in which all the patterns have a common suffix of length *minCtxt* at least. In context model environments for generation such as Factor Oracle, this common suffix is also named common *context* which has not to be confused with the *musical context* that we mentioned earlier in this thesis. Figure 21 illustrates this forest of trees built from the abstract sequence presented in Figure 20.

These trees of suffix links named *Suffix Links Trees* or *SLT* gives us the possibilities for new musical chaining of patterns. If, instead of continuing the sequential reading of a specific pattern, we examine the tree containing this pattern; once pruned, we are sure that all the pattern present in the tree have a common suffix of length *minCtxt* with the pattern we have just finished reading, that is, they all finish by the same succession of *minCtxt* symbols. If now we jump in the original sequence from where we arrived with the consecutive reading, following any path in the tree, we arrive in a new position in the sequence which finishes by the exact same linear succession (skeleton of Factor Oracle) of *minCtxt* symbols that the one we have already read and we can continue by reading the next symbol (along the skeleton) at the destination of the jump. Doing so, we chained a pattern of the original sequence (the origin of the jump) with a new continuation (the next symbol of the skeleton at the destination of the jump) thanks to a path in the tree.

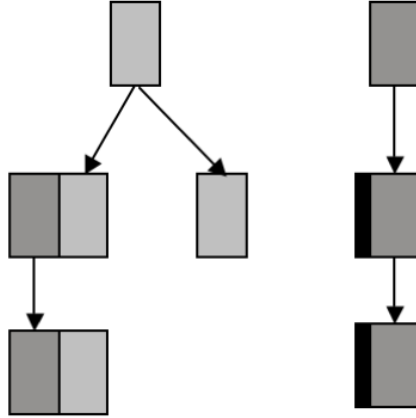


Figure 21: Trees of Patterns Constituted by the Suffix Links of an Abstract Sequence

This mechanism has been firstly implemented in [Poirson 02] with consideration of *forward transitions* (see 6.2.4) — links that we do not use anymore in our system — and the method of alternation between continuity and jump has been presented from a probabilistic point of view in [Assayag 04]. However, as noted in this last article, there are no available probability model of Factor Oracle graph. This mechanism can thus not be implemented effectively as a probabilistic drawing such as those used with Markov Chain. This point has been a serious problem when trying to parametrize statistically this alternation but it has been a very successful asset when examining the coherence of the generated discourse on the musical level because of the musical strength of *continuity*. This generation principles and the reasons for not using *forward transitions* are thoroughly explained in [Assayag 07] —in this article, *forward transitions* are named *factor links* and the *jump* mechanism is accurately named *context-switching*. This last article also illustrates and formalizes thoroughly the use of suffix trees and present the practical method (relying on what they call *reverse suffix links*) to compute them efficiently. Except the programming environment and language, our implementation of this search for solutions in the trees barely differs from the one described in [Assayag 07]. The gathering, selection and use of the rhythmic coefficient though have been importantly revised and generalized as we will present in section 7.2 and in the third part of this thesis.

7.1.2 Rendering

Once we have generated new symbolic path in the model built on the original material, we need to come back to the sound realm and effectively render this variation. The rendering of new variations in our system relies on the principle of *concatenative synthesis* that is the chaining of fragments of music material — the duration of these fragments must be longer than 20-30ms to be considered as separated elements and are usually between 50ms and 1 to 3 seconds. In the case of an audio medium (see 5.1.1.2), the use of unchanged

portions of the original sequence in the model enables our system to playback directly chunks of the audio recorded from the input. In the case of purely MIDI input, our system naturally outputs one (or several) MIDI stream(s) which realization on the sound level has to be done by an external synthesizer. The rendering entity is called *player* in our general anatomy of the system. The recombination of unaltered pieces of the original material creates at the same time a very close proximity of the computer base generation to the learnt material and a definite capacity of engendering musical proposals and novelty.

7.1.2.1 Output

Whether MIDI or audio, the output of the system is constituted of small portions of the original material concatenated one to another to form a continuous musical discourse. As we have previously seen (7.1.1), our knowledge model and the usage of it ensure that the beginning and ending of the portions of material we recombine have the same content (at least for a few symbols) so they can smoothly be chained one to the other.

When dealing with audio streams, a simple crossfade (typically during 30ms) is then sufficient to concatenate the two pieces together and most of the time the editing point is really inaudible without any specific adjusting. In the case of recombinations of MIDI pieces, the editing is more complicated especially in the case of polyphonic MIDI. Indeed because of the reduction of the MIDI slices to unique symbols, two slices different in content may be labeled by the same symbol and considered as “equivalent” in our model. In this case, when editing the MIDI stream to concatenate such slices (and their continuations) and because we concatenate MIDI (i.e. symbolic) information, we have to take care of existing and continued or disappearing notes. The tiny details of this process are non trivial and a list of all possible cases had to be considered to render properly a polyphonic MIDI stream.

7.1.2.2 Features

Very early in its design, our system benefited from the research around analysis and processing on-going at IRCAM which lead to efficient tools for accurate and good quality transposition and time-stretching [Bogaards 04]. These tools have been integrated in our working environment and even though it could appear as a simple feature or effect on our improvisation system, their integration can not be done afterwards as would be any audio effect added to the system. We will see why and how we have to link these two feature to the generation mechanism of our system.

Time-Stretching Time-stretching consists essentially in our case of the slowing down or speeding up of the improvisation output of the system. In the case of audio streams, we rely for this task on *superVP*, a powerful phase vocoder capable of on-line analysis and re-synthesis of sound. This sound analysis and re-synthesis of the musical output is done downstream from the generation processes already described but the speed of the

discourse output by the system naturally conditions the speed of its upstream generation. Indeed, if the the system has to output an audio stream twice as fast as the original material, the generation algorithms (navigation in the knowledge) has to take place twice as fast as well to feed editing point information on time. On the contrary, if the audio speed is slowed down, then we do not need to generate the informations of unchanged portions and jumps (see 7.1.1) as urgently. Because of this speed synchronization, the time-stretching has to be linked to the generation algorithm, or at least, the generation algorithm needs the speed information to send the correct information in time. We will see in the second part of this thesis the possible architecture for such a coupling.

We could argue that the generation of musical phrases could be done *out of time* ie *a priori* then sent to be rendered. And this has been the case in a number of prototypes of the systems but the will of being more *reactive* to the environment of the generation necessitate to adopt an on-line architecture for the generation as we will thoroughly develop in the third part of this thesis.

Transposition *SuperVP* enables good quality transposition up (or down) to one octave or more. In the case of constant audio transposition, this process can be done totally downstream from the generation and without any consequence on the generation mechanism. However, when using the knowledge model built on the note extraction of the input, we can add smarter transposition “effects” with the usage of this knowledge. For example, we can do *compositional* transpositions as symmetry around a central note or pedal (constant) note. In this case, the knowledge contained in our pitch model informs the transposition system to get a adequate result. This has been implemented thanks to the work of G. Bloch.

7.2 Conducting

We explained in the first chapter of this part, [section 4.2, OMax Principles](#) that our system is meant to be at least guided by a human operator that we name *supervisor*. This commitment does not exclude composition (or other) situations where the *supervisor* may be replaced by computer scripts or other automatic artefacts executing actions or decisions on the system. In both case, we need high level controls over the system to musically *conduct* or drive the system.

7.2.1 Voices

At first, the system was developed as monophonic, that is, capable of generating only one improvisation at a time. Actually, in [\[Assayag 06b\]](#) the possibility to replay *planned* MIDI sequences has been included to obtain a emulation of multiple generations. But it is appealing when working on computer systems with several listening possibilities to also reflect this multiplicity in the generative part of the system and multiply the voices to

propose polyphony. In our case, several multiplications are possible on several levels and the examination of those raises important questions and has strong consequences on the architecture of the system as we will develop in the third part of this thesis.

7.2.1.1 “Clones”

One possibility to give polyphonic capacity to our system is to multiply the number of instances of the algorithm navigating in the knowledge model to generate new discourses (7.1.1). Doing so, and if this algorithm is purely deterministic as it has been presented and was initially the case, we create several *agents* capable of jumping and recombining the original material but which would actually generate the same improvisation as soon as they start from the same initial position in the original material. This does not add real polyphony as it just generates delayed version of the same improvisation. Therefore we introduced in the jumping mechanism a random pick of the actual jump among the best solutions. This pick ensures that two generations on the same knowledge model i.e. paths in the same graph will diverge at some point and produce two different musical discourses. This way, we enable a polyphony of “clones” by replicating the generation and rendering part of the systems and we generate new variations from the same knowledge model.

7.2.1.2 Descriptions

But as described in [section 5.1](#) several types of information and thus several symbolic models can be built on one input. These parallel models gives us different variation possibilities and we can benefit from these different *analysis*. Several scenarios can be imagined from one “clone” capable of using several descriptions at once or successively to several “clones” each one exploiting one of the descriptions. This latter scenario was the first scenario initially implemented in our system because, mainly, of the coherence it brings to the musical discourse. Indeed, the strong symbolic model justifies by itself the coherence of the recombinations: paths generated on the model built on *notes* will be intrinsically consistent melodic variations on the original material — thanks to the suffix properties explained [6.2.3](#). We explain an architecture of our system to apply this principle in [section 9.2](#).

However, the usage of several descriptions to supply the variations of one “clone” only has been a recurring musical request. Two typical and musical situations have been mainly requiring this possibility. In group improvisations and especially in free improvisations, the coherence of the musical superposition of each individual discourses can be achieve on several level. For example, one of the musician can develop its melodic line while two other are realizing a textural background. Each musician is freely deciding and switching his role and more specifically the aspect of the overall content he or she is focusing on. A computer based “clone”, when taking part in such situation would greatly benefit from the same possibility that is, being able to exploit one or the other description when

needed and furthermore switch from one description to the other without any rupture in its musical discourse.

A second situation imagined and discussed with Steve Coleman (saxophone) when he had the occasion of testing the system is the concurrent usage of all the descriptions for one “clone”. As we were trying to get a unity in the practical usage of all the descriptions, the idea emerged that the *decision* to use one or the other description for one “clone” could forge its musical characteristics. Exactly as some musicians are improvising with very strong melodic systems — which is mainly the case of Steve Coleman — while some others are using more the textural aspects of the sound, the system could have *musical personalities* defined, for example by a ratios of melodic, spectral, harmonic recombinations. This means that one “clone” ie one generated discourse has to be generated concurrently by several instances of the knowledge model. This mechanism has again, strong architecture implications. We present an architecture capable of such behavior in [chapter 10](#) of this thesis.

7.2.2 Content Shaping

The navigation and use of the whole model built (on the fly) on a particular input may produce sometimes *too surprising* or inadequate musical discourse especially when the input material is very heterogeneous or when the external context is more constrained than in free improvisation. Therefore, in addition to choosing the *type* of variations through the choice of the description underlying the knowledge model as explained in the previous section, we need mechanisms to control more finely the *content* recombined through the generation algorithm. For that we imagined two means: we can easily restrain the navigation to definite *regions* of the knowledge model, corresponding to certain sections of the learnt material which may exhibit desirable properties. We can also use the annotations we have presented Section 6.3.1 to favor patterns which exhibit searched properties.

7.2.2.1 Regions

Constraining the navigation in a definite and contiguous section of the knowledge model to control the content of the generation have been introduced very early in the system. [Durand 03] already used a function which enables to specify boundaries (in percentage) on the model to define where to start the generation. Along our work, we developed and augmented the usage of this kind of *regions*. At first, we only implemented two regions and enable the manual usage of those. Then, we actually notice that the smooth alternation between two different but linked regions — linked for example in musical content which will be revealed in the graph by strong links between both regions — produces a very musical discourse combining two ideas in a consistent way. Thus we included a mechanism to allow the navigation in several regions at once and/or an automatic alternation between regions. As we still use the suffix links of the model to jump from one region to the other we always ensure the smooth transition from one to the other region.

We presented in section 6.3.3 our work on the automatic clustering of the information contained in the knowledge model. The output of this work enable to recognize fragmented portions of the model which are consistent in content. Thanks to this work, we can generalize even more the notion of regions and use these identified fragmented sections of the model resulting from the analysis of the links to the control the content of generation done by the system. Constraining the navigation to these split regions enable to control even more finely the content recombined in the computer based generation with, for example, the usage of one specific thematic element which may appear several times along the material learnt and thus being well recognized by the automatic clustering. The resulting computer generated variation will exhibit and reinterpret recognizably this thematic element and benefit from all the expositions and references to this element in the original material.

7.2.2.2 Usage of Annotations

Our knowledge model ensures the smooth concatenation of portions of the original material for all the descriptions we use. However, as presented in section 6.3.1 we can extract more information than needed to build the model. These extra information can be used to musically drive the content of the computer based generation, on top of the navigation in the model. When examining and picking jumps as explained 7.1.1.2, we can benefit from additional criteria to favor the playback of certain patterns depending of their musical properties as the dynamic for example (which is not taken into account in the Factor Oracle graph). The importance of these additional information can be either set automatically (with scripts or depending on the context i.e. the current analysis of the input) or set by the supervisor to bend the musical discourse generated by the computer towards arbitrary musical directions. For example, putting forward the usage dynamic/energy information in the generation process enables the supervisor to control the dynamic of the computer's discourse and play a *forte* variation — not with a simple volume amplification on the output of the computer, but directly by using *forte* parts of the material learnt.

The pulse extraction (see 5.1.2.4) which is both continuous and periodic can be integrated systematically in the process of jump selection when the external musical context is relevant, that is, naturally in pulse contexts. In this case thanks to a difference in phase with a acceptance threshold and a time-stretching if required, the jumps — found in the knowledge model by the same process as explained earlier — can be selected on their pulse characteristic to preserve the original pulse of the material or to adapt to the current pulse context. Doing so we respect in the generation process characteristics of the original material which are not modeled in the graph. This opens up a very powerful enhancement in our system: the ground principle of our model rely on the recurrence of patterns on the melodic, timbral or harmonic aspect of the music. This does not assume any regular structure in the musical discourse and does not constrain our system to specific style. But not being able to follow such pulse or harmonic structure *de facto* exclude our system from improvisations contexts which may refer briefly or permanently to these

structures. Using the pulse extraction as an external and complementary information let us choose depending on the situation and along the playing with the system if a regular pulsation is indeed an important aspect of the current improvisation that we need to respect or not. In other words, not basing the knowledge of our system on the assumption of a pulsed musical content and adding this constraint afterwards enhances drastically the versatility of the system.

This mechanism developed for the pulse annotation is the first step towards an automatic adaptation to the context. We plan to extend this mechanism to the harmonic annotations and context. Dynamic or energy annotations can already be used the same way i.e. for a systematical following of the current musical context. However, any systematic behavior of the system on the musical leading of the improvisation may shadow its novelty capacities by this repetitiousness and is likely to bore the musician(s). Therefore we decide to leave the usage of the none periodic annotations to the will of the *supervisor* or to any compositional scenario.

Part III

Software Architecture

Software Environment

The early prototypes of the OMax improvisation software have been based on a illustrative separation of the real-time processing and the higher level modeling across two environment: Max/MSP [Cycling74] and OpenMusic [OpenMusic]. Both are visual programming environment but the first one, Max/MSP is dedicated to MIDI and audio real-time processing while the other, OpenMusic is oriented towards music composition.

In practice, OpenMusic is programmed using the object oriented facilities of the Lisp language. The high level of abstraction of the Lisp language historically allowed to manipulate, study or modify easily the internal knowledge model of the system. The rudimentary MIDI functions of OpenMusic were at the time sufficient to prototype the very early versions of the system, especially those without real-time interaction. From 2005 the system combined OpenMusic's capacities for abstract structures with Max/MSP's facilities for real-time handling [Assayag 05]. The name *OMax* (O-Max, for OpenMusic and Max) was invented at this occasion.

Max/MSP, was originally developed at IRCAM [Puckette 91] and is specifically designed to handle a mixture of *control* and *signal* processing in data-flow oriented *patching* interface. Nowadays simply named Max, it is in its 6.1.x version and integrally developed and edited by Cycling'74, a californian company. Its visual interface takes the form of a visual canvas, the *patcher*, in which *objects* are connected one to another to assemble complex *patches* i.e. programs. Cycling'74 provides hundreds of objects to handle, process, store or present *control* and *audio* messages or data. To enrich this included collection, functions programmed in C or C++ languages can be encapsulated in the software Application Programming Interface (API, written in C) supplied by Cycling'74 to build this way *external* objects or *externals* implementing any desired function.

To apply the principles that we presented in the previous chapter, we have invented new software architectures for our system. We present in this part of the thesis the three main steps of this architecture. In the first chapter (section 8.1), we present the *core* elements of the software: two low level structures. One structure is capable of holding the data extracted from the input(s) and the other structure implements the Factor Oracle graph. Along this part, we will describe how these two structures were initially dependent and how we progressively made them independent to use them extensively. In the second chapter of this part (chapter 9), we present the global architecture of the first *stable* prototype for our improvisation system. Then, exploring the drawbacks of this first prototype, we present in the third chapter of this part (chapter 10) a newer and totally redesigned architecture for our software which adopts a database approach of our knowledge model(s). From the very beginning of our work, we were attached to adopt an efficient programming oriented towards the *real-life* testing of the system. That means that we took very much care to produce effective prototypes to test our research with musicians, possibly on stage in the critical concert situation where the usability, the musical pertinence and a certain stability are required.

Chapter 8

Fluid Data Access

In the early prototypes of the system, the illustrative separation of the high level knowledge model in OpenMusic and the real-time processing to analyze and render a MIDI or audio stream in Max/MSP, as presented in the previous section, implied the definition of a specific protocol for the interaction between both environment, using the OSC protocol as support. To enable thorough explorations and more efficient usage of the knowledge and strongly link the memory of the system with the musical reality of both the analysis part and the generation part of the system, we needed a more fluid access to the model. We present in the first section of this chapter the core implementation of data and graph structures in the software environment Max/MSP that we have done to obtain this fluidity between extracted and analyzed information from the streams, and the storage and modeling of these ([section 8.1](#)). Going then from this low-level implementation to a higher level interaction with this core, we explain in [section 8.1.2](#) the interfaces needed to interact with these structures in our programming software environment. Then, in the third section of this chapter ([section 8.2](#)), we present one of the first high level illustration of the efficiency of these new structures: we propose the online visualization of the knowledge model directly in our software environment Max/MSP ([section 8.2](#)).

8.1 Data & Graph Structures

We discussed in [chapter 3](#) the need for an improvisation system to be adaptable to its context. In the previous part, we presented the principle on which we base our system. We claim in this part that the software architecture is not neutral towards this adaptability and the software design drastically changes its usage and possibility. In this section, we present the core structures we propose for our system ([section 8.1.1](#)). We pretend that the fluid access to these core elements enhance the reactivity of the whole systems and the interfaces to access these structures induces some orientations of the overall architecture. This section examine the low-level aspects of this new core for our system.

8.1.1 Low-Level Structures

On the contrary to the previous versions of the knowledge model (in Lisp) described in [Assayag 06b] which combined in one structure both the content — ie the data extracted and analyzed from the audio — and the graph attributes (transitions, links etc.), we decided to split these information in two different structures. On one side, we store the data extracted from the audio, on another side, we build the Factor Oracle graph corresponding to these data. There are two main advantages of separating these structures. Firstly, it follows our conceptual description presented in the previous part, in which we have a process of extraction of information and segmentation of the input streams and then, possibly a knowledge model build on certain parts of this information but only if relevant. On the extracted data we need thus to store more information describing the input than what is strictly necessary for the knowledge model. Secondly, it allows freer research around a new knowledge model. If separated from the data, the Factor Oracle graph can be replaced very easily and new models can be tested very soon without rewriting the whole learning part. In the next paragraphs, we describe the very low level implementations of these two structures in our programming environment, Max/MSP.

Data Structure The first structure we implemented is a hierarchical index database relying on the `vector` implementation of the C++ Standard Template Library (STL)¹. The elements referred in this structure are named *labels* and can contain all the information of the description of the audio and musical content: date on the medium timeline (see 5.1.1.2), duration, section and phrase information for higher level segmentation, tempo and pulse information etc. These *labels* are specialized (heritage mechanism in C++) for one or the other description of the streams: MIDI (slices), melodic (pitches), timbral (MFCCs), harmonic (chromagrams).

Graph Structure The second structure implements an *agnostic* Factor Oracle graph. We mean by *agnostic* a purely symbolic graph structure containing the less musical content possible for each state which are referred only by there *state number* in the graph. Our Factor Oracle structure is also based on the `vector` implementation of the C++ STL; The elements of the structure, the *states*, include all and only the information needed for the graph structure: list of transitions, suffix link, list of reversed suffix links (see [Assayag 07] for an explanation of these latter links).

As we build our model and descriptions incrementally from the analysis stage, those two structures are protected by multiple readers / single-writer locks in the multithreaded Max/MSP environment.

These two structures were at first linked (internally) in two ways: they shared a common name to refer them in Max/MSP and the states of the graph and the labels of the

¹Users of Max/MSP a.k.a. *Maxers* will notice that the `coll` object proposes also an index database. However its slowness and lack of hierachical storing makes it unusable for our intensive needs.

data structure coincide in indexes i.e. each state in the graph corresponded to a label in the data structure and the other way around. In other words they were developed to work strictly on the same segmentation of the audio or MIDI stream. That especially means that one description (for example the melodic description through pitches) is stored in one instance of the *OMax.data* structure and the corresponding Factor Oracle is held in a related instance of the *OMax.oracle* structure. Two descriptions data (for example melodic and spectral) can not be stored in the same *OMax.data* structure. This coincidence of structures and indexes of both structures was justified by the need for the musical content of a state to compute its letter in the alphabet and compare it along the Factor Oracle building algorithm. We will see later, in [chapter 10](#), that we modified this aspect along our research, in a different approach of the system.

8.1.2 Programing Interfaces

On the *patching* level of Max/MSP, we developed an ensemble of seven other objects around these two *externals* for the data and graph structures in order to enable writing, reading and querying them. [Figure 22](#) illustrates this collection of *externals*.

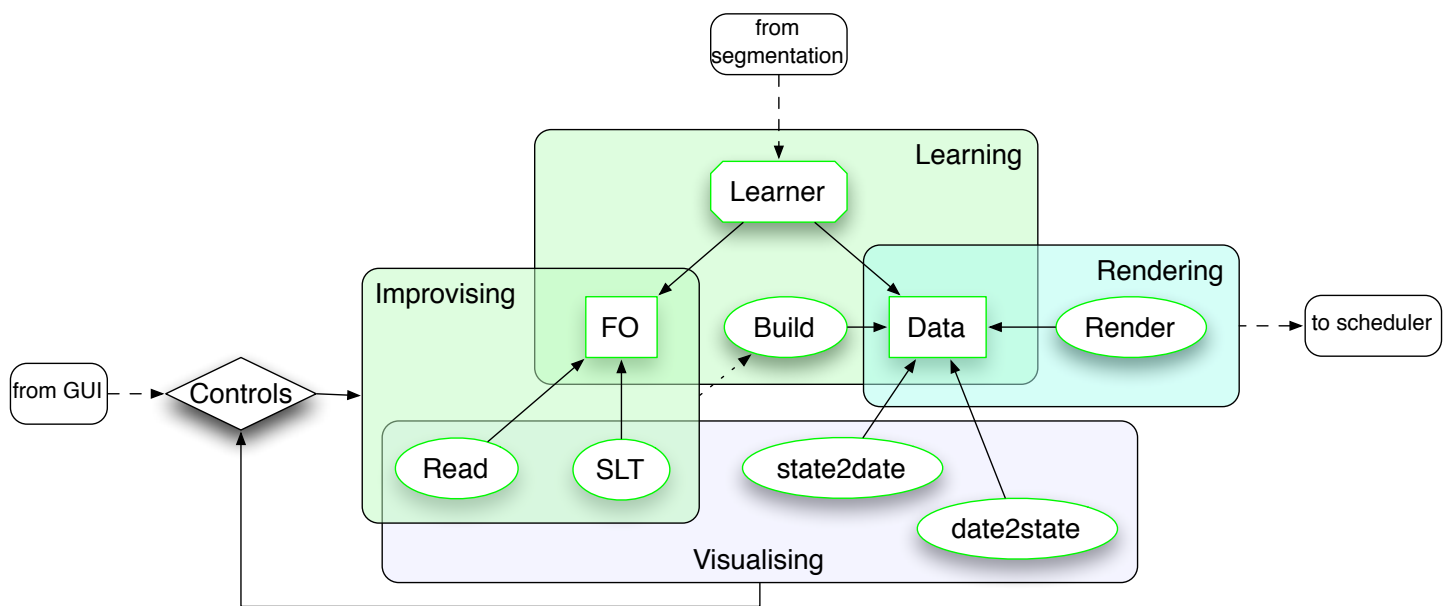


Figure 22: The Initial Collection of OMax's Externals for Max/MSP

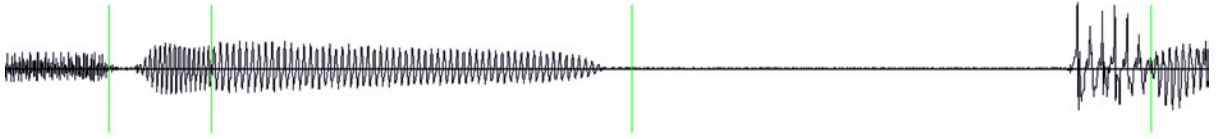
In this version, the *OMax.learn* object receives the data of the segmented input stream and is in charge of running the Factor Oracle algorithm to build the graph and writing both structures. Both structures can be read through specific externals which can be instantiated several times anywhere in the system: the *OMax.read* object outputs for each state of the graph all its transitions, its suffix link and *lrs* and reversed suffix links and *lrs*, the *OMax.render* object accesses the data structure and outputs the whole content of each label. It is important to note that in the version of the externals described here, these two objects refer the structures content through the indexes in the databases.

Because we need to link these indexes with a real musical *medium*, we implemented the *OMax.date2state* and *OMax.state2date*. These two objects (thanks to hashtables of the STL C++ library) are able to efficiently translate dates of the musical *medium* into the corresponding state/label index of the structures and the other way around.

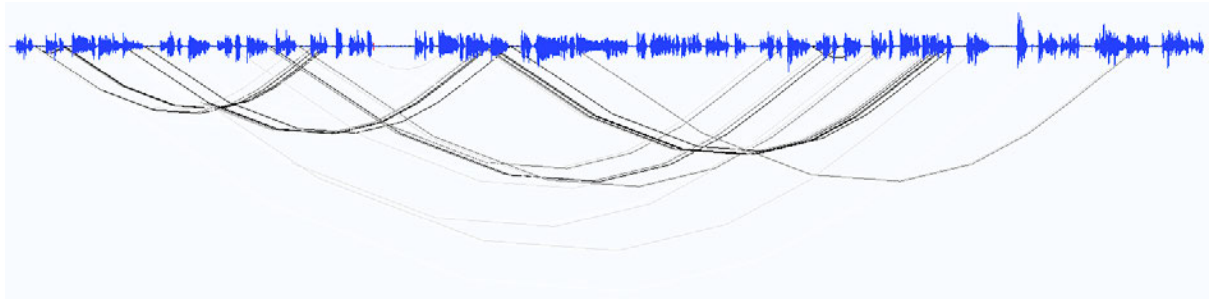
Our primary resource for the navigation of Factor Oracle is the Suffix Links Trees (SLTs, explained section 7.1.1.2), we added thus a specific object capable of navigating efficiently the graph to extract the pruned SLT of any state. Finally, from the initial data and thanks to the knowledge model, we build a new musical sequence. This is the role of the *OMax.build* object which can refer the labels contained in the data structure and form a new sequence of those that can be access to render the new musical discourse.

8.2 High Level Application: Visualization

The first high level illustration of this fluidity of access to the data and graph thanks to the gathering of the core elements in a single and efficient software environment is the possibility of displaying and visually exploring a superposition of the knowledge model and of the musical data. In this direction, we started by visualizing the audio content with a simple waveform and added the segmentation information of our units. Figure 23a



(a) Visualization of a Detail of the Segmentation

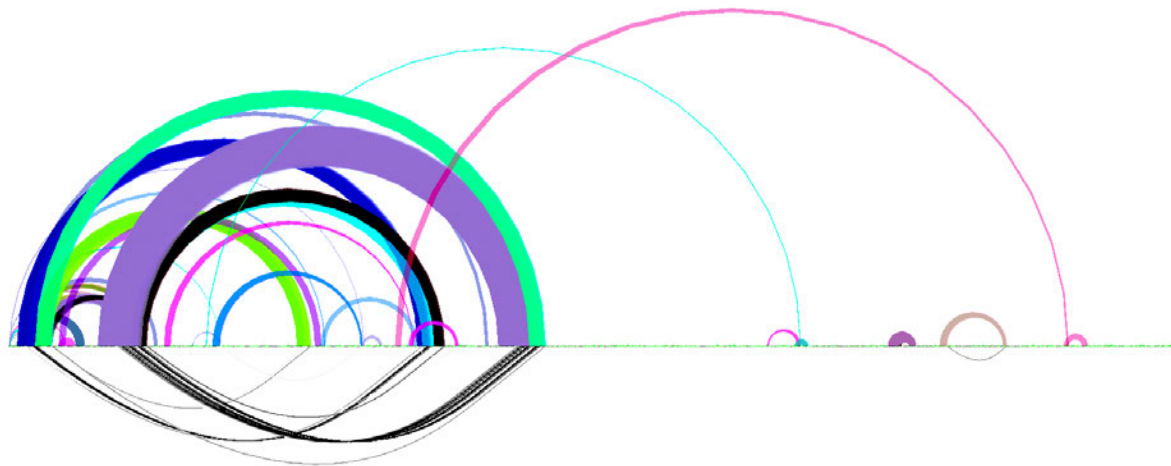


(b) Visualization of the Knowledge Model Overlaid with the Sound Waveform

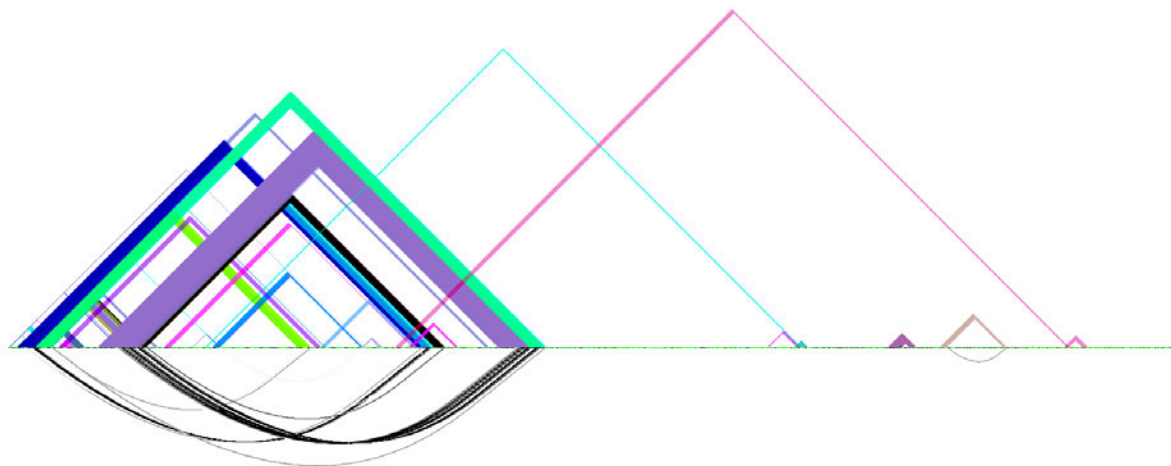
Figure 23: Visualization of the Sound with Higher Level Information

presents a detail of such display. The next and new step was to visualize the state of the whole knowledge model on a particular audio. As the generation part of the system relies on the suffix links of the Factor Oracle graph, we choose to visualize only those as a informative summary of the model. An example of such representation is given Figure 23b where the waveform of the corresponding sound is overlaid on the suffix links of the graph. Darker links denotes higher *lrs* i.e. longer patterns than lighter links.

The visualization of our knowledge model and generation process lead to a thorougher research of several months that we undertook at the very beginning of our work. It is presented in detail in [?]. The main objective of this research was to visualize the internal state of our system for both the acoustic musician playing and the supervisor of the system. However, pattern visualization is not trivial especially in the case of music in which time plays a very important role. We studied several possibilities coming from text or genetic visualization researches to represent the recurrent patterns recognized by the system. Eventually, we decide to display the suffix links of the model with arcs to make them easy to embrace at a glance and efficient to emphasize the structure of the graph. Example of such visualizations are shown [Figure 24](#).



(a) Visualization of the Suffix Links with Colored Arcs

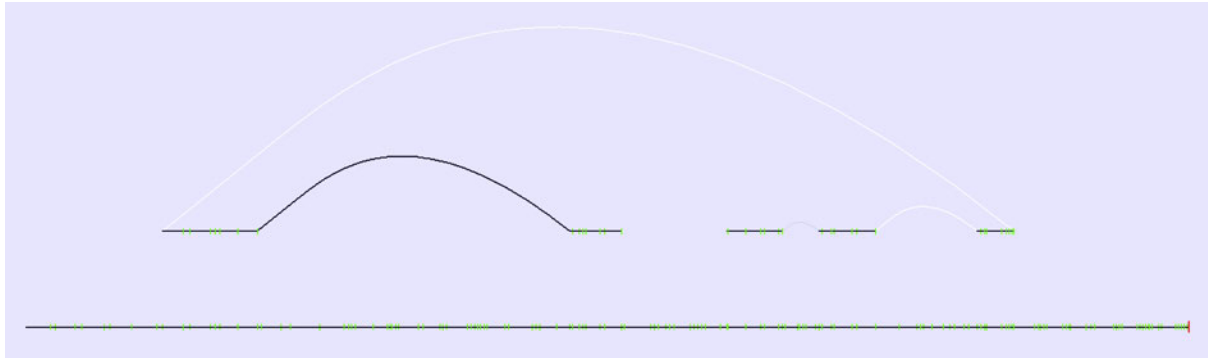


(b) Visualization of the Suffix Links with Colored Triangles

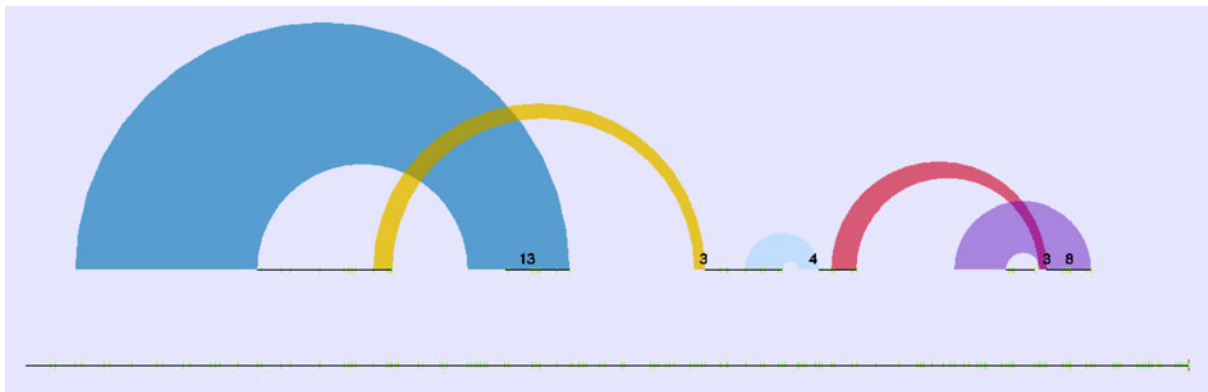
Figure 24: Visualization of the Suffix Links with Colorful Representations

The generation part of our system has also been the subject of studies to find efficient visualizations. The main issue in this part of the research was an essential problem of linearity of the visualization along the time: after several tests, we had to conclude that a

non-linear representation of a musical discourse is not instantly readable for a musician playing with or along the system. This conclusion constraints strongly the possible visualizations for the generative part of the system. For the current usage of the system, we kept in the end a classical linear *timeline* visualization of the knowledge as presented above and decided to superpose it with a visualization of the generation part. But as our system recombines fragments scattered along the original material it is impossible to preserve the linearity for the visualization of both the input material and the computer generated discourse. Example of the visualization of a very small computer generated sequence with the corresponding original material is given Figure 25: the horizontal line at



(a) Visualization of a Generated Sequence with Suffix Links



(b) Visualization of a Generated Sequence with Colored Arcs

Figure 25: Visualization of a Small Generated Sequence with its Original Material

the bottom of these captures represents the original material, the small portions scattered above and connected with simple links or arcs (representing the same links) show the path in the graph followed to generate a new sequence. It is not trivial even on a simple example as the one presented Figure 25 to retrieve the order of the jumps justifying the computer generated sequence.

The visual explanation of each jumps in the graph has no reason to be a priority either for the musician feeding the system or for the supervisor of the system who can examine other clues in the interface. Therefore, we adopted a simpler visual for the generation part of the system: a simple arrow pointing on the portion of the original

material currently replayed and a shadow arrow indicating where it is going to jump for the next recombination is sufficient to realize what is currently going on, on the musical surface at least. This simpler visualization is illustrated [Figure 26](#) with four different *read head* of different colors corresponding to four “clones” (see [section 7.2.1.1](#)). The violet “clone” (right most arrow) shows an example of duplicated shadow arrow (on the left side of the blue arrow) which indicates the destination of the next jump in the graph.

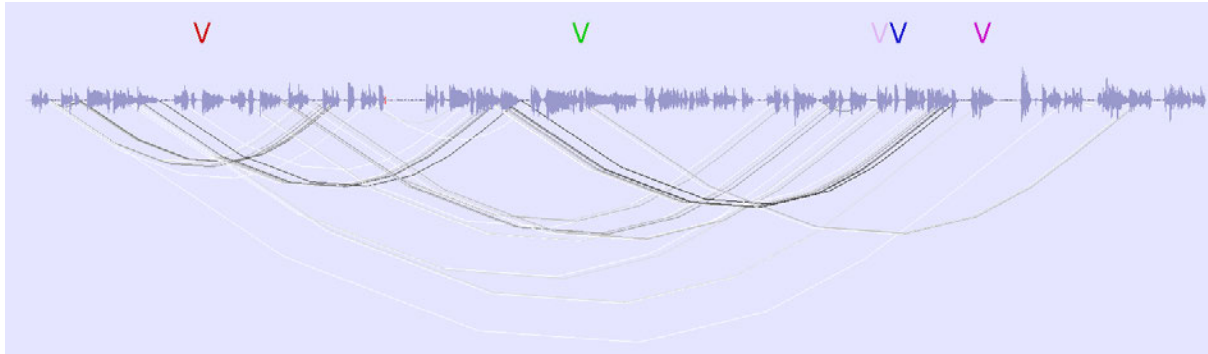


Figure 26: Visualization of the Improvisations with Read Head and Next Jump

We show in this chapter how we implemented low-level structures to learn and store both the knowledge model and the musical descriptions of the input into the sole Max/MSP environment. Then we present the programming interfaces to interact with these structures. Through the example of the novel visualization imagined to display the internal state of the system, we intent to show how the renovation of the core of the system opened widely the possibilities of research and experimentations around the higher level software architecture for an efficient interaction with musicians. Following this first application example, we will present in the second chapter of this part the work achieved mainly at the *patching* level offered by Max/MSP and which is a very efficient environment for prototyping new software architectures. Naturally and as we will see later in the third chapter of this part ([chapter 10](#)), the high level software architecture research implies sometimes new evolutions of the low level core that we have presented in this section.

Chapter 9

Multiple Processing

We present in this chapter the high level architecture we designed initially for the system. This architecture takes into account the constraints of our programming environment and issues on efficiency, adaptability and performances. Important work on this architecture has been done to benefit from the parallel processing of several audio and/or MIDI streams and the extraction, analysis and modeling of several descriptions of these inputs as described in [chapter 5](#). This work concerns both the perception and the generation part of the system. We present in the first section (section [9.1.1](#)) of this chapter, the architecture for multiple description of a single input as defined in section [5.1.1](#) with integration of the visualization presented in the previous chapter. Then we present in section [9.1.3](#) the architecture for multiple inputs. As we explained in [section 7.2](#) we also aim to enable polyphony in the system and we showed the different principles to achieve such polyphony. The effective implementation of this polyphony and the musical conducting of it also required to imagine an efficient architecture in order for the *supervisor* to be able to drive the system (or enable the usage of scripts to do so). We explain first in [subsection 9.2.1](#) the organization for single output then we show [subsection 9.2.2](#) how we obtained a multiple output architecture.

Two main lines oriented our software architecture research and the design of this system. Firstly, we favor as much as possible an implementation which follows the conceptual principles that we presented in [Part II](#). Thus we opted for a very modular architecture. Each module implements a properly defined function of the system and the inputs and outputs of each module were carefully studied to be restricted to the most efficient scheme. We also took care to build *self-contained* modules as much as possible to avoid underground, hidden or complicated references from one module to the other. Naturally, this intention is utopian and a practical system necessarily includes some links between its different entities. But we took care to reduce as much as possible the number of these dependencies between modules. This control of the inter-modules dependency also favors the versatility of the system. The recombination of these modules into another configuration of the system to suit a different improvisation situation is greatly facilitated in practice with the ease for reconnecting the modules together in another arrangement

— the number of musicians may vary first but also as we have seen the number and nature of actual streams, the complementarity of the inputs etc. The number of “clones” and their features may need also to be adapted to the musical situation and many other aspects of the system.

Secondly, we aimed for this system a proven usability in *real-life* musical situation, from the *impromptu* improvisation in a garage with newly met musicians to the long organized concert on stage with a large audience and much more technical resources. Therefore we took care of the user interfaces of each module and the relative importance of the parameters in the playing situation. For example, the size of an Fast Fourier Transform (FFT) window of an extraction module is a very relevant parameter to properly configure the system before playing with a musician. But, because it is situated at the beginning of the processing chain and may influence the configuration of the rest of the whole system, it has absolutely no reason to be easily changeable along the playing and thus may be hidden or only informatively and discreetly displayed on the playing interface.

9.1 Multiple Perceptions

We defined section 5.1.1 our notion of an *input* which can be constituted of several *streams*. Then we described the different types of information extraction (section 5.1.2) and analysis (section 5.2) we can achieve on the different streams. We introduced the knowledge model that we build on part of the information extracted from the streams (section 6.2) and we explained how other part of the information can enrich this model (section 6.3). These multiple perception possibilities have been modularly implemented in our system. The architecture for the multiple description of one input is described in the next section (9.1.1) and the visualization of such input is shown. Then we explain how we duplicated those inputs to listen to several musicians in parallel. Because the primitive duplication of the inputs poses an obvious problem of visualization and interaction with the too numerous computer windows and parameters, we had to implement a centralization system to handle this duplication correctly.

9.1.1 Parallel Chains

As explained in the introduction of this chapter, we tried as much as possible in the architecture of our system to follow the conceptual organization of the processes. The chain for one description of an input exhibits three types of modules:

- Extraction modules are in charge of handling audio signal and converting it into a flow of raw information: initially pitch extraction module and spectra extraction module have been implemented. The harmonic information extraction has been added later.

- Grouping or clustering modules: specifically designed for each type of raw information, these modules implement exactly the different micro-to-macro units segmentation described in [section 5.2](#)
- Learning modules: these modules are nearly generic, that is identical for all the descriptions. They make use of the main three objects of the collection of externals implemented for our system and described in the previous chapter: *OMax.data* to hold the description's data, *OMax.oracle* to contain the graph and *OMax.learn* to write both structures including the building algorithm of the graph.

Additional modules are required for time-stamping data before feeding it to the learning module. There is only one type of time-stamping module which is duplicated for each description. This module also serves to segment the flow of information with higher level elements: *phrases* and *sections* are based on silence detection. The grouping module outputs an *activity* information which indicates if something consistent can be analyzed in low level description, if no stable units can be found then it supposes that the *activity flag* is zero and the higher level segmentation interprets it as a silence. *Phrases* boundaries correspond to silences shorter than *Sections* boundaries. Therefore *Sections* are higher level elements than *Phrases*. More importantly, the end of a section corresponds also to the stopping of the recording. Thus it ruptures the linearity of the medium with an interruption of the material. The mechanism to handle the starting of a new section has been described [section 5.2.2](#).

The time-stamping and all the timing issues are relying on another module: the recording module. This module is in charge of the whole medium: it records the raw audio or MIDI material of the *medium* and unwind the corresponding timeline which constitute the common reference for all the descriptions. In the case of an audio material, it relies on the writing of a `buffer~` object of Max/MSP. This architecture is illustrated [Figure 27](#) with a typical combination of the melodic (pitch) and spectral (MFCCs) descriptions on a single audio stream.

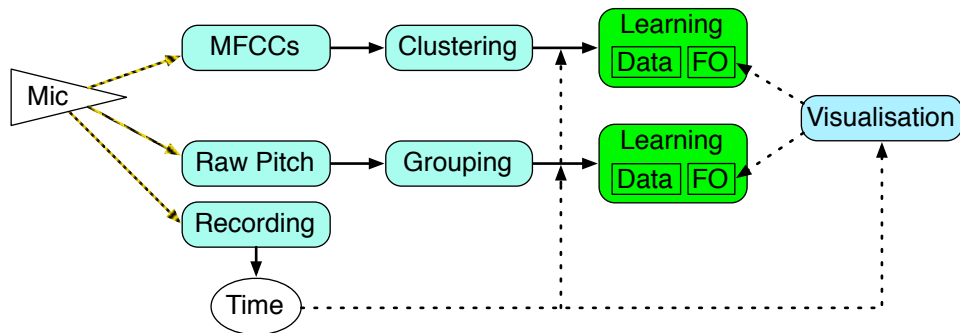


Figure 27: Dual Descriptions Input Chain

In terms of inter-module dependency, the extraction and analysis modules are totally independent and only receive their on-line information from the upstream module or the

audio stream. The recording module is also independent but possesses a name to be referred by the time-stamping and segmentation module (not shown on the diagram of Figure 27). Each of the learning modules also possesses a name to refer both the data and the knowledge model structures. Figure 28 shows an example of an Input patch interfacing the architecture described in this section. We can discern on this capture

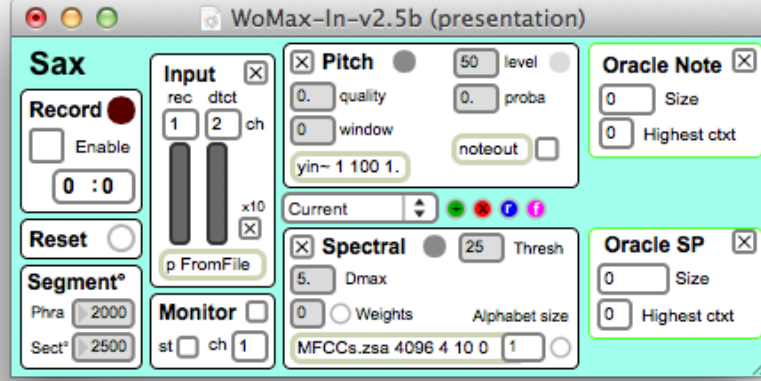


Figure 28: Dual Descriptions Input Patch

exactly the different modules described. The *reset* button is self-explanatory. The *Monitor* module enables to listen to the input streams and the small menu and buttons separating horizontally the pitch description (above) and the spectral description (below) are used to handle presets for the input parameters.

9.1.2 Visualization

A last module is added to handle the visualization of the whole input implementing in the whole architecture our research on pattern representation previously explained in section 8.2. This module is the most complicated of the input because it needs to be informed of the other modules: timeline & medium name is need to render the unwinding material, each description needs to be referred in order to display the current state of the model. As our first and main usage of these parallel description have been with only two descriptions: melodic and spectral descriptions, we adopted a symmetrization of the visualization presented in the previous chapter. The two knowledge models built are displayed above and below the common timeline of the medium. An example of such representation is given Figure 29. Below the horizontal timeline, the pitch model is depicted with the colored arcs. Above the timeline, the state of the spectral model is also visualized with arcs. We can notice that arcs representing pitch patterns are thicker than arcs showing spectral patterns. The thickness of each arc is proportional to the actual

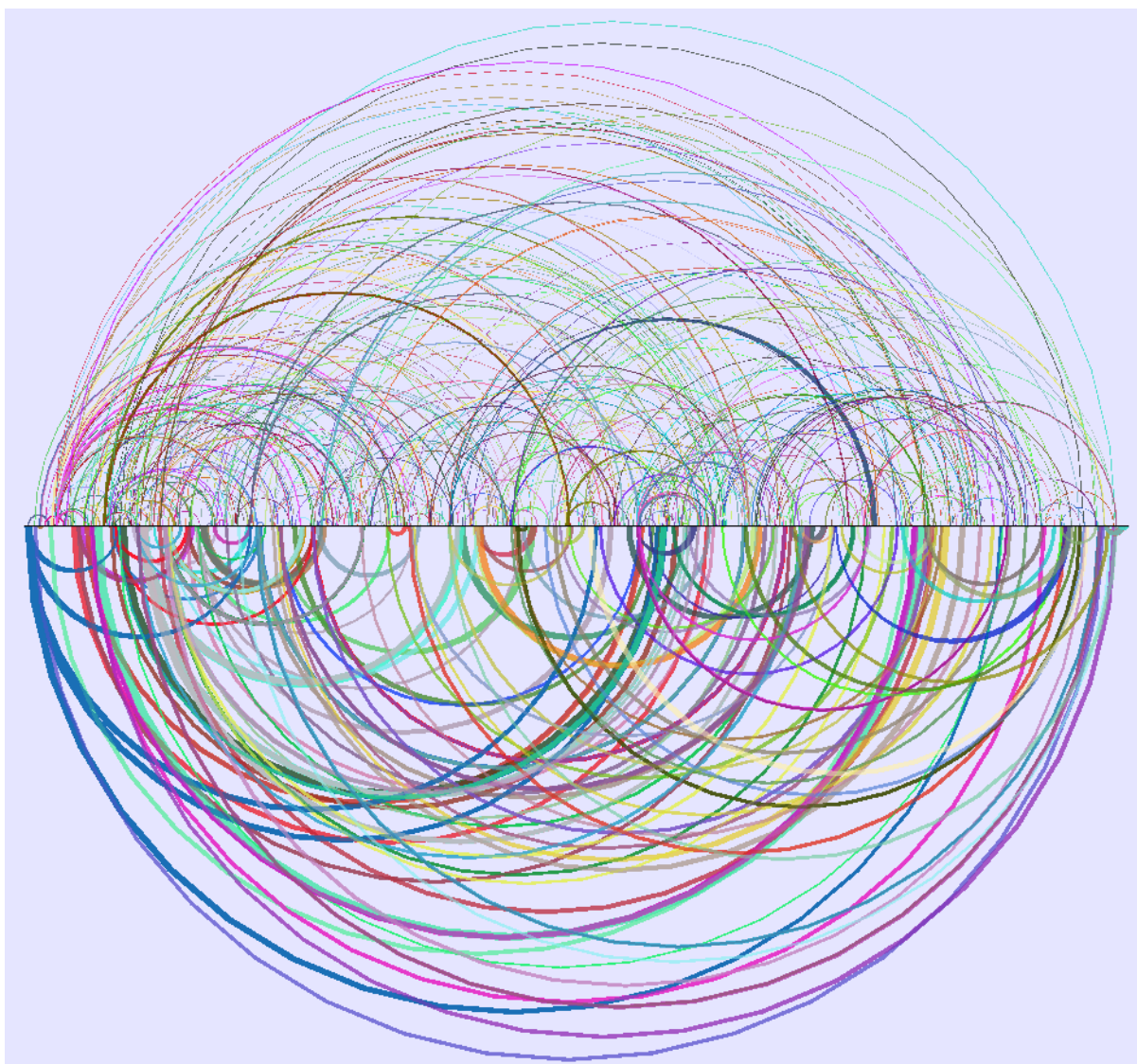


Figure 29: Dual Descriptions Visualization

duration of the pattern. As *notes* of the melodic patterns are typically of much longer duration than *timbre* of the spectral patterns, the patterns themselves are also of much longer duration. Hence the difference of thickness.

The whole input “program” we have described from the beginning of this chapter has been designed to function identically with or without the visualization. The visualization rendering is done in separated window and includes functions to capture the mouse, mainly to select regions directly on the visual timeline. This visualization may not be useful for prepared improvisation in which the original material is already known and timed. Removing the visualization modules keeps the software intact in terms of functionality and may enables to embed more easily the system in any other Max/MSP patch. This has been effectively done in several compositions in which our system was only a small part of the computer system.

9.1.3 Duplication of Inputs

In a collective improvisation, we wanted our system to be able to listen and learn from several musicians at once. Naturally, the first and main idea to do so is to duplicate the input part of our program and this does work if we take care of the names referring the underlying data and graph structures¹. However, the question of the interface and the visualization can not be as simple when a supervisor is controlling the system because of the profusion of computer windows it creates. We had to imagine a centralization system as presented in the next section (9.1.4).

Doing this duplication of the inputs strictly as proposed in this version of our system, and on the contrary to the discussion presented section 5.1.1.2, we suppose that the different inputs are totally independent. We will see how this will be improved in the second approach proposed in the next chapter (10). In the approach presented here though, as we duplicate the whole input patch (shown Figure 28), each instance includes its medium and the associated timeline. No correlation is done either between the timelines or in the analysis of the material. Figure 33 illustrates this architecture where each input embed independently its own recording and associated timeline and the extractions, analyses and modeling modules.

9.1.4 Centralization

As previously mentioned, for interfaces and visualization reasons, we still needed to centralize the different inputs and the names to address their internal structures. We adopted a software solution slightly opened up in the previous versions of the system: we implemented a *launcher* in charge of instanciating all the other parts of the program with the correct names. This launcher centralizes all the references. This means however that the usage of the launcher has to respect a correct order and the user has to define the input before creating the visualization part which will need the correct names. This order and the need for centralization may be a major drawback in the collective improvisation situation in which we wish for a system as versatile as possible.

9.2 Multiple Outputs

We explained in section 7.1 the two different parts necessary for the generation of a new musical discourse in our system: the *improvisation logic* (section 7.1.1, illustrated Figure 30) and the *player* (section 7.1.2). The former is responsible of generating a path in the knowledge system, the latter is in charge of effectively rendering this path in sound. We will see in this section that the implementation of these two modules poses important architecture problems especially in the Max/MSP environment. We will explain first the looped architecture we adopted for one “clone” then how we handle the polyphonic extension of this architecture.

¹Max/MSP does not enable proper handling of namespaces. Variables and references are always global.

9.2.1 Generation Architecture

We discussed in section 7.1.1.1, the importance of the *continuity* parameter and its possible dependency on the Factor Oracle graph. Originally, the coherence of the variations generated by the system was granted by the forward reading of a number of successive states in the graph before jumping — jumps and thus variations being justified by the suffix trees. In the version of the system that we present in this chapter, we respect this dependency and always attach an improvisation logic (see Figure 30) to a particular Factor Oracle. This is natural when there is only one Factor Oracle in the input but as discussed in section 7.2 and section 9.1.3, to gain versatility in the generation part we need to benefit from the multiple listenings and knowledge model built in parallel. An architecture to achieve this is proposed in the next chapter.

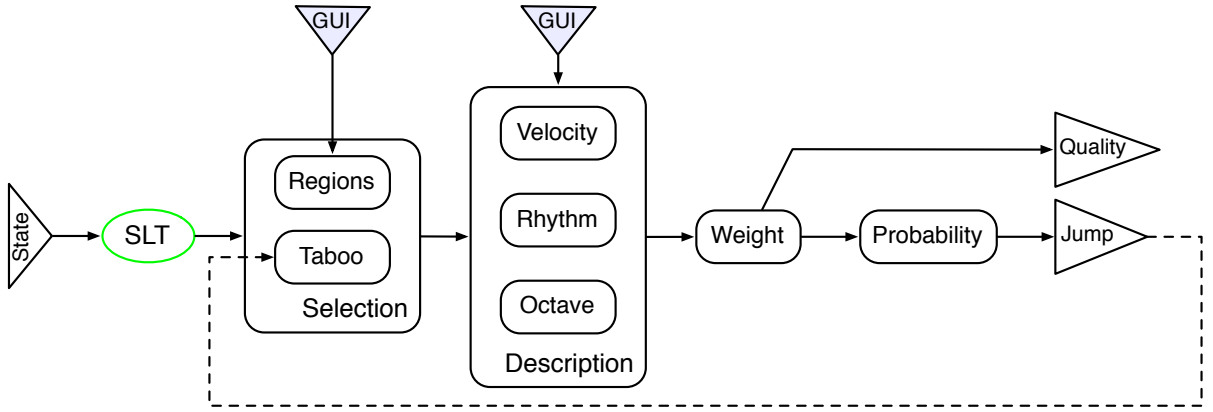
Nevertheless, in this first approach of the generation, an instance of an improvisation logic is linked to an instance of the knowledge model which means that a variation generated on the original material is always constrained to one specific description of the material, either melodic, timbral or harmonic. We present here the architecture of the improvisation logic which will also serve as a reference for the next chapter's architecture. The improvisation logic module is itself modular and its implementation depends for a few parameters and a few internal modules on the type of description. The general scheme of the navigation is illustrated Figure 30. The continuity, that is the forward reading of



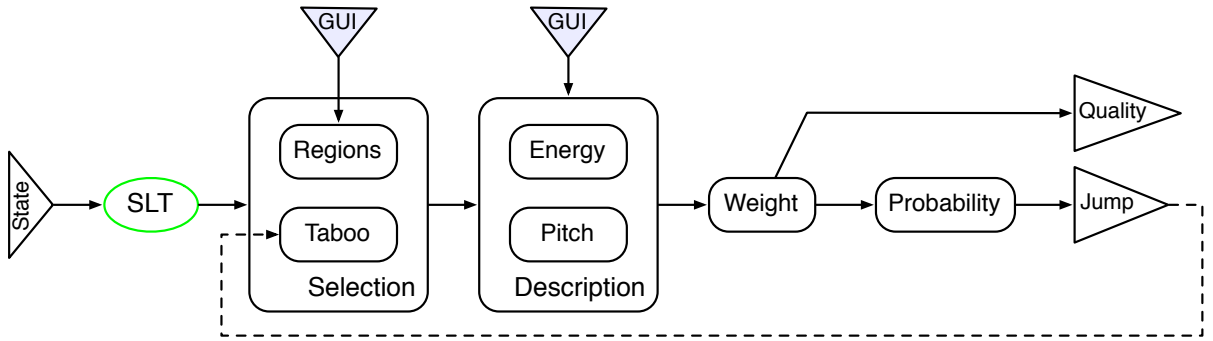
Figure 30: Generic Architecture of the Improvisation Logic

several states in the graph does not require any specific function except a counter. On the other hand the jumping mechanism necessitates first to know the current position — this may appear as trivial but we will see that it is the primary reason to design a looped architecture. From this position, we collect all the possible jump destinations from the jump trees (see 7.1.1.2) and gather them. Then we radically exclude some of these solutions notably because of a taboo mechanism avoiding strict loops in the graph and because of the region mechanism explained section 7.2.2.1 enabling to control the content of the computer generated variation. Then we add to the description of each solution some complementary information for example coming from the pulse, energy/dynamics or harmonics annotations described earlier in this thesis or from ad-hoc calculations depending on the Factor Oracle used — an example of the ad-hoc information is the octave if we computed the knowledge model with a 12 semi-tone alphabet. Based on these descriptions of all the possible solutions, we compute an overall weight for each solution — which may reflect also choices of the user to emphasize certain aspect of the material

through the graphical user interface (GUI) — and we draw randomly among one of the best solutions i.e. those with the highest weight. The weighting systems also enables to output an indicator of the *quality* of the solution picked, compared to all the other possible jumps. Figure 31 show the specialization of the improvisation logic module in the case of pitch and spectral knowledge models on the input. This specialization shows the different complementary descriptions of the solutions. The *rhythm* information presented in this version correspond to the *rhythmic coefficient* of [Assayag 07]. This coefficient (C_r) rely on the durations d_x and d_y of two patterns and is computed with this formula: $C_r = |\log(\frac{d_x}{d_y})|$. The smaller the coefficient is, the closer in duration the two patterns are. This coefficient enables to get a simple notion relative *density* to avoid rhythmical ruptures in the generation.



(a) Diagram of the Pitch Improvisation Logic



(b) Diagram of the Spectral Improvisation Logic

Figure 31: Specialization of the Improvisation Logic for Pitch and Spectra Descriptions

Anticipation To enable the player to render *on time* the musical discourse, that is without waiting for the next event to be decided we actually need the improvisation logic to be early in its walking through the graph. In this architecture, the improvisation logic is a few steps ahead thanks to an anticipation window. These few steps means actually a few *states* ahead in the walking of the graph possibly with jumps and not a few *jumps* ahead because contiguous portions of the original material between two jumps may represents

seconds of music. As we aim for a reactive system, we want to keep this anticipation as short as possible.

In this version of the system, we store the improvised sequence in the same kind of data structure as the description data through an new instance the external objects for OMax (*OMax.data* and *OMax.build*). The data stored for the improvised sequence are the description data of each state read from the original sequence including the states' number (in the original sequence) which enables to retrieve the path followed in the graph by the improvisation logic. While the improvisation logic writes this sequence, the player reads continuously this data structure to discover the portions of the original material to be read. Naturally, we do not want the player to introduce any audio artefacts when the portions to read are contiguous so only jumps triggers the typical crossfades used by the player to concatenate audio chunks.

A specific kind of player implements the time-stretching feature (with or without transposition) described section 7.1.2.2 thanks to *SuperVP* technology. This feature is directly under the control of the supervisor of the system thus we can absolutely not make any assumption on the actual speed of the unwinding of the generated discourse. This is why the player has to trigger the walking of the graph. If the speed is higher then the traveling of the graph must be faster as well. This closes the loop between the improvisation logic and the player: the player triggers the computation of the next steps in the path, these next steps are written by the improvisation logic in the sequence read by the player. Figure 32 illustrates this whole loop with the internal elements of the improvisation logic and player. Unfortunately, the loop architecture is intrinsically a bad and risky implementation in the Max/MSP environment because of its event-base scheduling. It creates very often instantaneous infinite loops which can provoke stack overflows. The fine adjusting of this architecture for improvisation logic and player has been very sensitive and time-consuming. This maladjustment between the architecture design and our programming environment is also one of the reason why we favored a different approach of the generation principles of our system in a second phase of our work.

9.2.2 Polyphonic Extension

Broaching the question of polyphony for an improvisation computer system raises the question of the number of voices. This question concerns three different aspects of the system.

- On the musical side, it is important to measure the impact of polyphony on the overall result in the improvisation. In particular, the number of separated voices influence drastically the room taken by the computer in the global musical matter and the space left for the other musicians. Too many voices may be very soon overwhelming especially when they are meant to create mirrors and variations of the acoustic musicians discourse that is second voices and not background elements.

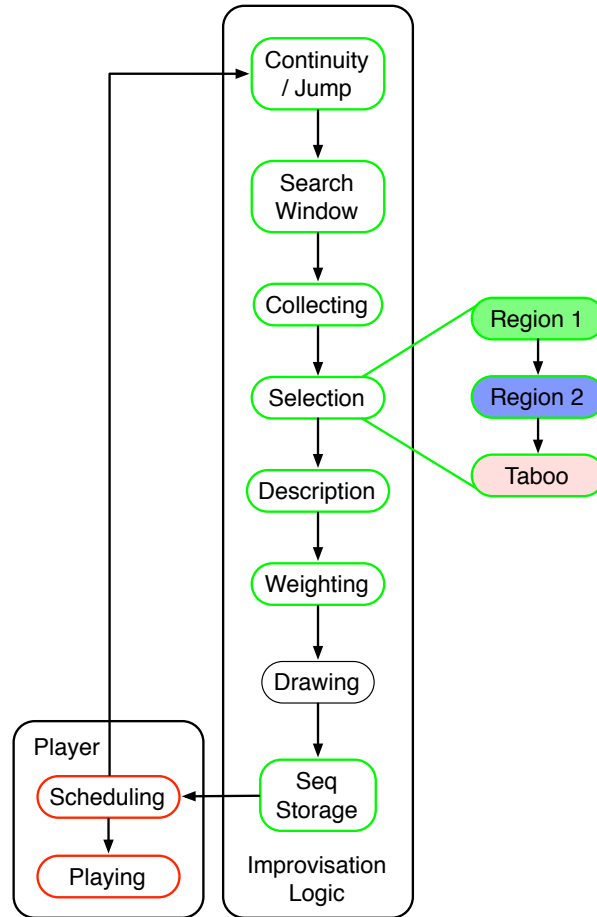


Figure 32: Generation Loop of with Improvisation Logic and Player

In our system, oriented towards the direct and balanced dialog between acoustic musicians and computer generated improvisation, we do not need to include a profusion of polyphonic voices.

- We choose for our system an *instrumental* direction which means that the computer has to be controlled by a supervisor even if only sparsely or with very few interactions as *start* and *stop*. It is then of responsibility of the supervisor to drive and control the different “clones” sounding. A large number of “clones” at once requires very different interface and interaction facilities to handle them with responsiveness and pertinence. We preferred to limit the number of simultaneous voices and favor individual and finer control of each of them.
- Finally a third remark on the software architecture level also supports this simplicity: the *players* which manipulates directly MIDI or audio stream are CPU consuming especially when equipped with the time-stretching and transposition possibilities of a phase-vocoder module. Thus we need to avoid multiplying their instances.

The biggest setup of the system we tested had 4 inputs (one per musician) and included up to 9 simultaneous players. However the playing of 9 distinct voices simultaneously

was never used. A typical improvisation session or concert make use of 1 to 4 or 5 voices maximum.

To achieve this prudent polyphony, we designed players which can be attached to an improvisation logic navigating any knowledge model. The relevant information needed by the player once attached to an improvisation logic is only: if the next portion to play in the medium corresponds to the next state in the graph or if there is a jump to plan. In that latter case, the editing points (the dates of the origin and the destination of the jump on the medium timeline) are required. As they manipulate audio streams, the player modules must be created at the loading time of the system because of Max/MSP handling of the digital signal processing chain². To attach an improvisation logic to a specific player, we make use of the same centralization system as in 9.1.4. The launcher which references all the inputs and knowledge models also references the players and lets the user choose and load an improvisation logic for a given player. The connection between the logic and the knowledge model and between the logic and the player is done when the logic is loaded. However, logic modules manipulate only symbolic information so they do not modify the audio chain. Therefore their loading can be achieved transparently and smoothly while the system is running. This is also a reason for the strict separation of the improvisation logic modules from the player modules.

Thanks to this architecture of fixed players interacting with different improvisation logic, we can also configure the system to have several players attached to the same improvisation logic. In this case the players will read the same path in the graph that is the exact same variation on the original material. At first a side-effect of our implementation this became a very interesting musical feature while actually testing with musicians. Indeed the strict echo or the synchronizations and desynchronizations of the same material is a classical effect in music. We enrich this possibility with a simple mechanism to obtain strict and voluntary synchronization of two players. One player sends automatically to the other its exact position at each step of the generated path. This feature is also particularly relevant when making use of the transposition possibility of the *SuperVP* player. We obtain this way an instantaneous and controllable harmonization of the computer generated discourse.

Figure 33 exemplify a possible setup of the system in this architecture. It includes 4 inputs with several descriptions on each of them (combined into one chain of modules to avoid overloading the diagram) and 4 players. Only 3 improvisation logic (abbreviated as *Impro* on the diagram) are loaded in this example to illustrate the possibility for two players to render the same path in graph. Plain arrows denote the loading role of the *Launcher*, dashed arrows indicate underlying reference to the structures through pointers. For the sake of clarity, the references needed by the visualization module are not drawn. The red framing of the improvisation logic stresses the *hot* loading possibility of these modules.

²Maxers know that the modification of any audio object or connection while the audio processing is running ruptures the chain and unfortunately outputs undesirable clicks.

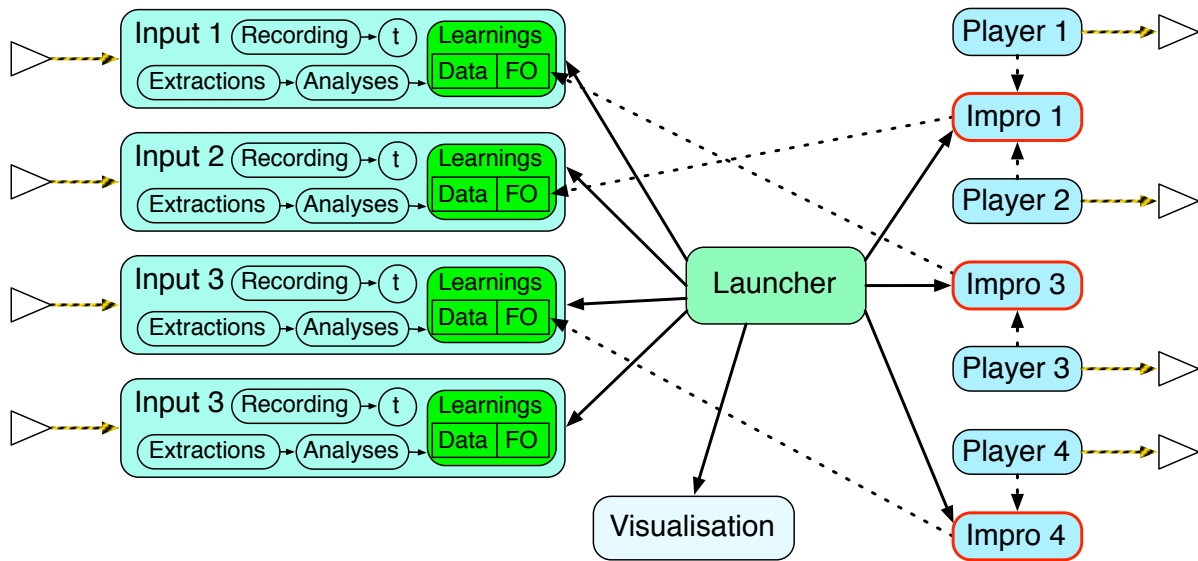


Figure 33: Summary of the First Multiple Architecture of the System

We described in this chapter a first architecture of our system capable of handling the multiplicity of the perception part of the system — multiplicity in streams, descriptions, inputs etc. This design also takes on the usage of this multiplicity in the generation part of the system. This architecture has been thoroughly tested up to a stable version used in a *real-life* concert. It has also been the base for two other important software realization of the system. A public version of the OMax software has been released 2010 and incrementally updated since then. The architecture of this public version is strongly based on the architecture described in this chapter, but it fixes the numbers of *improvisation logics* and *players* to enable the user to take in hand the system more easily. The documentation of this version is joined to this thesis in Appendix C.

A research team in Concordia University, Montreal, under the direction of S. Bhagwati has built a very impressive “Comprovisation” system with the version of our system that we presented in this chapter. *Comprovisation* is a portmanteau word coming from *composition* and *improvisation*. It describe the musical research of S. Bhagwati who develops very structured and prepared pieces still relying on the improvisation capacities of a musician — and a computer system in this case. In a project named *Native Alien*³ in collaboration with N. Navab, they developed, around a customized version of our system, a much bigger computer system integrating several compositional aspects as well as other computer techniques as sound synthesis. Their *sound environment for soloist and 8 comprovising computer channels* make intensive use of all the parts of our system to listen to the acoustic musician and generate eight voices precisely controlled by N. Navab and following a generated score. This realization is a successful usage of our research — as much technically as musically — which has been played with leading musicians as Lori Freedman (clarinets) or Vinny Golia (saxophones).

³<http://matralab.hexagram.ca/projects/native-alien/>

The architecture for our system that we presented in this chapter has been particularly well tried and gives pertinent results. However we showed that one improvisation logic of this architecture is still attached to a specific instance of the knowledge model which we consider as one of the serious limit to the benefit we could obtain from the many perceptions possibilities we developed. For example the transition from one model to the other can not be done smoothly in realtime. A skilled supervisor can simulate such a transition by overlapping two players referring to two improvisation logics walking the graphs of two descriptions of the same input but this remains a very audible artefact in the playing with the system. In the next chapter we present a new architecture to overcome this limit and allow even more versatility for the computer generated discourse.

Chapter 10

Database Approach

The previous chapter presented one of the first stable architecture for our system which accomplished the multiple listening and polyphony objectives of our system. However we also saw some drawbacks of this architecture: the intricated references, the following need for centralization and the lack of fluid transition from one description or input to the other. We present in this chapter a new architecture for the system that resulted from the careful analysis of all the principle and evolutions of the system. This architecture proposes to generalize and imply more systematically the principle we have seen. It starts with the gathering of all the instances of the knowledge model into a database oriented design. A disconnection between the low level description and the model building is needed to achieve this architecture. We present those two point in [section 10.1](#). Then, we make use of this database approach to formalize the query, filtering and sorting of the jump solutions gathered thanks to the knowledge models ([10.2](#)).

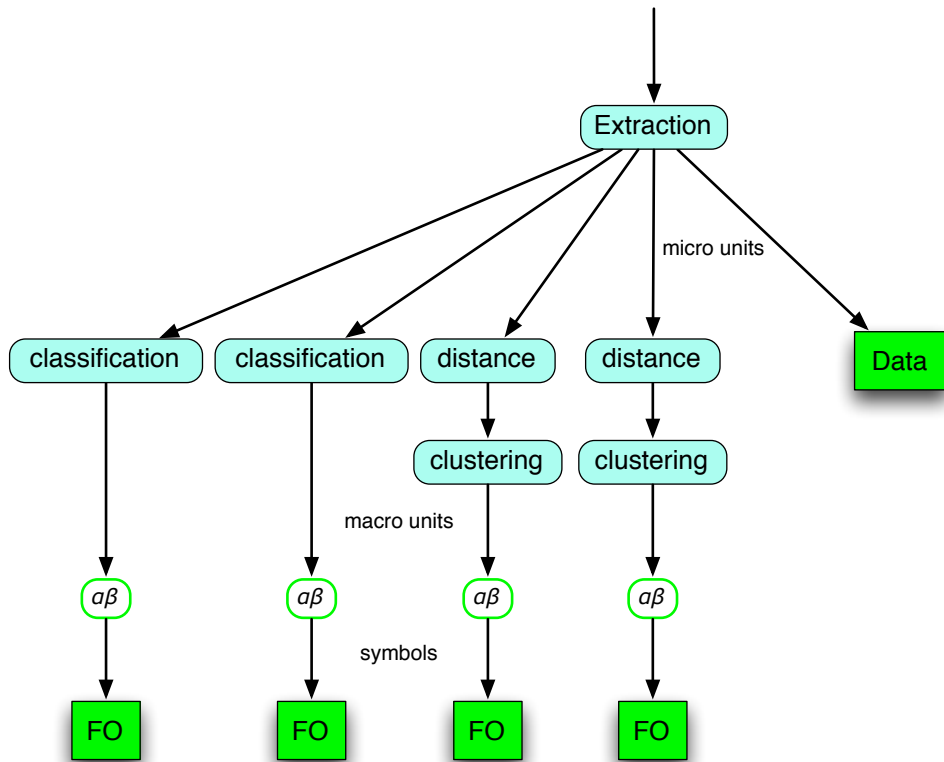
10.1 Knowledge Query

In the previous chapter, we saw how we first multiplied the descriptions of the streams then the number of inputs. We also explained how we initially tried to benefit from these new possibilities on the generation part thanks to a modular separation of the improvisation logic in charge of exploiting the knowledge models and the players effectively rendering the computer based improvisation. In this chapter, we will see how we came back on this separation to enhance greatly the usage of the multiplicity of descriptions. We will show that through an evolution of the data & graph structures enabling the disconnection of the information analyzed from the audio and the model built on this information, we favor even more the building of several models in parallel on the same data and we facilitate the usage of these models concurrently. We first describe the differences of the implementations of the data and graph structures in our software environment ([section 10.1.1](#)) then we explain how we interact with these new structures in parallel ([section 10.1.2](#)). Finally these changes lead us to a database approach of the concurrent knowledge models ([section 10.1.3](#)).

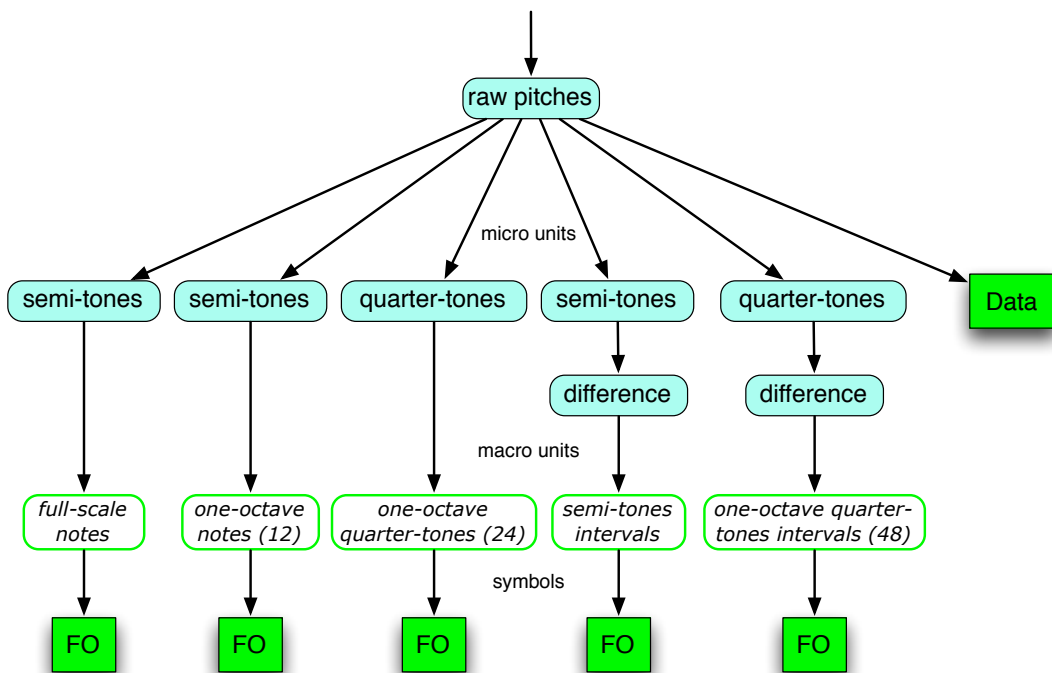
10.1.1 Independence of the Structures

In [section 8.1](#) we presented the structures we implemented — in the form of external objects for Max/MSP — to store and interact with instances of the knowledge graph and to hold the information coming from the segmentation of descriptors of the input streams. In particular, we explained that for one description of an input stream, the data structure recording the description’s information (for example pitch information) and the graph modeling the patterns found in this description share the same segmentation if not the same data structure in memory. Thus the same indices are used in both graph and data structures. In the evolution of the software architecture that we present in this chapter, we propose to rupture this strong connexion between the descriptors data and the model build on the macro-units formed with these data. Moreover, we choose to send to the data structure the raw and unsegmented information coming from the different extractions of descriptors (micro-units) for each stream. The data structure becomes this way simply a specialized container for consecutive (and rather short) frames describing one aspect of the content of an input stream. Each description is held in one instance of this renewed data structure e.g. one *OMax.data* object stores the raw pitch information, another instance stores the raw spectral information etc. While disconnected instances of *OMax.oracle* objects can hold several graph built on macro-units formed either on pitch and/or spectral description for example.

We still need to form the macro units and label them to build our knowledge model as explained in the first part of this thesis. However in this renovation of the system, we specialize this processing to the building of one specific instance of the graph. This way, we enable the effective building of several knowledge model on the same content (or description) but with different classifications, clusterings, segmentations and labelings ($\alpha\beta$) as illustrated [Figure 34](#). For example, on pitch description, raw information are gathered into notes and the patterns of notes can be modeled in two different graphs depending whether we label them with a full scale of pitches or with a one-octave scale of pitches. Doing so, we amplify again the multiplicity of descriptions available in the system but we will see in [section 10.1.3](#) how we changed radically the approach of this multiplicity to generate new musical discourses. This theoretical separation of data and graph structures has some implementation consequences. The first of which is the removal of the *OMax.learn* object which was in charge of writing both the data and graph structures in parallel. This object is now obsolete since each structure is independent. This independence also means that the algorithm to build the Factor Oracle has been included in the *OMax.oracle* object itself. The same way, data are addressed directly to the *OMax.data* object to store them in the data structure. We will see in [section 10.1.3](#) that the *OMax.build* object is also obsolete because a computer base generation does not correspond anymore to the walking of one graph and one description but to the query of several graphs built on several descriptions. [Figure 35](#) updates the diagrams of OMax external objects for Max/MSP with these changes. A second major change done to the Factor Oracle structure is a side effect of this real separation between description data



(a) Generic Segmentations and Clusterings for Parallel Modeling



(b) Example of Parallel Analysis of Pitches Leading to Several Models

Figure 34: Parallel Segmentations Leading to Unaligned Models

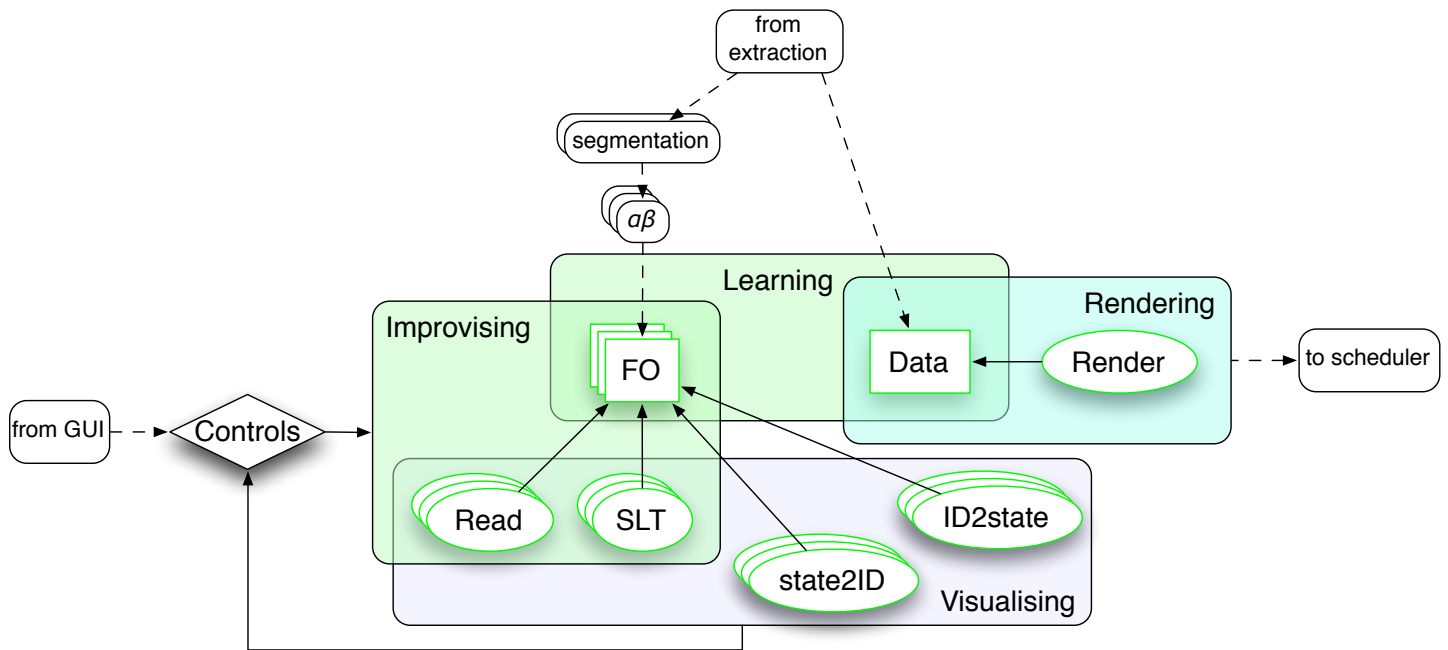


Figure 35: Renovation of the Collection of OMax's Externals for Max/MSP

and higher level modeling: as a Factor Oracle structure does not refer to a specific data structure anymore and runs internally the incremental building algorithm, we have to provide it with the letter of our labeling for each element to learn. Indeed, Factor Oracle algorithm includes a strict comparison of the label to identify equivalent elements. But we wish to preserve the *agnostic* implementation of Factor Oracle structure — i.e. not containing any information external to the graph — so we choose to provide an abstract letter encoded in an integer to the *OMax.oracle* object, whatever the alphabet is. In the case we want to store the actual content of the alphabet, we do that externally in the module in charge of the labeling.

10.1.2 Prominence of Medium

We have just explained that we broke the link between the data structure holding the description information and the macro-unit segmentation, labeling and the knowledge model. Another important consequence of this rupture is the crucial need for a common reference to all these descriptions, segmentations and graphs in order to use them all indiscriminately. As we stressed in the very beginning of the second part of this thesis, the medium — that is the effective recording of the musical material and its associated timeline — constitute the ground reference to which every process ensues. Thus in this architecture, we make the medium and its associated timeline even more essential as a common reference for every description and every knowledge model.

On the data structure side, every micro-unit is thus timestamped with the medium timeline and furthermore, this structure is now solely indexed by these timestamps. As no more higher segmentation is stored in this structure, no other index or referencing

would be suitable. For example, in the case of spectral extraction (MFCCs) all the successive frames are stored directly into the data structure and addressed through their corresponding dates in the medium without consideration of higher level segmentation. Various clustering of these frames may lead to different macro-units segmentations for the parallel building of several models. For the same reason the two hashtables converting timestamps into state number of the graph (former *OMax.date2state* object) and state of the graph into timestamps (former *OMax.state2date* object) have to be attached now to the *OMax.oracle* object. Indeed the macro-units are now only referred in the knowledge model (through their symbol). We renamed the two objects interfacing these hashtables into *OMax.ID2state* and *OMax.state2ID* to make explicit the future use of external material which may not have been learnt in the system (external audio files with off-line analysis for example). In this future case, the *ID* will not only encode the timestamping but also an identification of the corresponding medium (buffer, audio file, MIDI file. . .).

Pursuing this unification of multiple descriptions and model instances use in the generation part of the system as well, we propose in this version of the system to abandon the linkage of the *improvisation logic* with a specific Factor Oracle. In particular we revised the expression of the *continuity* mechanism which alternates between contiguous reading of the original material and jump. We reimplemented this alternation in form of a separated module based directly on time references. There are several advantages in this view: first it is totally independent from any instances of the model. Secondly, it enables a more natural expression of the *continuity*: we can directly set the system to find new variations every x (milli)seconds whatever the material is. Thirdly, it avoids having very long portions of material played unchanged in the case of very long macro-units (typically, long notes of several seconds each) which correspond to very few states of the graph. However, to find some relevant jumps possibilities in real-time along the generation, this *timing* must be linked with the rendering of the player. Indeed, we need to know *where* the current read head of the generation is in the original material to find jumps which origin will be accessible and respects the desired *continuity*. Therefore this new *continuity* module can not be totally self-contained and needs to be attached to a player for real-time generation.

10.1.3 Knowledge Models as Database

The goal of the changes presented in this chapter is to orient our view of the multiple knowledge model instances towards a database approach. The collection of these graph constitutes from a farther distance a huge ensemble of links connecting several parts of the material together thanks to several descriptions. Thanks to these links, we can join two portions of the material to create a variation. Then we can see this generation mechanism as the regular query of this database for possible jumps depending on the current position of the read head. To be efficient, we needed to unify the expression of these queries so

that every instance of the model can participate to the general search of solutions. This is why we favored the common medium time reference as explained in the previous section.

This database view overcomes the major critic we addressed to the previous generation architecture of our system: one “clone” is no more linked to one specific description of the input material. We query in parallel all the knowledge models built and gather the jump possibilities of all of them before effectively playing one of them. We can of course reduce the query to one model only and we emulate in this case the former behavior of the *improvisation logic* necessarily locked in one Factor Oracle graph. But we can also change over time the instance of the knowledge model we want to query without breaking the musical discourse of the “clone”. Doing so, we can switch seamlessly from a melodic variation on the material to a timbral or harmonic reinjection of this material for example. With this architecture, we enable a much finer control over the core principles of the generation of each separated “clone”. This new database view also enables to imagine off-line queries to search for particular solutions which may come from compositional ideas or any kind of preparation. For example, we may want to find with off-line algorithms paths in the graph with particular constraints, or simply the shortest path that arrives at a precise point in the material. The algorithms used may not enable real-time search of these paths but our database view allows to query the whole knowledge to come up with a suitable solution concurrently with the real-time generation (see Figure 37). This view enables as well the future inclusion of any kind of knowledge model in the database as soon as they provide continuation links between two parts of the material. Figure 36 summarize the new design of our system following the database approach. It shows two inputs with their streams and medium and the gathering of all the knowledge models. Three instances of the generative part are shown. We describe in more details this part in the the next section.

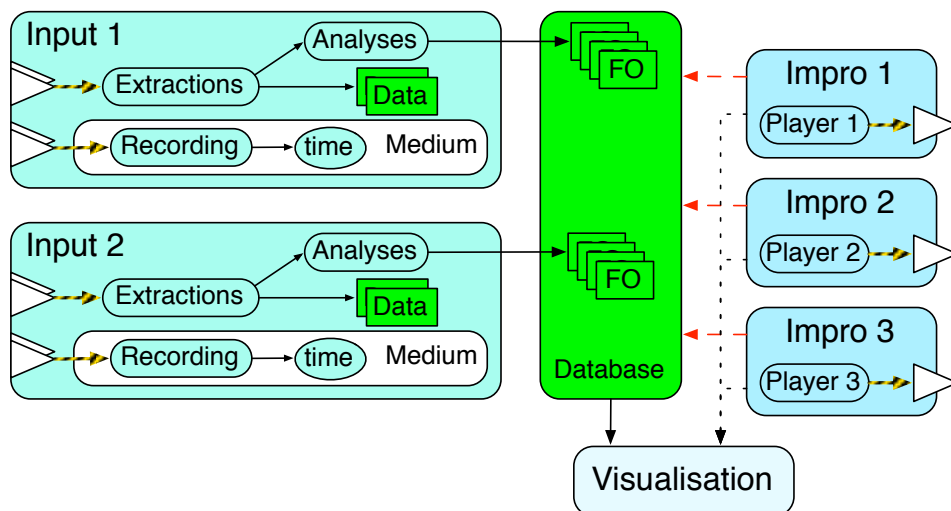


Figure 36: Overall Architecture of the Database Approach of the System

10.2 Filtering & Weighting

In the previous chapter we presented section 9.2.1 the generation part of system in a version which included three main steps:

- a strict selection of the jumps found in the graph
- the weighting of the accepted solutions
- and finally the draw among the best jumps the one to be effectively rendered.

This selection and weighting elements of the former generation process anticipated the generalization we achieve in the database oriented version described in this chapter. We give more detail about the strict selection and the weighting in this section.

10.2.1 Data Flow

In our work to unify the generation process for all the descriptions, we also generalized the modular implementation of the generation. Each step in the process correspond to a specific module and whenever it is possible, the same generic module is used for all the descriptions. These modules follow the general scheme illustrated Figure 37. On the contrary to the looped architecture presented section 9.2.1, this new architecture is based on a continuous data flow principle. No data structure is written to hold the generated paths and the player is not looped around the improvisation logic. As soon as it is enabled, the player reads continuously the original material and renders a crossfade with another portion of the material whenever it is sent the editing points (origin and destination dates) of a jump. The *continuity/jump* module is in charge of handling the alternation between the continuous reading of the material and the search for a jump. It queries the player for the current position in the reading of the original material then anticipates and outputs time-windows corresponding to the expected origin of the next jump. These search windows are send to the knowledge database. A generic module, given a pointer to one of the instances of the knowledge model, is in charge of searching in the graph the possible jumps in this model. The number of instances of these search modules is not limited. All the solutions are gathered, indexed by the date of the editing point (origin of the jump). Then, they go through the filtering and weighting modules that we will describe in the next sections. Once thoroughly described and ordered depending on their weight, the same principle as before is applied: we either take the best solution or draw randomly among the three best if we want to add variety to the process.

There are several benefits in this new architecture. Firstly we disconnected every possible module from the knowledge model(s) thus we enable a general scheme for the generation independently from the model(s) used. This enables to include several instances of the Factor Oracle graph but it may also open to different types of models. We also opened the loop of the previous architecture which helps the reactivity of system: we can now choose efficiently if we want to plan a particular path (*Off-line Search* module of

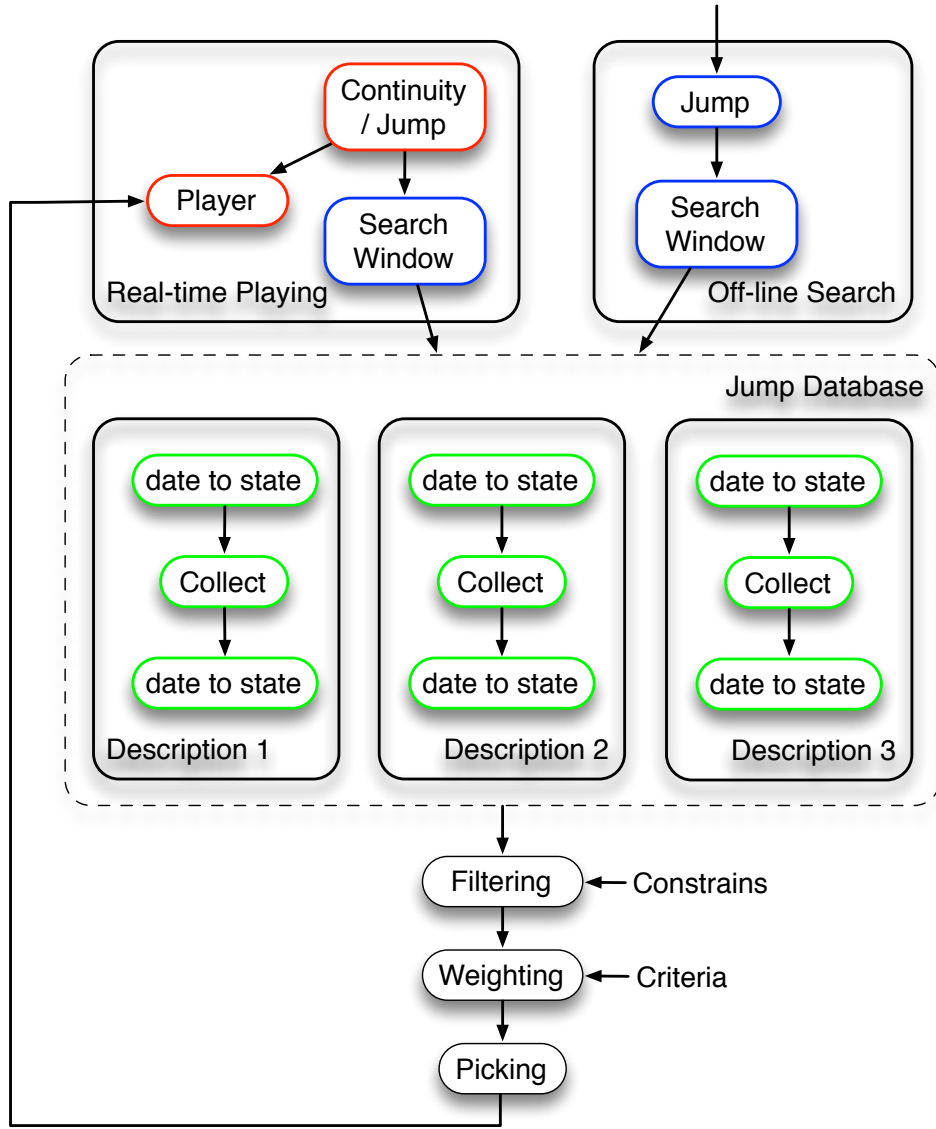


Figure 37: Diagram of the Database Approach of the Generation

Figure 37) or if we generate it on-the-fly. The player does not need to read the generated sequence in a data structure and can effectuate it whenever it is needed. The conducting of this path along the improvisation is thus more efficient thanks to a lower latency between the decisions and their actual realization. To store an anticipated path if needed, we just have to store the next editing points and send it to the player successively. With this database approach, we are also free to make one “clone” improvise on one or the other description or combine several descriptions by just enable the search in the different graphs. This combination or switching between descriptions (and models) can happen at anytime. It can be decided by the supervisor along the playing or planned depending on the expected material. As soon as a model is *activated* in the database, the next search for jump will include the result given by this model as well. Finally, we generalized the filtering and weighting mechanism to favor particular musical aspects.

10.2.2 Filtering

In the data flow process of the generation we gather the jump possibilities justified by the collection of models and send them first to a strict selection process. This selection can be composed of several modules each performing the filtering of solutions on different criteria. We made generic the programming of these modules: each one receives the solutions and outputs the accepted solutions on one side and the rejected solutions on another side. Each module has also a *bypass* function to deactivate it and let through all the solutions.

10.2.2.1 Regions

This formalization of a filtering module leads us to an important renovation of the regions system. In this database view of the system, a region is just a simple filtering module as any other, based on an allowed portion of the original material. With this view, we could program a generic *Region* module and instantiate it as many time as we wanted to obtain as many regions as needed for our use. We could also imagine and realize *negative regions*, that is specify directly forbidden portions of the material instead of specifying inclusive regions. Depending of the connection between these modules we can also easily achieve the union or the intersection of the regions. Musically, this means a great variety in the way we want to constrain the generation to utilize specific material.

This renovation of the region mechanism — which was previously definitely included in the improvisation logic — also enable to review, enhance and generalize specific usage of the regions. One musical use in particular gave very pertinent results since the first versions of the system: the continuous update of one of the region to the last x seconds of the learning. This way the region *follows* the present time of the running material (when the system is learning) and when we lock the generation in these x seconds we achieve a musical *smart delay* effect: the computer generated discourse reuses only the most recent material learnt but still recombines it into unheard variations. Then the computer seems to *follow* the improviser it learns from. We generalized this mechanism and made a module capable of updating any region to either follow the current last x seconds of the recording or *extending* any region continuously to include the most recent material while preserving its left-side (past) boundary.

The automatic alternation between the *follow* mode presented in the previous paragraph and the *free* mode in which the generation is allowed to use all the material learnt has also given very successful musical results. Again, thanks to this renovation of the regions, we generalize this principle and implemented a module capable of alternating automatically between any two regions. It is dotted with a settable period and a settable balance ratio between the two regions. This way we achieve the musical effect of regularly combining ideas coming from different part of the material learnt which is a very usual process in improvisation.

Phase The database approach of the generation and the generalization of the filtering of the solutions gave us the correct framework to implement a mechanism to preserve the pulse characteristic of the material as explained in section 7.2.2.2. This selection modules compares the pulse annotation of the material at the origin and the destination of the jump, in particular the *phase* in between two beats (see 5.1.2.4). It accepts jumps only if the phase difference (between the origin and the destination of the jump) is below a certain threshold. This way the original pulsation of the material is preserved. The same idea can be applied to the phase difference between the destination of the jump and an external running phase. This enables to synchronize the generation to an external pulse. If this external pulse information comes from the extraction of the current common pulse in a band for example, then we are able to synchronize our computer generation to this common pulse.

10.2.3 Weighting

In the data flow process for generation we chain the filtering of solutions depending on regions or phase constrains to the weighting system already outlined in section 9.2.1. As we did for filtering modules, we generalize the prototype of a weighting module. This type of modules receive the accepted jumps possibilities through a list of origin and destination dates and computes a weight depending on some specific quality of the jump. The weight is append to the origin and destination dates so that at after going through all these weighting modules we just have to sum the separated weights to compute an overall quality of the jump. For each module we also enable the possibility to choose the spreading range of this weight. By adjusting relatively each weight range, we can thus favor one or several aspect of the material compared to the others.

Context In the previous version of the system where the generation logic was attached to a specific Factor Oracle graph, the context length (the *lrs* i.e. the length of the common pattern at both end of the jump, see 7.1.1.2) was the primary and unchangeable criteria to discriminate best jumps. In the database approach, the *lrs* is one of the modules of the weighting process. The consideration of this quality is settable, as previously explained, the module's interface enable to choose the mapping range or the weight.

Energy We described section 6.3.1.2 how we can benefit from the amplitude information of the pitch extraction algorithm or from the first coefficients of MFCCs vector to get an annotation describing the musical dynamic. We also explained section 7.2.2.2 that we can use this information to bend the musical discourse towards desirable content. We implemented two weighting modules to do so with the energy. These modules depends on the MFCCs extraction. The first module is a comparison module i.e. it compares the energy at the origin and the destination of the jump to favor smooth jumps, jumps which have close dynamics. The second module however is a *conducting* module which enables the supervisor to favor jumps arriving to a specific range or energy. A small anticipation

window is given and the modules computes the mean energy at the destination of the jump over this anticipation window. Then it attributes a weight depending on the proximity of this mean energy to the dynamic range by the supervisor. These energy weighting modules which naturally can be enabled individually are used both to favor a smooth generation and the fine conducting of the content.

Register The same anticipation mechanism as the one explained in the previous paragraph for energy can be applied to conduct the musical register of the computer variation. Specifying a window (in number of notes) the mean pitch of the next few notes at the destination of the jumps is computed and a weight is calculated to favor high or low pitches. As the pitch and segmentation in notes depend on the pitch extraction, this module is only available if pitch extraction is effectively present and active. Through this module, the supervisor of the computer can control the register favored by the computer and for example create an inclination towards the material learnt which make use of the bass register of the instrument. The database approach of our generation system enabled to enhance greatly the possibilities of our system. Through the systematic query for jumps in the models, we gather more solutions than previously and we can filter and weight these solution according to musical criteria. The supervisor can this way conduct more finely one “clone” by choosing the type of variations (i.e. the knowledge model(s) used), emphasizing one or several musical aspects of the solutions: pulse, energy, register etc. [Figure 38](#) shows the interface of this new generation system.

We did not discussed the possibilities for polyphony in this architecture. However we did implement several of the generation process we described without any kind of interference. Thus we achieve polyphony simply by duplicating the generation architecture described in this chapter.

The version of the system presented in this part is the latest we tested. This database approach and implementation have given very successful results both on the musical rendering and playing with musicians and on computer aspects (performances, ease to adapt etc.). We achieve through this version a versatility and a very quick adaptation to the performance which is extremely useful in the case of improvisation. We can now integrate our system to very various sorts of musical contexts from the free abstract timbral duet to the structured and pulse Jazz band. All the musical situations we have tested along our work are listed in the *List of Musical Experiences* at the end of this thesis. The next chapter details some of these experiences and explains how it influenced our research. Naturally, we have many more prototypes and refinements already implemented but which have not been tried. We also have many more ideas to enhance and improve this system.

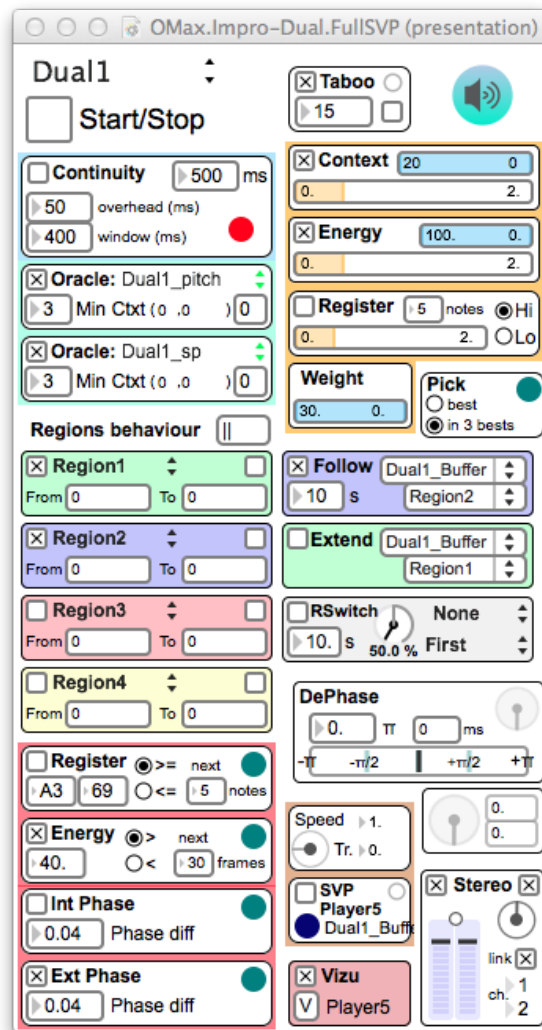


Figure 38: Interface of the New Generation Part of the System

Chapter 11

Musical Experiences

It has been a very important care in my research to confront the system to the musical reality of improvisation. I tried along these years to test the prototypes in the most various and numerous musical situation as possible and gathered (informally) feedback from the musicians on these experiments. These various work sessions, concerts, *impromptus* and other musical situations are almost exhaustively in the *List of Musical Experiences* at the end of this thesis to which I will make often reference in the current chapter. I will detail here a few notable experiences and explain how these musical situations influenced the work presented in this thesis.

11.1 Duets

The first and simpler improvisation situation I tried with the system is naturally the case of duet between an acoustic musician, improvising in its own style and the supervised computer, learning in real-time and generating new musical discourse based on what it just learnt. Technically and musically this situation appears as the simplest possible, however since the computer only bases its discourse on the material provided by the musician along the current improvisation, it deeply relies on the acoustic musician to be inventive and bring new material to augment the variations possibilities of the computer. The computer proposes new variations and thus also provide a consistent part of novelty and creativity. But the primary musical material only relies on the acoustic musician in this case. For example, a musician starting the improvisation by a looped pattern provides very easily analyzed material considering the knowledge model of the system based on repetition. But we can not expect the computer to escape this loop and suggest novel directions until the musician gives himself different material than the initial loop.

Nevertheless, this simple situation allowed me to really adjust and refine the listening part of the system. One of the first extraordinary musician I met and played with was Michel Doneda, a French saxophonist (soprano). He is a long accomplished musician, mostly improvising in very diverse contexts from totally free improvisations to collaboration with independent rock bands and in various countries. He arrived in our working sessions

(Doneda09) to prepare a collective concert (Martin10) and played with his very own characteristic style constituted almost exclusively of continuous granular and colored “noises” of subtle blowing and fingerings on his soprano saxophone. To capture the patterns in this very unexpected style, I had to adapt very precisely the spectral detection. In particular, I had to make a specific MFCCs distance profile (see section 5.2.1.3) to enable the system to delimit proper clusters. In this same direction, the work with the French percussionist Laurent Mariusse (Mariusse10a) who often constitutes his own set of percussions with a marimba/vibraphone, a *thunder* metal plate but also kitchen bowls, thick stone cutting discs and other unexpected objects, pushed further the research around the spectral listening to achieve coherent recombinations of this kind of heterogenous material.

The pitch detection part of the system had greatly benefited from the few months residency of Steve Lehman in our team. Steve Lehman is a leading saxophonist of the contemporary New York Jazz scene. Composer as well, his pieces have strong written structures and he make use of all the techniques coming from the contemporary music especially from the *spectral music*. He is especially skilled in the use of quarter tones and microtonal intonations. So that during the time we tested the system together (Lehman11), we developed a quarter tones pitch detection extending our initial semi-tone scale. This scale gave very interesting results with the system enabling refined melodic pattern recognitions and a stronger adaptation to the tuning of a specific instrument or instrumentalist. I reused it later with singers for example helping the recognition of vibrato patterns. But the fixed quarter tone definition was not precise enough to adapt to the specific fingerings of Steve’s saxophones which provided finer microtonal tunings. He extracted the pitch detection module of the system and customized the microtonal scale to fit his saxophone and playing. He also integrated this module to his own computer system for improvisation.

A third experiment in duet is here to note: I had the great opportunity to meet and improvise with Steve Coleman. There is no need to present Steve Coleman, he is one of the most influent saxophonist on the Jazz scene since the 80’s. His improvisation and composition systems strongly relies on pitches’ structures and processes (scales, modes; evolutions of those, mutations etc.). After explaining the functioning of the system through the example of pitch description and melodic recombinations to Steve Coleman, he tried it very accurately with two short improvisations (around 3 minutes each) using two non-overlapping modes. Predictably, the system showed two disconnected parts in the graph. Then he combined both modes in a third improvisation which led to many links in the graph with the two previous parts. An idea came from this experiment that we did not succeed to implement yet. I would like to be able to make this kind of musical mode emerge from an improvisation. For example, if a musician used a 7-pitches scale in an improvisation, I would like to enable the system propose variations reducing this scale to 5 of these pitches for example. That would mean strictly excluding patterns using the

2 other pitches and always finding a jump to exclude these patterns. We did not solve the problem of efficiently navigation with such strong constrain yet.

11.2 Collective Free Improvisation

I placed the system often in a second usual improvisation context: collective free improvisation. In this situation, several musicians improvise together without any pre-determined references to a specific style or idiom. The musical issue for the system in such context is to be able take on a pertinent place among the musicians while its material depends from the input of one or several of them. A first important concert in this context took place 2010 at IRCAM ([Martin10](#)) gathering four improvisators from different horizons: Brice Martin, bassoonist coming from a contemporary music background, Beñat Achiary, a basque singer with a very extended vocal technique, Médéric Collignon playing mainly a pocket bugle but also making music with several small objects and experimenting extraordinary vocal techniques as well, and Michel Doneda, saxophonist we already presented in the previous section. We described in chapter 9 the multiple input and multiple output version of the system that I thoroughly tested on this occasion. The audio stream coming from a microphone on each musician constituted the four inputs of the system and on each musicians, I had two descriptions and model learnt in parallel: a pitch description and a spectral description. I had 9 players rendering the paths of 4 improvisation logics maximum. This means that I had a maximum of 4 different variations generated at a time but I had the possibility to read them simultaneously at different speeds with or without transposition. A pre-defined structure had been planned: alternating between duets with an acoustic musician and the computer system and collective *intermezzi*. During the concert which lasted about an hour my role as supervisor of the system was crucial and could be named *computer musician*. I had to handle, control, place in space and build a pertinent discourse with the computer system in this musical “arena”. This meant intensive musical decisions on which musician to listen to, which description to use, which “clone” to play, which material to use etc. This as been a very founding experience for the research presented in this thesis.

A second experience of smaller scale in collective free improvisation drove the research around the different role of streams and inputs explained in section 5.1. I invited for work sessions with the system, two acoustic musicians, a clarinetist, Jean-Brice Godet and an viola player, Cyprien Busolini, who are long used to improvise together ([Godet11](#); [Godet11a](#)). They regularly play in the parisian free improvisation scene and have a style of improvisation together which is characterized by the prominence of the overall musical material over the individual discourses. They seem to create and manipulate one global musical matter, a fusion of both instruments, rather than developing their personal trajectories intersecting, evolving together or splitting apart. The usual way to play with the system in trio (two acoustic musicians and the computer and its supervisor) up to these sessions was to consider each instrument as an input for a separate copy of the

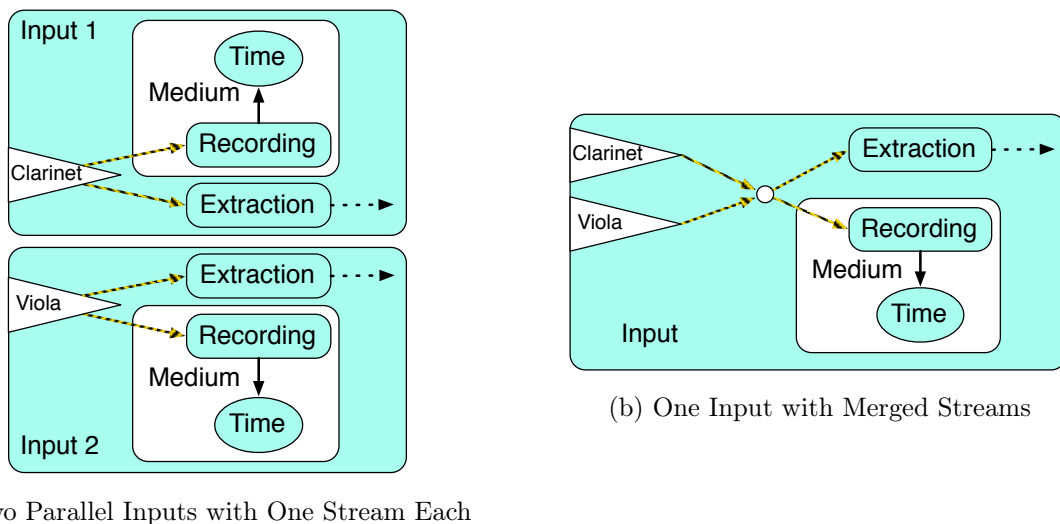


Figure 11.1: Inputs and Streams with Two Instruments

software. When trying to play this way with them, I could not really capture and interact with their musical content by using either one or the other instance of the system because their musical matter was not just a superposition of the two materials but the real fusion of both materials. This fusion was not captured by the system when listening to the musicians separately. No dialog seemed to start in this configuration because the system was not able to apprehend and participate to the global musical matter. So I had then to revise our notion of input and start thinking that I would get better results merging the clarinet and the viola audio streams, taking the mixture as the new global input as presented [Figure 11.1](#). The conceptual separation of streams and inputs took all this sense in this situation. Naturally, this merging had notable consequences on the extraction and analysis because the pitch description becomes almost obsolete with a polyphonic input. However, as the main matter of the improvisation was this fusion of timbre, the spectral analysis done on the merging of both stream instantly opened the musical dialog.

11.3 Jazz Band

Later on, I had the occasion of trying to use our system with a formed Jazz band. After a few duet improvisation with the French saxophonist Raphaël Imbert, he introduced me (and the software) to his music company, the Nine Spirit Company, based in the south east of France. The company is a variable ensemble with strong Jazz roots but Raphaël is very found of mixing several styles of music his concerts and recordings. The place of the system was then rather easy to find, following Raphaël's musical ideas. When playing with this band, there usually was the classic Jazz *rhythmic section* with at least a drum set and a double bass player, sometimes completed with keyboards. One of the first usage of the system in this context was to learn a Jazz chorus (usually from the saxophonist) then take on by itself a new solo later on in the piece. As we did not include any notion

of harmonic grid in the system, we had to assume the floating/disconnected nature of the computer generated solo. But Raphaël included this characteristics in his composing and made suitable place for this kind of abstract superposition.

In my experiences, I had the occasion to prepare the system for the playing with another formed Jazz band: Carine Bonnefoy's quintet. Carine Bonnefoy is a French Jazz composer, arranger, pianist and conductor, working in particular in the Conservatoire National Supérieur de Musique et de Danse de Paris (CNSMDP). In her quintet, she is composing and playing the piano, there are a trumpet player, a saxophone player, a double bass player and a drummer. We recorded studio sessions and rehearsals to prepare a concert with the system and the band at CNSMDP in October 2012. We extended the use of the system in order to generate its own chorus by hybridizing instruments choruses. The idea developed in collaboration with Georges Bloch was to learn successively in the same input of the system two choruses played by the trumpet player and the saxophonist at different moments of the same piece and on the same harmonic grid and tempo. The system was then able to generate a new chorus jumping back and forth from the trumpet material to the saxophone material creating some kind of new hybrid instrument improvising in the piece. The same lack of harmonic grid as mentioned earlier had to be taken into account when planning these pieces and choruses. But in this case of successive learning of two different instruments, both the pitch and spectral descriptions gave pertinent and different musical results. The ability of the system to generate a solo with the combination of ideas and sounds of two instruments added a remarkable interest in its usage in such context, even though it could not follow the harmonic grid.

With the material recorded during the studio sessions with Carine's quintet described earlier, we could also test for the first time the pulse preserving mechanism we developed for the system (explained in sections 5.1.2.4, 7.2.2. and 10.2.2). During these sessions, a multitrack recording of each separated instruments was made, including the drum part. I could thus evacuate the critical problem of real-time pulse extraction and annotate off-line the pulse for all the tracks thanks to the drum track. Then we could effectively test the efficiency of the phase representation (section 5.1.2.4) and phase constraint (sections 7.2.2 and 10.2.2) for our system. It gave very successful results so that we could use it during the concert planned with Carine's quintet and the virtual saxophone and trumpet hybrid could improvise and keep up with its tempo. As we had no real-time extraction of pulse at the time, we could only preserve the pulse while generating and it was up the musicians to synchronize themselves with the computer's pulse.

Lately, I played regularly with Raphaël Imbert's Nine Spirit Company. He invited me several times to play in his concerts and had me come in his musical residencies with different persons from the company. During one of the last residency with him and his musicians ([Imbert13](#)), in a much larger setup, two trombonists were playing in the band. I took advantage of this situation to test a new application of both the merging of streams and the spectral description of the streams. The two trombonist played together a common

duet chorus in one of the piece. I recorded and learnt this double solo in one input of the system as they were playing in the same microphone. The spectral description gave very successful result as the two instruments were sometimes playing together, sometimes alternating in a classical question/answer scheme. The model built on the timbres of both instruments was really meaningful. Then the computer was able to generate a very lively and punchy jazz solo for two trombone.

During the same residency ([Imbert13](#)) and on the occasion of another concert right after ([Imbert13a](#)), I decided to test the real-time extraction and usage of the pulse. As explained in section 5.1.2.4, real-time pulse extraction is a whole research subject on its own. But thanks to the implementations of L. Bonnasse-Gahot and J. Chao of E. W. Large's work on beat tracking with oscillators, I could get a working module to extract more or less a relevant pulse. It has actually been very interesting to compare the results of the pulse extraction with two different drummer playing with the band. I placed piezo contact microphones on both cymbals of the drum set to extract the impacts to drive the beat tracking module. This pulse was used at the same time to annotate the learning of the system and to synchronize the generation. Even though the pulse tracking was not perfect and I had to drive it carefully and check regularly its pertinence, the musical result was very impressive. With this setup we could for the first time really feel that the computer was participating to the common pulsed improvisation without been disconnected or floating above the rest of the band. It constitutes a really promising direction that we have to continue exploring.

11.4 Prepared Improvisations

In his exploration of the system, Raphaël Imbert liked also very much the possibility to prepare the software with sound files on which we can apply the same analysis, modeling and generation principles — exactly as if they were real-time sources. His ethnomusicological research leads him to often incorporate historical material into his compositions. The re-improvisation of such musical material through our system has been really appealing to this idea. Thus, he gave me regularly historical recordings like the first solo recording of Coleman Hawkins (famous Jazz saxophonist from the 20th century), an old african-american preach, a recording of a prisoners' work song which rhythm is given by the sound of pickaxes on stones etc. In most of these cases, the musical idea was to use one or several computer based generation on these materials to constitute a musical framework for an original composition or improvisation of the band. The use of the african-american preach for example leads to a collective idiomatic improvisation close to hip hop style.

Raphaël Imbert pushed me to go further in the use of external recordings with a particular recording of the 20th century saxophonist John Coltrane playing an improvised *coda* of one of his composition: *I Want to Talk About You*. In this case, I prepared the system with this recording of J. Coltrane then a real-time improvisation of Raphaël

on the same theme is learnt on-the-fly as a present *complement* of the original coda. I make the system learn the new solo of Raphaël on the same input as the recording of J. Coltrane. Thus, as with the hybrid chorus with trumpet and saxophone material previously presented, the computer generates in this case a hybrid coda using both the ideas and sound of the original coda and Raphaël’s re-interpretation of the theme. Along these experiences of “historical-hybrid” improvisations, I improved the recognition of common patterns between both materials by blocking the alphabet discovery (along Raphaël’s real-time learning) and forcing the spectral recognition to use the clusters already defined when learning J. Coltrane’s solo. The idea is to avoid the explosion of alphabet size and increase recombination links between the two sources. Doing so, I could have successful melodic and timbral hybridizations. One of the advantages of the spectral description in this setup is to include in the analysis the recording quality of both material. Naturally, the quality of the recording of J. Coltrane’s historical solo is very different from the quality of a real-time recording of Raphaël: the historical recording is noisier, with less precision but there is almost no music in background while Raphaël’s recording has usually a cleaner sound but may be overlaid with other musicians accompanying him or audience’s reactions. The spectral description includes these differences as the whole sound is considered. The common timbral patterns found by the system thus enable to jump from one material to the other with smoother transitions than with the melodic description which only considers the pitch of the notes. I played this “piece” based on J. Coltrane’s coda several times in concerts with Raphaël ([Imbert11a](#); [Imbert11b](#)) and it has always created much impression by its timeless flavor.

11.5 Speech Improvisation

A very different type of improvisation with the system that I personally tested was the playing with vocalists and singers. Very early in my work, I had the occasion to improvise with Beñat Achiary, a Basque singer. His background is deeply linked with the Basque tradition of south-west of France but he now mixes several vocal techniques and has been practicing vocal improvisation for a very long time. He started his first testing of the system ([Achiary09](#)) with the reading of written poetry in French and Basque languages and explored directly the relation with the meaning of words. Obviously, the system has absolutely no notion of words, sentences or meaning. Therefore, there is an inevitable rupture of those elements of the language in the computer based generation. However, the spectral description implemented in the system is very suitable to capture the recurrences of the language’s sound. And the discourse generated by the computer is a subtle recombination and jumbling of various words together which very much emphasizes the sonorities sometimes hidden behind the meaning of texts. This work on texts with Beñat has really been a poetic meeting which opened nice artistic directions.

I followed this work on texts and deconstruction of the meaning of words to emphasize their musicality through my participation to a project of musical theater. Benjamin

Lazar a French comedian and director called me to realize the computer part of one of his modern creation. This project in which he was at same time writer and director gathered a soprano (Claire Lefilliâtre), a composer (Vincent Manac'h), an artistic director (Geoffroy Jourdain) and myself to build together a play which would talk about the thin frontier between language and music in our human experience. The generative system was naturally one piece of software only in the whole setup imagined and realized for the piece. Its use during the theater play was concentrated around a scene in which a mother is cradling her (imagined) child in her arms. She is telling an ancient tale to lull the child and the system is recording this story. Along the tale, the voice of the mother is duplicated thanks to the system and through spatialization, the multiple voices are traveling around the audience. The voices coming from the computer generation deconstructed the meaning of the words and the story thanks to the system. The “goal” of this process was to immerse the audience in the particular state of a child letting him or her go to the sleep with the melody of his or her mother’s voice. There exist a moment in this falling asleep when the words do not make sense anymore and only the sonorities and melody of the language accompany the mind to the sleep. I controlled the system during this scene to accompany the audience to this feeling and we had very positive feedback on this particular moment of the play.

Following these fruitful experiences on speech I then integrated to the system, the specific analysis of the prosody done in N. Obin’s work (see section 6.1.3.1). N. Obin’s process enables to segment speech material into syllables and label them with their prosodic contour. The result of this process can be learnt in the OMax system instead of the regular pitch analysis. This way the system is capable of generating new speech material with respect to the prosody of the voice, that is the natural melodic flow of the language. This speech generation gives very convincing results. It produces a fluent stream of voice which sounds like real sentences but which dismantles subtly the words and meaning of the language. The result is very close to glossolalia. I had the occasion to explore this direction first on the voice of the French actor André Dussolier reading the book *À la recherche du temps perdu*, a famous French novel from Marcel Proust. The result is a truly disturbing flow of the quiet and grave voice of André Dussolier telling convincing gibberish with a coherent French prosody. Then, I realized a first stable prototype of this prosodic version of the system that G. Bloch used in a public performance for an exhibition of the *Centre Georges Pompidou* about imaginary languages. This prosodic version combined with his video extension of OMax was used to explore the famous chess match scene of the movie *Le septième sceau* from I. Bergman. G. Bloch replaced the original dialog of the scene with its translation in the imaginary language of the artist Jaap Blonk. Then he let the system recombine both the sound and image of the scene using in parallel the prosodic and spectral knowledge models of the dialog. I pursue this work with N. Obin’s speech processing with the recording of several French and American rapper like Roce, Dr. Dre and Snoop Dog. Combining these experiences with the new pulse extraction of the system, I could give plausible reinterpretation of famous titles like

Still Dre. The work with the French rapper Roce on the same principles has been filmed in a very short movie produced for la *Cité des sciences et de l'industrie*, a famous Parisian science museum which opens December 2013 an exhibition about the human voice.

11.6 PedagOMax

Finally, I had the occasion to invent a specific version of the system for another very different context. The parisian contemporary music ensemble *Court-Circuit* asked me to play a part in a pedagogical project which aims to introduce contemporary music in a *Collège*¹ in Paris. I developed a version of the system which can be controlled through 8 joysticks so that we organized regular workshop with up to 8 children, one musician-improvisator of the ensemble and my-self. In this setup, the system listens and learns from the acoustic musician improvising. Then, four computer “clones” are controlled collaboratively by pairs of joysticks. For each “clone” one joystick conducts its behavior with 3 axes and one switch: the switch starts and stops the generation, the throttle of the joystick controls the volume — that is the musical dynamic —, the X axis controls the time-stretching and the Y axis controls the transposition. The second joystick attached to a “clone” is used to choose the musical content of the generation through the manipulation of a region. Two axes and one switch of this second joystick are used: the switch activates or deactivates the effect of the region — when the region is *on*, the “clone” is force to recombine musical material contained in this region, when the region is *off*, the “clone” is free to recombine any part of the original material —, the X axis is used to move the region along the timeline to select material towards the past (left) or the present time of the learning (right), the Y axis enables to widen (up) or narrow (down) the length of the region — so that the “cloned” is constrained in a large portion of the material or a very small and focused area. This way, each of the four “clones” is *played* by two children. This workshop has been a real success. Children understood very quickly, with the help of the visualization, how the system works and we could build all together prepared and free improvisations where everybody was careful to common musical discourse.

¹which is the common school for children from 11 to 14 years old, equivalent to the american *Middle School*

Chapter 12

Conclusion & Future Works

12.1 Conclusion

We started this thesis with a review of several systems for music generation. We took voluntarily various generative systems including some oriented towards music compositions and presented three high level criteria enabling us to organize this review on the operational level, that is on *how* those systems work. In the second chapter of this review, we embraced another point of view and described how a system oriented towards improvisation should behave on the musical level, that is what these systems musically produce. The second part of this thesis thoroughly presented the concepts and principles for a generative improvisation system based on stylistic reinjection. The organization of this part was inspired by cognitive reflexions and adopted an anthropomorphic structure coming from the consideration of the global entity formed by the system and its supervisor. This entity should behave as any acoustic musician with his instrument. We detailed the listening, learning and generative capacities of the system and the conceptualizations of those. In the third part of this work, we proposed new structures and architectures to achieve such a performance computer system. We exemplified this software design research with successive architecture renewals and prototypes, and described several implementation issues we encountered along the building of two main versions of the system. Finally in the last chapter of this thesis, we told numerous musical situations in which we played with the system along with leading acoustic musicians. These wide-ranging performances from studio work sessions to large public concerts have considerably helped to challenge and enrich the work presented in this thesis. They finally help us to measure its drawbacks and project the future work remaining to be done around this system.

At the end of this thesis, we come off with a working and efficient system for interactive and agnostic music improvisation systems. We made it versatile and controllable to enable the playing in many musical contexts. To adapt to those contexts we included various and efficient analysis of the music and gave the supervisor powerful means to develop pertinent musical discourses. But this system has naturally some limitations. Some of these limitations come from choices we made. For example our *on the fly* hypothesis

made us avoid the usage of abundant external material, a choice which may be criticized when huge databases of sounds and pieces are available. Our core paradigm and the last architecture presented may actually not be incompatible with such databases but certainly require specific adaptations. Some other limitations coming from this last architecture may be overcome in further explorations for example the empowering of the system with more automatic decisions would relieve the supervisor from low-level choices and give him or her a higher view on the musical possibilities. To do so, we could benefit from higher level analysis of the on-going music, for example the extraction of the acoustic musician's improvisation's global structure would be of great interest. The knowledge model that we use is essentially sequential and non-hierarchical. Even though we proposed a clustering algorithm to extract some organization of the patterns, it may not give very fruitful result to analyze higher order organization such as harmonic grid, orchestration differences etc. The same kind of higher organization may come from the preparation of an improvisation. Our way to play with the system in such context is either to externally script the parameters to use or for the supervisor to perform in real-time the improvisation's plan. Thus we do not benefit, in the internal system's working, from the prepared information as a structured scenario and a source of knowledge. Finally, we systematically use the sound capture of a musician as the system's material for generation. The *deforming mirror* created this way is intriguing and appealing to musicians and the manipulation of such recorded material gives an intrinsic consistence in the generation's details as e.g. the articulation. But inspired from the existing numerous sound synthesis we could try exploring possible realization of our stylistic reinjection with other sound material. This idea has been successfully broached in the Native Alien project which make use of our system with granular synthesis instead of our regular concatenative players.

12.2 Perspectives

Our thorough study of the principles and architectures for an improvisation system lead us to explore several conceptions and designs for such a system. Judging by the numerous musical experiences with the system and the interest shown by several world-class musicians, we succeeded in the building of a musically pertinent system. Yet this work calls for many more improvements and opens up new research direction to embrace more and more aspect of musical improvisation.

The perception part of the system could benefit from new descriptions alongside with melody, timbre and harmony, and enhance this way its capacities to capture several aspects of the musical discourse. In particular, neither the pulse extraction, nor the tests we have done on notes' durations have lead to an efficient usage of the Factor Oracle graph for the rhythmic modeling. Although our usage of pulse information represents a major breakthrough for the OMax scheme from now on capable of taking part in a strongly pulsed context, the precise modeling of rhythmical patterns would enable to generate variations of these patterns and create thus an interesting musical dialog on this

important musical aspect as well. We did not study another very important process of the perception of improvisation: self-listening. Indeed, when improvising, acoustic musicians not only listen to the others' discourses and to the global result but also to their own sound and the adequacy of it in the overall music. The system we built do not reproduce this mechanism. One of the direction to do so would be to consider the output of the system as a new input stream. However this *external* looping would not benefit from the intimate knowledge of the systems' generation. Therefore, there ought to be a more efficient way to give the system musical self-awareness.

On the knowledge part, Factor Oracle graph has proven to be a very powerful core for our knowledge model. However, we surely have now to integrate new models in the memory of our system which properties would complement those of Factor Oracle. For example, the search for paths satisfying arbitrary constraints in the Factor Oracle does not appear as a simple issue. Several other structures have been invented to enable the efficient search for such paths. The new architecture we proposed in this thesis allows a straight forward implementation of new models in the knowledge database. Those models may enable the artificial emergence of arbitrary properties in the generation part even though the original material did not explicitly show this property. For example, the possibility to improvise on a particular subset of notes (mode) from the original material has been of strong appeal when trying the system in the idiomatic Jazz context. On a shorter term, the automatic selection of the proper description to use for the generation may be an interesting research direction. Indeed, up to now, it is still up to the supervisor to choose the pertinent description(s) on which musical variations are based. Our architecture gave the supervisor the mean to do so along the playing, without interrupting the musical discourse of the computer. However, we could imagine a *smarter* system, capable of detecting automatically from the musical context the correct description(s) to activate. This direction would pursue the idea to achieve a system very responsive to its context — all the way to the adaptation of its listening — and the evaluation of the adequacy of its own musical discourse to the context with self-listening.

Finally, we wish that the integration of our system into musical projects will continue and expand. We believe that many more musical explorations are yet to try. We hope that long contributing musicians as well as musicians discovering the system for the first time will keep on suggesting ideas and bendings of our system.

List of Musical Experiences

Concerts

Martin et al.: Concert “L’improvisation” at IRCAM

Martin10

Paris, Feb. 11, 2010.

Concert in l’Espace Projection at IRCAM, Paris in the framework of the Colloquium *Analyze Improvisation*. Structured improvisation around OMax, 1 hour

Brice Martin, *Bassoon*

Beñat Achiary, *Voice*

Médéric Collignon, *Trumpets and Voice*

Michel Donéda, *Saxophones*

Benjamin Lévy, *OMax*.

Mariusse et al.: Improvisations avec OMax

Mariusse10

Rennes, France, May 18, 2010.

Concert for the *Journées d’Informatique Musicale* 2010 in Rennes, around 35 minutes of free improvisation

Laurent Mariusse, *Percussions*

François Nicolas, *Piano*

Nicolas Misdariis, *Saxophone and Computer*

Gérard Assayag, *OMax*

Benjamin Lévy, *OMax*.

Godet et al.: Music and Dance Improvisation

Godet10

Paris, Oct. 2, 2010.

Concert for the opening of a sculpture exhibition in Paris. Around 45 minutes of free Music and Dance improvisation inspired by the sculptures and paintings of the exhibition

Jean-Brice Godet, *Clarinets*

Jose Luis Sultan, *Dance*

Colette Grandgérard, *Sculpture & Painting*

Benjamin Lévy, *OMax*.

Imbert et al.: Concert in Cité de la Musique of Marseille**Imbert11a**

Marseille, France, Jan. 20, 2011.

Raphaël Imbert, *Saxophones*

Pierre Fénichel, *Double Bass*

Simon Sieger, *Piano, Keyboards and Trombone*

Thomas Weirich, *Guitars*

Jean-Luc Di Fraya, *Drums and Voice*

Gérard Assayag, *OMax*

Benjamin Lévy, *OMax*.

Di Donato: Des puces, des souris et des hommes**Di-Donato11**

Lyon, France, Mar. 15, 2011. URL: http://www.dailymotion.com/video/xw5n5y_jacques-di-donato-plays-omax-in-lyon-2011_music.

Performance in Lyon presented in a cycle of scientific conferences.

Jacques Di Donato, *Clarinet*

Gérard Assayag, *OMax*

Benjamin Lévy, *OMax*

Jacques Saramut, *Speaker*.

Lubat: Concert with the Lubat Company in New Morning, Paris**Lubat11**

Paris, May 9, 2011.

Bernard Lubat, *Keyboard, Conducting* and his musicians

Gérard Assayag, *OMax*

Benjamin Lévy, *OMax*.

Lubat et al.: Musique du XXIème siècle, L'homme, l'outil et la machine**Lubat11a**

Noaillan, France, Aug. 14, 2011.

Concert in the church of Noaillan, South-West of France, opening concert of the 34th edition of the Hestejada dé las arts Music festival

Bernard Lubat, *Keyboards and Voice*

Fabrice Vieira, *Guitar and Voice*

Marc Chemillier, *OMax*

Benjamin Lévy, *OMax*.

Lubat et al.: Futuroscopie, Musique contemporaine d'Occitanie**Lubat11b**

Pompejac, France, Aug. 18, 2011.

Concert in the church of Pompéjac, South-West of France during the 34th edition of the Hestejada dé las arts Music festival

Bernard Lubat, *Keyboards and Voice*

Fabrice Vieira, *Guitar and Voice*

Beñat Achiary, *Voice*

Marc Chemillier, *OMax*

Benjamin Lévy, *OMax*.

Lubat: Concert at Espace Paul Jargot, Crolles

Lubat11d

Crolles, France, Sept. 30, 2011.

Bernard Lubat, *Piano*

Assayag Gérard, *OMax*

Benjamin Lévy, *OMax*.

Martin: Concert in Studio de l'Ermitage, Paris

Martin11a

Paris, Nov. 9, 2011.

Concert of the *Sons Neufs* Jazz Festival

Brice Martin, *Bassoon*

Benjamin Lévy, *Computer*.

Imbert et al.: OMax @ Lomax

Imbert11b

Aix-en-Provence, France, Nov. 15, 2011. URL: http://www.dailymotion.com/playlist/x2d46f_RepMus_omax-_at-_lomax/1#video=xvpyag.

Concert with the Nine Spirit Company at Seconde Nature in Aix en Provence

Raphaël Imbert, *Saxophones*

Pierre Fénichel, *Double Bass*

Simon Sieger, *Piano, Keyboards and Trombone*

Thomas Weirich, *Guitars*

Jean-Luc Di Fraya, *Drums and Voice*

Gérard Assayag, *OMax*

Benjamin Lévy, *OMax*.

Godet et al.: Concert at Naxos Bobine, Paris

Godet11b

Paris, Dec. 7, 2011.

Free improvisation, around 45 minutes

Jean-Brice Godet, *Clarinets*

Cyprien Busolini, *Viola*

Philippe Cornus, *Percussions*

Benjamin Lévy, *OMax*.

Imbert et al.: Concerts at Roulette, Brooklyn and Columbia University Imbert12

New York, USA, May 16–18, 2012. URL: <http://repmus.ircam.fr/improtechpny>.

Concerts of the Workshop *ImproTech Paris-New York 2012 : Improvisation & Technology*

Raphaël Imbert, *Saxophones*

Simon Sieger, *Piano, Keyboards and Trombone*

Thomas Weirich, *Guitars*

Benjamin Lévy, *OMax*.

Martin: Concert in Concordia University, Montreal

Martin12

Montreal, Canada, May 24, 2012. URL: <http://matralab.hexagram.ca/comprovisations2012/about/> and http://www.dailymotion.com/video/xvbn7l_brice-_martin-_and-_benjamin-_levy-_omax-_performance-_at-_comprovisation-_2012-_workshop_creation.

Concert at Matralab for the Comprovisations, Improvisation Technologies for the Performing Arts Research-Creation Workshop

Brice Martin, *Bassoon*

Benjamin Lévy, *OMax*.

Imbert: Concert with the Nine Spirit Company in Marseille

Imbert12a

Marseille, France, July 17, 2012.

Opening concert of the jazz festival "des cinq continents"

Raphaël Imbert, *Saxophones*

Sarah Quintana, *Voice and Guitar*

Paul Elwood, *Banjo and Voice*

Pierre Fénichel, *Double Bass*

Simon Sieger, *Piano, Keyboards and Trombone*

Thomas Weirich, *Guitars*

Jean-Luc Di Fraya, *Drums and Voice*

Benjamin Lévy, *OMax*.

Imbert: Concert with the Nine Spirit Compagny in Junas

Imbert12b

Junas, France, July 18, 2012.

Concert of the jazz festival in Junas

Raphaël Imbert, *Saxophones*

Sarah Quintana, *Voice and Guitar*

Paul Elwood, *Banjo and Voice*

Pierre Fénichel, *Double Bass*

Simon Sieger, *Piano, Keyboards and Trombone*

Thomas Weirich, *Guitars*

Jean-Luc Di Fraya, *Drums and Voice*
Benjamin Lévy, *OMax*.

Roux: Miroir

Roux12

Uzeste, France, Aug. 23–24, 2012.

Concert in Uzeste, South-West of France during the 35th edition of the Hestejada dé las arts Music festival

Sylvain Roux, *Flutes*

Benjamin Lévy, *OMax*.

Martin: Il est libre OMax

Martin12a

Uzeste, France, Aug. 25, 2012.

Concert in Uzeste, South-West of France during the 35th edition of the Hestejada dé las arts Music festival

Brice Martin, *Bassoon*

Benjamin Lévy, *OMax*.

Godet: CHI 2013 — IRCAM Lab Tour

Godet13

Paris, Apr. 30, 2013.

Concert at IRCAM for the Lab Tour of Computer and Human Interactions Conference, two free improvisations (about 10 minutes each)

Jean-Brice Godet, *Clarinets*

Benjamin Lévy, *OMax*.

Imbert et al.: Residency & Concert with the Nine Spirit Company

Imbert13

Baux de Provence, France, May 25–29, 2013.

Concert and Residency in the framework of Marseille-Provence 2013, European Capital of Culture

Raphaël Imbert, *Saxophones*

Pierre Fénichel, *Double Bass*

Simon Sieger, *Piano, Keyboards and Trombone*

Romain Morello, *Trombone*

Christophe Leloil, *Trumpet*

Thomas Weirich, *Guitars*

Alain Soler, *Guitars*

Cedric Bec, *Drums and Voice*

Benjamin Lévy, *OMax*.

Imbert et al.: Rates of Spirit

Imbert13a

Arles, France, June 2, 2013.

Concert with the Nine Spirit Company in the framework of Marseille-Provence 2013, European Capital of Culture

Raphaël Imbert, *Saxophones*

Pierre Fénichel, *Double Bass*

Simon Sieger, *Piano, Keyboards and Trombone*

Thomas Weirich, *Guitars*

André Rossi, *Piano*

Jean-Luc Di Fraya, *Drums and Voice*

Benjamin Lévy, *OMax*.

Lambla et al.: Soustraction addictive**Lambla13**

Uzeste, France, Aug. 18, 2013–Aug. 19, 2012.

OMax based composition during a Concert-Conference of the 36th edition of the Hestejada dé las arts Music festival in Uzeste, South-West of France

Pierre Lambla, *Saxophone*

Jaime Chao, *Voice, Guitar, Computer*

Benjamin Lévy, *OMax*.

Roux: Miroir**Roux13**

Uzeste, France, Aug. 19, 2013.

Concert in Uzeste, South-West of France during the 36th edition of the Hestejada dé las arts Music festival

Sylvain Roux, *Flutes*

Benjamin Lévy, *OMax*.

Imbert et al.: L'âme du temps**Imbert13b**

Tournai, Belgium, Aug. 31, 2013.

Concert of the Music & Philosophy Festival *Les [rencontres] inattendues*

Bernard Stiegler, *Philosopher*

Raphaël Imbert, *Saxophones*

Karol Beffa, *Piano*

Johan Farjot, *Rhodes*

André Rossi, *Organ*

Marion Rampal, *Voice*

Arnaud Thorette, *Viola*

Pierre Fénichel, *Double Bass*

Benjamin Lévy, *OMax*.

Demonstrations

Mariusse: Prototype Evening

Mariusse10a

Paris, June 19, 2010.

Live demonstration of OMax with a musician in IRCAM's Studio 5, 1 hour

Laurent Mariusse, *Percussions*

Gérard Assayag, *Speaker*

Benjamin Lévy, *OMax*.

Lê Quang: Concert-Presentation of OMax in the MaMux Conférence at IRCAM

Le-Quang11

Paris, May 20, 2011.

Vincent Lê Quang, *Saxophone*

Benjamin Lévy, *OMax*.

Mezzardi: Performance-Conference in Le Blanc-Mesnil, Paris Suburbs

Mezzardi12

Le Blanc Mesnil, France, May 5, 2012.

“Magic” Malik Mezzardi, *Flute*

Benjamin Lévy, *OMax*.

Kalinowski: Around Acoustics and Synthetic Sound

Kalinowski12

Krakow, Oct. 12, 2012.

Live demonstration of OMax with a musician for the Music Academy of Krakow

Jan Kalinowski, *Cello*

Benjamin Lévy, *OMax*.

Imbert et al.: Le logiciel musicien OMax

Imbert13c

Tournai, Belgium, Sept. 1, 2013.

Demonstration-Concert for the Music & Philosophy Festival *Les [rencontres] inattendues*

Raphaël Imbert, *Saxophones*

Karol Beffa, *Piano*

Benjamin Lévy, *OMax*.

Sound Installation

Robin: Sound Installation in the Garden of Villa Medici, Rome

Robin10

Rome, Italy, June 2–8, 2010.

Standalone improvisation with OMax in an short-lived building in the gardens of Villa Medici

Yann Robin, *Composition*

Benjamin Lévy, *Realisation*.

Workshops

CNSMDP: OMax Workshop at CNSMDP

CNSMDP10

Paris, Oct. 9, 2010.

Gérard Assayag, Georges Bloch and Benjamin Lévy performing OMax with the students of the Jazz department of the National Conservatoire of Paris.

CNSMDP: OMax Workshop at CNSMDP

CNSMDP11

Paris, Mar. 31, 2011.

Benjamin Lévy performing OMax with the students of the generative improvisation department of the National Conservatoire of Paris.

Chao: Informatique instrumentique

Chao11

Uzeste, France, Aug. 15–20, 2011.

OMax workshop in public during the the 34th edition of the Hestejada dé las arts Music festival in Uzeste, South-West of France

Benjamin Lévy, *OMax* and invited musicians.

Lubat: Workshop at Espace Paul Jargot, Crolles

Lubat11e

Crolles, France, Oct. 1, 2011.

Bernard Lubat, *Piano* and invited musicians

Benjamin Lévy, *OMax*.

Chao: Informatique instrumentique

Chao12

Uzeste, France, Aug. 19–21, 2012.

OMax workshop in public during the the 35th edition of the Hestejada dé las arts Music festival in Uzeste, South-West of France

Benjamin Lévy, *OMax* and invited musicians.

Uzeste, France, Aug. 22, 2013–Aug. 24, 2012.

OMax workshop in public during the the 36th edition of the Hestejada dé las arts Music festival in Uzeste, South-West of France

Benjamin Lévy, *OMax* and invited musicians.

Work Sessions

Achiary: Work session at IRCAM to prepare concert [Martin10](#)

Achiary09

Paris, Nov. 19, 2009. URL: <http://recherche.ircam.fr/equipes/repmus/WoMax/sessions/>.

Beñat Achiary, *Voice*

Benjamin Lévy, *OMax*.

Martin et al.: Work session at IRCAM to prepare concert [Martin10](#)

Martin09

Paris, Nov. 2, 2009. URL: <http://recherche.ircam.fr/equipes/repmus/WoMax/sessions/>.

Brice Martin, *Bassoon*

Médéric Collignon, *Bugle, Voice & other accessories*

Benjamin Lévy, *OMax*.

Doneda: Work session at IRCAM to prepare concert [Martin10](#)

Doneda09

Paris, Nov. 4, 2009. URL: <http://recherche.ircam.fr/equipes/repmus/WoMax/sessions/>.

Michel Doneda, *Saxophones*

Benjamin Lévy, *OMax*.

Collignon: Work session at IRCAM to prepare concert [Martin10](#)

Collignon09

Paris, Dec. 11, 2009. URL: <http://recherche.ircam.fr/equipes/repmus/WoMax/sessions/>.

Médéric Collignon, *Pocket Bugle and Voice*

Benjamin Lévy, *OMax*.

Martin: Work session at IRCAM to prepare concert [Martin10](#)

Martin09a

Paris, Dec. 14, 2009. URL: <http://recherche.ircam.fr/equipes/repmus/WoMax/sessions/>.

Brice Martin, *Bassoon*
Benjamin Lévy, *OMax*.

Collignon et al.: Work session at IRCAM to prepare concert Martin10 Collignon10

Paris, Jan. 7, 2010. URL: <http://recherche.ircam.fr/equipes/repmus/WoMax/sessions/>.

Médéric Collignon, *Bugle, Pocket Bugle and Voice*
Beñat Achiary, *Voice*
Benjamin Lévy, *OMax*.

Kimura: Work sessions at IRCAM Kimura10

Paris, July 8, 2010.

Work sessions at IRCAM Mari Kimura, *Violin*
Benjamin Lévy, *OMax*.

Kimura: Work sessions at IRCAM Kimura10a

Paris, Aug. 18, 2010.

Work sessions at IRCAM Mari Kimura, *Violin*
Benjamin Lévy, *OMax*.

Imbert: Work sessions at IRCAM Imbert10

Paris, Dec. 3, 2010.

Raphaël Imbert, *Saxophones*
Benjamin Lévy, *OMax*.

Imbert: Work sessions at IRCAM Imbert11

Paris, Jan. 11, 2011.

Raphaël Imbert, *Saxophones*
Benjamin Lévy, *OMax*.

Godet et al.: Work session at IRCAM to prepare the concert Godet11b Godet11a

Paris, May 13, 2011.

Jean-Brice Godet, *Clarinets*
Cyprien Busolini, *Viola*
Benjamin Lévy, *OMax*.

Lehman: Work sessions at IRCAM**Lehman11**

Paris, June 13–22, 2011.

Steve Lehman, *Saxophone*

Benjamin Lévy, *OMax*.

Godet et al.: Work session at IRCAM to prepare the concert Godet11b **Godet11**

Paris, Sept. 22–23, 2011.

Jean-Brice Godet, *Clarinets*

Cyprien Busolini, *Viola*

Philippe Cornus, *Percussions*

Benjamin Lévy, *OMax*.

Lubat: Work session at IRCAM to prepare the concert Lubat11d **Lubat11c**

Paris, Sept. 26–27, 2011.

Bernard Lubat, *Piano*

Assayag Gérard, *OMax*

Benjamin Lévy, *OMax*.

Coleman: Work session at IRCAM**Coleman11**

Paris, Oct. 26, 2011.

Martin: Work session at IRCAM to prepare concert Martin11a **Martin11**

Paris, Nov. 3–8, 2011.

Brice Martin, *Bassoon*

Benjamin Lévy, *Computer*.

Lê Quang: Work session at IRCAM**Le-Quang12**

Paris, Mar. 10, 2012.

Vincent Lê Quang, *Saxophone*

Benjamin Lévy, *OMax*.

Blesing et al.: Work session at IRCAM**Blesing12**

Paris, Mar. 30, 2012.

Alain Blesing, *Guitar*

Karim Haddad, *Composition*

Benjamin Lévy, *OMax*.

Paris, June 1, 2012.

Alain Blesing, *Guitar*

Karim Haddad, *Composition*

Benjamin Lévy, *OMax*.

Paris, July 13, 2012.

Denis Beuret, *Trombone*

Benjamin Lévy, *OMax*.

Bibliography

- [Allauzen 99a] Cyril Allauzen, Maxime Crochemore & Mathieu Raffinot. *Factor oracle: a new structure for pattern matching*. In SOFSEM'99, volume 1725 of *LNCS*, pages 291–306, Milovy, Czech Republic, November 1999. Springer-Verlag.
- [Allauzen 99b] Cyril Allauzen, Maxime Crochemore & Mathieu Raffinot. *Factor oracle, suffix oracle*. Rapport technique, 1999.
- [Ariza 09] Christopher Ariza. *The interrogator as critic: The turing test and the evaluation of generative music systems*. Computer Music Journal, vol. 33, no. 2, pages 48–70, 2009.
- [Assayag 99] Gérard Assayag, Shlomo Dubnov & Olivier Delerue. *Guessing the Composer's Mind : Applying Universal Prediction to Musical Style*. In ICMC: International Computer Music Conference, Beijing, China, Octobre 1999.
- [Assayag 04] Gérard Assayag & Shlomo Dubnov. *Using Factor Oracles for machine Improvisation*. Soft Computing, vol. 8-9, pages 604–610, 2004.
- [Assayag 05] Gérard Assayag & Shlomo Dubnov. *Improvisation Planning and Jam Session Design using concepts of Sequence Variation and Flow Experience*. In Sound and Music Computing 2005, Salerno, Italie, Novembre 2005.
- [Assayag 06a] Gérard Assayag, Georges Bloch & Marc Chemillier. *Improvisation et réinjection stylistiques*. In Le feed-back dans la création musicale contemporaine - Rencontres musicales pluri-disciplinaires, Lyon, France, Mars 2006.
- [Assayag 06b] Gérard Assayag, Georges Bloch & Marc Chemillier. *OMax-Ofon*. In Sound and Music Computing (SMC) 2006, Marseille, France, May 2006.
- [Assayag 06c] Gérard Assayag, Georges Bloch, Marc Chemillier, Arshia Cont & Shlomo Dubnov. *Omax Brothers : a Dynamic Topology of Agents*

- for Improvization Learning*. In Workshop on Audio and Music Computing for Multimedia, ACM Multimedia 2006, Santa Barbara, USA, Octobre 2006.
- [Assayag 07] Gérard Assayag & Georges Bloch. *Navigating the Oracle: a Heuristic Approach*. In International Computer Music Conference '07, pages 405–412, Copenhagen, Denmark, Août 2007.
- [Assayag 09] Gérard Assayag. *Habilitation à diriger les recherches - Algorithmes, langages, modèles pour la recherche musicale : de la composition à l'interaction*. Thèse de doctorat, Université Bordeaux I, 2009.
- [Assayag 10] Gérard Assayag, Georges Bloch, Arshia Cont & Shlomo Dubnov. *Interaction with Machine Improvisation*. In Shlomo Dubnov Kevin Burns Shlomo Argamon, editeur, The Structure of Style, pages 219–246. Springer Verlag, 2010.
- [Azran 06] Arik Azran & Zoubin Ghahramani. *Spectral Methods for Automatic Multiscale Data Clustering*. In CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 190–197, Washington, DC, USA, 2006. IEEE Computer Society.
- [Bachir-Loopuyt 10] Talia Bachir-Loopuyt, Clément Canonne, Pierre Saint-Germier & Barbara Turquier. *Improvisation : usages et transferts d'une catégorie*. Tracés, no. 18, pages 5–20, 1 2010.
- [Bay 09] Mert Bay, Andreas F Ehmann & J Stephen Downie. *Evaluation of Multiple-F0 Estimation and Tracking Systems*. In ISMIR, pages 315–320, 2009.
- [Biles 94] John A. Biles. *GenJam: A Genetic Algorithm for Generating Jazz Solos*. In International Computer Music Conference, 1994.
- [Biles 13] John A. Biles. *Performing with Technology: Lessons Learned from the GenJam Project*. In Massachusetts Northeastern University Boston, editeur, 2nd International Workshop on Musical Metacreation (MUME). AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2013.
- [Blackwell 02] T.M. Blackwell & P. Bentley. *Improvised music with swarms*. In Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on, volume 2, pages 1462–1467, 2002.
- [Blackwell 12] Tim Blackwell, Oliver Bown & Michael Young. *Live Algorithms: Towards Autonomous Computer Improvisers*. In Jon McCormack &

- Mark d’Inverno, éditeurs, Computers and Creativity, pages 147–174. Springer Berlin Heidelberg, 2012.
- [Bloch 08] Georges Bloch, Shlomo Dubnov & Gérard Assayag. *Introducing Video Features and Spectral Descriptors in The OMax Improvisation System*. In International Computer Music Conference, 2008.
- [Bogaards 04] Niels Bogaards, Axel Roebel & Xavier Rodet. *Sound Analysis and Processing with AudioSculpt 2*. In International Computer Music Conference (ICMC), Miami, USA, Novembre 2004.
- [Bonnasse-Gahot 10] Laurent Bonnasse-Gahot. *Donner à OMax le sens du rythme : vers un improvisation plus riche avec la machine*. Rapport technique, École des Hautes Études en Sciences Sociales, 2010.
- [Borgo 05] David Borgo. Sync or swarm: Improvising music in a complex age. Continuum International Publishing Group, 2005.
- [Bown 09] Oliver Bown, Alice Eldridge & Jon McCormack. *Understanding Interaction in Contemporary Digital Music: from instruments to behavioural objects*. Organised Sound, vol. 14, no. 02, pages 188–196, 2009.
- [Brown 01] J C Brown, O Houix & S McAdams. *Feature dependence in the automatic identification of musical woodwind instruments*. J Acoust Soc Am, vol. 109, no. 3, pages 1064–72, Mar 2001.
- [Brown 09] Andrew R. Brown & Andrew Sorensen. *Interacting with Generative Music through Live Coding*. Contemporary Music Review, vol. 28, no. 1, pages 17–29, 2009.
- [Canonne 11] Clément Canonne & Nicolas Garnier. *A Model for Collective Free Improvisation*. In Carlos Agon, Amiot Emmanuel, Moreno Andreatta, Gérard Assayag, Jean Bresson & John Manderau, éditeurs, Mathematics and Computation in Music, pages 29–41. Springer, 2011.
- [Carey 12] Benjamin Carey. *Designing for Cumulative Interactivity: The _-derivations System*. In NIME, Ann Arbor, Michigan, USA, 2012.
- [Chadabe 84] Joel Chadabe. *Interactive Composing: An Overview*. Computer Music Journal, vol. 8, no. 1, pages pp. 22–27, 1984.
- [Chao 11] Jaime Chao. *Portage de systèmes de détection de tempo*. Internship Report, Septembre 2011.

- [Cleophas 03] Loek Cleophas, Gerard Zwaan & Bruce W. Watson. *Constructing Factor Oracles*. In In Proceedings of the 3rd Prague Stringology Conference, 2003.
- [Collins 11] Nick Collins. *LL: Listening and Learning in an Interactive Improvisation System*. 2011.
- [Collins 12] Nick Collins. *Automatic composition of electroacoustic art music utilizing machine listening*. Computer Music Journal, vol. 36, no. 3, pages 8–23, 2012.
- [Conklin 01] Darrell Conklin & Christina Anagnostopoulou. *Representation and Discovery of Multiple Viewpoint Patterns*. In International Computer Music Association, pages 479–485, 2001.
- [Conklin 03] Darrell Conklin. *Music Generation from Statistical Models*. In Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences, pages 30–35, 2003.
- [Cont 07] Arshia Cont, Shlomo Dubnov & Gérard Assayag. *GUIDAGE: A Fast Audio Query Guided Assemblage*. In International Computer Music Conference, Copenhagen, Denmark, Août 2007.
- [Cont 08] Arshia Cont. *Modeling Musical Anticipation*. PhD thesis, USCD and UPMC, Octobre 2008.
- [Cont 11] Arshia Cont, Shlomo Dubnov & Gérard Assayag. *On the Information Geometry of Audio Streams with Applications to Similarity Computing*. IEEE Transactions on Audio, Speech, and Language Processing, vol. 19-1, 2011.
- [Cope 05] David Cope. Computer models of musical creativity. MIT Press Cambridge, 2005.
- [Cycling74] Cycling74. <http://cycling74.com/products/max/>.
- [Dannenberg 93] Roger B Dannenberg. *Software Support for Interactive Multimedia Performance*. Journal of New Music Research, vol. 22, no. 3, pages 213–228, 1993.
- [De Cheveigné 02] Alain De Cheveigné & Hideki Kawahara. *YIN, a fundamental frequency estimator for speech and music*. JASA: Journal of the Acoustical Society of America, vol. 111, pages 1917–1930, 2002.
- [Dessein 10] Arnaud Dessein, Arshia Cont, Guillaume Lemaitre et al. *Real-time polyphonic music transcription with non-negative matrix factorization and beta-divergence*. In ISMIR-11th International Society for Music Information Retrieval Conference, pages 489–494, 2010.

- [Dubnov 98] Shlomo Dubnov. *Universal Classification Applied to Musical Sequences*. In ICMC: International Computer Music Conference, Ann Arbor Michigan, USA, Octobre 1998.
- [Dubnov 03] Shlomo Dubnov, Gérard Assayag, Olivier Lartillot & Gill Bejerano. *Using Machine-Learning Methods for Musical Style Modeling*. IEEE Computer, vol. 10-38, pages 73–80, 2003.
- [Dubnov 07] Shlomo Dubnov, Arshia Cont & Gérard Assayag. *Audio Oracle: A New Algorithm for Fast Learning of Audio Structures*. In International Computer Music Conference, Copenhagen, Denmark, Août 2007.
- [Durand 03] Nicolas Durand. Apprentissage du style musical et interaction sur deux échelles temporelles. Dea atiam, Paris 6, June 2003.
- [Eronen 01] Antti Eronen. Automatic musical instrument recognition. Master’s thesis, Tampere University of Technology, April 2001.
- [Essid 04] Slim Essid, Gael Richard & Bertrand David. Musical instrument recognition on solo performances, pages 1289–1292. 2004.
- [Fang 01] Zheng Fang, Zhang Guoliang & Song Zhanjiang. *Comparison of different implementations of MFCC*. J. Comput. Sci. Technol., vol. 16, pages 582–589, November 2001.
- [Faro 08] Simone Faro & Thierry Lecroq. *Efficient Variants of the Backward-Oracle-Matching Algorithm*. In The Prague Stringology Conference, 2008.
- [Feig 92] Ephraim Feig & Shmuel Winograd. *Fast Algorithms for the Discrete Cosine Transform*. Signal Processing, IEEE Transactions on, vol. 40, no. 9, pages 2174–2193, August 1992.
- [Foote 00] J. Foote. *Automatic audio segmentation using a measure of audio novelty*. In Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on, volume 1, pages 452–455 vol.1, 2000.
- [Fu 77] King-Sun Fu & Shin-yee Lu. *A Clustering Procedure for Syntactic Patterns*. Systems, Man and Cybernetics, IEEE Transactions on, vol. 7, no. 10, pages 734–742, 1977.
- [Fujishima 99] Takuya Fujishima. *Realtime chord recognition of musical sound: a system using common lisp music*. In Proc. ICMC, 1999, pages 464–467, 1999.

- [Gillian 11] Nicholas Gillian, R Benjamin Knapp & Sile O'Modhrain. *A machine learning toolbox for musician computer interaction*. In Proceedings of the 2011 International Conference on New Interfaces for Musical Expression (NIME11), 2011.
- [Gómez 04] E. Gómez & P. Herrera. *Automatic Extraction of Tonal Metadata from Polyphonic Audio Recordings*. In AES, 2004.
- [Gómez 06] Emilia Gómez. *Tonal description of polyphonic audio for music content processing*. INFORMS Journal on Computing, vol. 18, no. 3, pages 294–304, 2006.
- [Goto 01] Masataka Goto. *An audio-based real-time beat tracking system for music with or without drum-sounds*. Journal of New Music Research, vol. 30, no. 2, pages 159–171, 2001.
- [Harte 05] Mark Harte Christopher; Sandler. *Automatic Chord Identification using a Quantised Chromagram*. In Audio Engineering Society Convention 118, 5 2005.
- [Holtzman 81] S. R. Holtzman. *Using Generative Grammars for Music Composition*. Computer Music Journal, vol. 5, no. 1, pages pp. 51–64, 1981.
- [Hsu 09] William Hsu & Marc Sosnick. *Evaluating interactive music systems: An HCI approach*. Proceedings of New Interfaces for Musical Expression (NIME), pages 25–28, 2009.
- [Jain 99] A. K. Jain, M. N. Murty & P. J. Flynn. *Data clustering: a review*. ACM Comput. Surv., vol. 31, no. 3, pages 264–323, 1999.
- [Jensen 06] J. H. Jensen, M. G. Christensen, M. Murthi & S. H. Jensen. *Evaluation of mfcc estimation techniques for music similarity*. In European Signal Processing Conference, EUSIPCO, 2006.
- [Jourdan] Emmanuel Jourdan. *What is Zsa.Descriptors?*
- [Kato 03] Ryoichi Kato. *A New Full-Text Search Algorithm Using Factor Oracle as Index*. Rapport technique, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2003.
- [Large 94] Edward W. Large & John F. Kolen. *Resonance and the Perception of Musical Meter*. Connection Science, vol. 6, no. 2-3, pages 177–208, 1994.
- [Large 95] E. W. Large. *Beat tracking with a nonlinear oscillator*. 1995.

- [Lefebvre 00] Arnaud Lefebvre & Thierry Lecroq. *Computing repeated factors with a factor oracle*. In Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms, pages 145–158, 2000.
- [Lefebvre 02] Arnaud Lefebvre, Thierry Lecroq & Joël Alexandre. *Drastic Improvements over Repeats Found with a Factor Oracle*, 2002.
- [Lefebvre 03] A Lefebvre, T Lecroq, H Dauchel & J Alexandre. *FORRepeats: detects repeats on entire chromosomes and between genomes*. Bioinformatics, vol. 19, no. 3, pages 319–26, Feb 2003.
- [Lévy 07] Fabien Lévy. *Soliloque about $[X, X, X \text{ and } X]$* , July 2007.
- [Lewis 00] George E. Lewis. *Too Many Notes: Computers, Complexity and Culture in Voyager*. Leonardo Music Journal, vol. -, pages 33–39, 2012/06/18 2000.
- [Loy 85] Gareth Loy. *Musicians Make a Standard: The MIDI Phenomenon*. Computer Music Journal, vol. 9, no. 4, pages pp. 8–26, 1985.
- [Lu 02] Lie Lu, Hong-Jiang Zhang & Hao Jiang. *Content analysis for audio classification and segmentation*. Speech and Audio Processing, IEEE Transactions on, vol. 10, no. 7, pages 504–516, 2002.
- [Malt 08] Mikhail Malt & Emmanuel Jourdan. *Zsa.Descriptors: a library for real-time descriptors analysis*. Sound and Music Computing, Berlin, Germany, 2008.
- [Mancheron 04] Alban Mancheron & Christophe Moan. *Combinatorial Characterization of the Language Recognized by Factor and Suffix Oracles*. In Proceedings of the Prague Stringology Conference 04, pages 139–153, 2004.
- [Maniatakis 12] Fivos Maniatakis. *Graphs and Automata for the Control of Interaction in Computer Music Improvisation*. Thèse de doctorat, Thèse de Doctorat de l’Université Pierre et Marie Curie, 2012.
- [Miranda 01] Eduardo Reck Miranda. *Evolving cellular automata music: From sound synthesis to composition*. In Proceedings of 2001 Workshop on Artificial Life Models for Musical Applications, 2001.
- [Moog] Robert Moog. *Bob Moog: A Timeline*.
- [Nouno 08] Gilber Nouno. *Suivi de Tempo Appliqué aux Musiques Improvisées*. PhD thesis, UPMC Paris 6, 2008.

- [Obin 11] Nicolas Obin. *MeLos: Analysis and Modelling of Speech Prosody and Speaking Style*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2011.
- [Obin 13a] Nicolas Obin, Anne Lacheret, Christophe Veaux & Victorri Bernard. The tonal annotation: Stylization of complex melodic contours over arbitrary linguistic segments. Benjamins, Amsterdam, 2013.
- [Obin 13b] Nicolas Obin, François Lamare & Axel Roebel. *Syll-O-Matic: an Adaptive Time-Frequency Representation for the Automatic Segmentation of Speech into Syllables*. In ICASSP, Vancouver, Canada, Mai 2013.
- [OpenMusic] OpenMusic. <http://forumnet.ircam.fr/product/openmusic/>.
- [Pachet 13] François Pachet, Pierre Roy, Julian Moreira & Mark d’Inverno. *Reflexive loopers for solo musical improvisation*. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 2205–2208. ACM, 2013.
- [Papadopoulos 99] George Papadopoulos & Geraint Wiggins. *AI methods for algorithmic composition: A survey, a critical view and future prospects*. In AISB Symposium on Musical Creativity, pages 110–117. Edinburgh, UK, 1999.
- [Papadopoulos 07] Hélène Papadopoulos & Geoffroy Peeters. *Large-scale study of chord estimation algorithms based on chroma representation and HMM*. In Content-Based Multimedia Indexing, 2007. CBMI’07. International Workshop on, pages 53–60. IEEE, 2007.
- [Peeters 06] Geoffroy Peeters. *Chroma-based estimation of musical key from audio-signal analysis*. In ISMIR, Victoria, Canada, Octobre 2006.
- [Poirson 02] Emilie Poirson. Simulation d’improvisations à l’aide d’un automate de facteurs et validation expérimentale. Dea atiam, Paris 6, July 2002.
- [Pressing 88] Jeff Pressing. *Improvisation: methods and models*, chapitre 7. Oxford Science Publications, 1988.
- [Puckette 91] Miller Puckette. *Combining Event and Signal Processing in the MAX Graphical Programming Environment*. Computer Music Journal, vol. 15, no. 3, pages pp. 68–77, 1991.
- [Puckette 98] Miller S. Puckette, Miller S. Puckette Ucsd, Theodore Apel, Ucsd (tapel@ucsd. Edu, David D. Zicarelli & Cycling (www. Cycling. Com. *Real-time audio analysis tools for Pd and MSP*, 1998.

- [Roads 04] Curtis Roads. Microsound. MIT press, 2004.
- [Robin 09] Yann Robin, Gérard Assayag, Alain Billard, Benny Sluchin & Nicolas Crosse. *Projet de recherche OMAX (étape I) : Composer in Research Report (2009)*. Rapport technique, Médiation Recherche et Création, 2009.
- [Ron 96] Dana Ron, Yoram Singer & NAFTALI TISHBY. *The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length*. In Machine Learning, pages 117–149, 1996.
- [Rowe 92] Robert Rowe. Interactive music systems: machine listening and composing. MIT press, 1992.
- [Rowe 09] Robert Rowe. *Split Levels: Symbolic to Sub-symbolic Interactive Music Systems*. Contemporary Music Review, vol. 28, no. 1, pages 31–42, 2009.
- [Schankler 11] Isaac Schankler, JordanB.L. Smith, AlexandreR.J. François & Elaine Chew. *Emergent Formal Structures of Factor Oracle-Driven Musical Improvisations*. In Carlos Agon, Moreno Andreatta, Gérard Assayag, Emmanuel Amiot, Jean Bresson & John Mandereau, editeurs, Mathematics and Computation in Music, volume 6726 of *Lecture Notes in Computer Science*, pages 241–254. Springer Berlin Heidelberg, 2011.
- [Schwarz 06] Diemo Schwarz, Grégory Beller, Bruno Verbrugghe, Sam Brittonet al. *Real-time corpus-based concatenative synthesis with catart*. In Proc. of the Int. Conf. on Digital Audio Effects (DAFx-06),(Montreal, Quebec, Canada), pages 279–282. Citeseer, 2006.
- [Surges 13] Greg Surges & Shlomo Dubnov. *Feature Selection and Composition using PyOracle*. In Massachusetts Northeastern University Boston, editeur, 2nd International Workshop on Musical Metacreation (MUME). AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2013.
- [Thom 00] Belinda Thom. *BoB: an interactive improvisational music companion*. In Proceedings of the fourth international conference on Autonomous agents, AGENTS '00, pages 309–316, New York, NY, USA, 2000. ACM.
- [Wessel 02] David Wessel & Matthew Wright. *Problems and Prospects for Intimate Musical Control of Computers*. Computer Music Journal, vol. 26, no. 3, pages pp. 11–22, 2002.

- [Williams 91] Ross Neil Williams. Adaptive data compression, volume 110. Springer, 1991.
- [Wright 98] Matthew Wright & David Wessel. *An improvisation environment for generating rhythmic structures based on north indian "Tal" patterns*. In International Computer Music Conference, Ann Arbor, Michigan, 1998.
- [Xu 05] Rui Xu & II Wunsch D. *Survey of clustering algorithms*. Neural Networks, IEEE Transactions on, vol. 16, no. 3, pages 645–678, 2005.
- [Young 08] Michael Young. *NN Music: Improvising with a ‘Living’ Computer*. In Richard Kronland-Martinet, Sølvi Ystad & Kristoffer Jensen, editeurs, Computer Music Modeling and Retrieval. Sense of Sounds, volume 4969 of *Lecture Notes in Computer Science*, pages 337–350. Springer Berlin Heidelberg, 2008.
- [Zicarelli 87] David Zicarelli. *M and Jam Factory*. Computer Music Journal, vol. 11, no. 4, pages pp. 13–29, 1987.
- [Ziv 78] J. Ziv & A. Lempel. *Compression of individual sequences via variable-rate coding*. Information Theory, IEEE Transactions on, vol. 24, no. 5, pages 530–536, 1978.
- [Zolnay 05] András Zolnay, Ralf Schlüter & Hermann Ney. *Acoustic Feature Combination for Robust Speech Recognition*. In Proc. IEEE Intern. Conf. on Acoustics, Speech, and Signal Processing, pages 457–460, 2005.

Appendices

Appendices A

Algorithms

A.1 Mel Frequency Cepstrum Coefficients Computation

For the timbral listening of our system, we base our analysis on the extraction of Mel Frequency Cepstral Coefficients (MFCCs). These multi-scale coefficients are commonly computed with the following steps:

1. Take the fast Fourier transform (FFT) of the signal
2. Slice and regroup the powers of the spectrum onto the mel scale (a perceptual scale of pitches) using triangular overlapping windows
3. Take the logs of the powers for each mel band
4. Compute the discrete cosine transform (DCT) of the list of mel log powers, as if it were a signal
5. MFCCs coefficients are the amplitudes of the resulting spectrum

A.2 Factor Oracle Incremental Algorithm

The Factor Oracle algorithm we use is based on [Allauzen 99a] and includes the computation of the *Length of Repeated Suffix* (*lrs*) proposed in [Lefebvre 00]. In the work presented in this thesis, we took care to include the improvements for *lrs* computation described in [Lefebvre 02]. We present here a condensed version of the algorithm in pseudocode.

Assuming the automaton is built to state $i - 1$ (including *lrs*) and we want to add the state i corresponding to the character σ_i

Create a new state numbered i

Assign a new transition labelled σ_i from state $(i - 1)$ to the new state:

$$\delta(i - 1, \sigma_i) = i$$

Retain state $i - 1$: $\pi_1 = i - 1$

Iteratively backtrack suffix links from state $i - 1$: $k = S(i - 1)$ **doing**

if no transition from state k is labelled by σ_i

then Assign a new transition from state k to the new state: $\delta(k, \sigma_i) = i$

Retain this suffix: $\pi_1 = k$

and **Track** its suffix link: $k = S(k)$

else End backtrack

if backtrack ended before state 0: $k > -1$

then an existing pattern is recognise and linked: $S(i) = \delta(k, \sigma_i)$

and $lrs(i) = \text{LengthCommonSuffix}(\pi_1, S(i) - 1) + 1$

else the suffix link goes to state 0: $S(i) = 0$

and $lrs(i) = 0$

for all the other suffix links $S(j)$ pointing on the same state (in increasing order): $\forall j \ / \ S(j) = S(i)$

if $lrs(j) = lrs(i)$ and $\sigma_{j-lrs(j)} = \sigma_{i-lrs(i)}$

then $S(i) = j$ and $lrs(i) = lrs(i) + 1$

and **exit**

LengthCommonSuffix(π_1, π_2) =

if π_2 is the suffix of π_1 : $\pi_2 = S(\pi_1)$

then Return $lrs(\pi_1)$

else While no common suffix is found: $S(\pi_2) \neq S(\pi_1)$

Backtrack suffix links from π_2 : $\pi_2 = S(\pi_2)$

Return the least *lrs* before common suffix: $\min(lrs(\pi_1), lrs(\pi_2))$

A.3 Factor Oracle Clustering Algorithm

The clustering algorithm proposed for Factor Oracle graph is based on a weighting system which allows a reinforcement mechanism and gives thanks to a threshold the boundaries to potentially fragmented regions similar in musical content. It uses a list of weights (integers) associated with every state (and initialized to 0). It is linear in time and space. Three control parameters are required:

- *minCtxt*: a minimal context length.
- *local*: a value to decide the local/global character of a link.
- *w*: a value to define a zone of “influence” of a link.

See Section 6.3.3 for more details. Defining:

- $W(i)$ the weight, and assuming $W(k)$ is known for all k from 0 to $i - 1$,
- I a temporary interval or union of interval of states,
- $r(i)$ the region ID and assuming it is known for all states with a non zero weight,
- a global region counter r_g (initialized to 0)
- a local region ID r_I
- a local list of region ID lr_d ,

we consider state i , its suffix $S(i)$ and context length $lrs(i)$:

If the context of the link is high enough: $lrs(i) \geq minCtxt$ **do**:

If the link is considered as local: $i - S(i) \leq local$

Then there is only 1 influence zone containing both ends of the link:

$$I = [S(i) - w * lrs(i), i]$$

Else the links is considered as global: $i - S(i) > local$

there are 2 influence zones containing each end of the link:

$$I = [S(i) - w * lrs(i), S(i)] \cup [i - w * lrs(i), i]$$

Determine the region ID: $r_I = FindID(I)$

Store the list of ID to drop: $lr_d = DropID(I, r_I)$

For every state in the influence zone: $\forall k \in I$

Increase weight: $W(k) = W(k) + 1$

and **Assign** the ID: $r(k) = r_I$

For every state of FO which has a non zero weight except the influence zone (already done): $\forall k \in [0, i] \setminus I / W(k) > 0$

If the region ID is in the drop list: $r(k) \in lr_d$

Then Overwrite ID: $r(k) = r_I$

```

FindID( $I$ ) =
    Find the state with maximal weight in the influence zone  $I$ :
         $i_{max} = \operatorname{argmax}_I(W(i))$ 
    If a non zero weight is found:  $W(i_{max}) > 0$ 
        Then Return its ID:  $r(i_{max})$ 
    Else no state in the influence zone is already in a region:  $W(i_{max}) = 0$ 
        Return the current region counter:  $r_g$ 
        and Increment it:  $r_g = r_g + 1$ 

DropID( $I, r_I$ ) =
    List all the region ID present in  $I$  (if any):  $lr_d = W(I)$ 
    Remove doubles in the list:  $\operatorname{unique}(lr_d)$ 
    Remove the chosen region ID (if present):  $\operatorname{remove}(r_I, lr_d)$ 
    Return the list of ID to drop:  $lr_d$ 

```

A.4 Virtual Fundamental Computation Code

In the case of a polyphonic MIDI input stream, we extract an approximate virtual fundamental as label of the slices (see 6.1.2.1). The virtual fundamental is a (low) frequency which multiples fall on the notes of the slice. In other words, the virtual fundamental is a possibly missing note which would be the bass note of the harmonic chord of the slice if it were purely harmonic — that is if all the frequencies were related with fractional ratios. The main function to compute this virtual fundamental is a recursive function that we implemented in C. It takes as argument the first and last elements of a table listing the frequencies in the slice (*freqs* and *end*), a range of values to look for a common divisor (*divmin* and *divmax*) and the floating point approximation allowed when comparing the actual frequencies of the chord (*approx*). Here is the code (in C) of the this main function:

```
float rec_virfun(float* freqs, float* end, float divmin, float divmax,
    float approx)
{
    float inf, sup;
    float quo_min, quo_max;
    float quotient;
    float resu = 0;
    if (divmin <= divmax)
    {
        if (freqs==end)
            return((divmin + divmax) / 2.);
        else
        {
            sup = freqs[0] * (1 + approx);
            inf = freqs[0] / (1 + approx);
            quo_min = ceil(inf/divmax);
            quo_max = floor(sup/divmin);
            quotient = quo_min;
            while (quotient <= quo_max)
            {
                resu = rec_virfun(freqs+1,end, max(inf/quotient, divmin), min(sup/
                    quotient, divmax), approx);
                if ((int)resu)
                    return resu;
                quotient++;
            }
            return 0.;
        }
    }
    return 0.;
}
```

A.5 C++ Classes

Along this thesis, we presented structures to hold description information and graph modeling of the input streams. We present a few details about the implementation of those in C++ language.

A.5.1 Data Structure

The data structure rely on a main C++ class named *O_data* which includes a vector of *O_label* elements containing the actual information extracted from the input stream. As we explained along this work, several descriptions are implemented in our system. This is achieve easily with the inheritance mechanism of C++. This way, the *O_label* class is the base class for every descriptions, it contains all the common attributes: date, duration, phrase and section number etc. Five child classes are derived from the *O_label* class to specialize it for each description: characters (for testing purpose), floats (generic data and Chromagrams), MIDI (polyphonic MIDI slices), pitch (melodic description), spectral (MFCCs). [Figure 39](#) illustrates this hierarchy of classes.

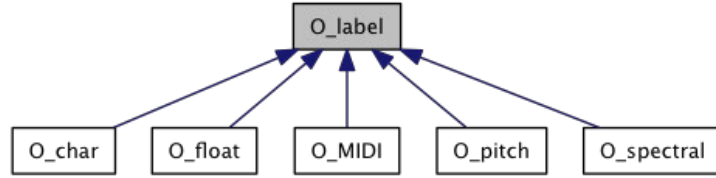


Figure 39: Inheritance Diagram of the Label Class

A.5.2 Graph Structure

The graph structure rely on a main C++ class named *O_oracle* which includes a vector of *O_state* elements containing the actual states of the graph. This class also holds the two hashtables (C++ map structures) for the conversion between state numbers and date on the medium timeline. The *O_state* class contains the information for each state of the graph: transitions (*trans*), suffixes (*suff*), reversed suffixes (*r_suff*) etc. [Figure 40](#) illustrate this implementation with the collaboration diagram of the *O_oracle* class.

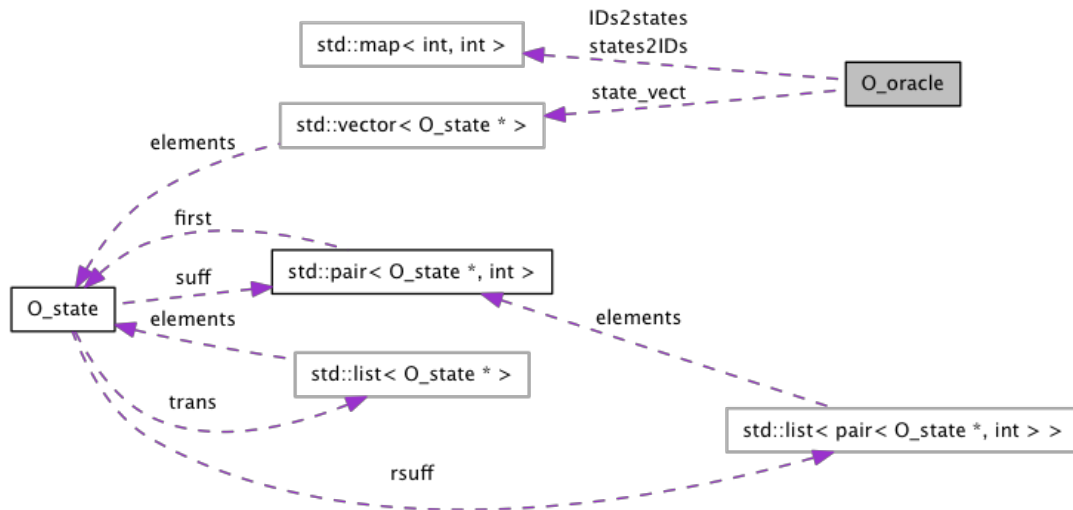


Figure 40: Collaboration Diagram of the Factor Oracle Structure Class

Appendices B

Publications

B.1 NIME 2012

2012, we wrote and submitted an article for the 12th international conference on New Interface for Musical Expression. It has been accepted and we presented our work in this conference on the 23rd of May. We include this article in this appendix. It describes several aspects of the research of our work, in particular, the spectral description of audio streams, the visualization part and some architecture considerations.

OMaxist Dialectics: Capturing, Visualizing and Expanding Improvisations

Benjamin Lévy
STMS Lab
IRCAM, CNRS, UMPC
1, place Igor Stravinsky
75004 Paris
Benjamin.Levy@ircam.fr

Georges Bloch
CNSMDP & IRCAM
209 av. Jean Jaurès
75019 Paris
gbloch@cnsmdp.fr

Gérard Assayag
STMS Lab
IRCAM, CNRS, UMPC
1, place Igor Stravinsky
75004 Paris
Gerard.Assayag@ircam.fr

ABSTRACT

OMax is an improvisation software based on a graph representation encoding the pattern repetitions and structures of a sequence, built incrementally and in real-time from a live Midi or Audio source. We present in this paper a totally rewritten version of the software. The new design leads to refine the spectral *listening* of OMax and to consider different methods to build the symbolic alphabet labeling our symbolic units. The very modular and versatile architecture makes possible new musical configurations and we tried the software with different styles and musical situations. A novel visualization is proposed, which displays the current state of the learnt knowledge and allows to notice, both on the fly and a posteriori, points of musical interest and higher level structures.

Keywords

OMax, Improvisation, Machine Learning, Machine Listening, Visualization, Sequence Model, Software Architecture

1. PRINCIPLES

OMax [2][4][6] is a software environment oriented towards human-machine interaction for musical improvisation. It learns in real-time by listening to an acoustic musician and extracting symbolic units from this stream. It then builds a sequence model on these units constituting an internal knowledge. The internal model of OMax (named Factor Oracle [1][5]) is a graph which incrementally recognizes the repeated factors (patterns and subpatterns) of any symbolic string. Factor Oracle only needs strict on the symbolic units to be built. They can be called *letters* over a formal *alphabet*.

OMax is able to navigate through this model to create one or several “clones” of the musician feeding the system [2]. These “clones” are recombinations of the original discourse justified by the model and realized by cutting and pasting the original material in real-time (audio editing or MIDI deslicing, see 2.1 and [4]). This *stylistic reinjection* [4] creates a specific musical interaction in which the musician is constantly confronted to a reinterpreted version of his own playing. It emphasize the memory effects and usage of (self-)reference found in improvisation contexts such as collective free improvisation or jazz.

This technique, close to concatenative synthesis, has been extended in a video version of OMax [6]. The video of the musician (or any other visual material) is aligned and recombined as a slave synchronization with audio.

Previous versions of OMax made use of two very different software environment, Max/MSP and OpenMusic (hence its name “OMax”) respectively for real-time signal processing and abstract model construction and navigation processes. In this paper we will present a new version of OMax we developed solely based on Max5. We will describe the material it is now able to “understand” and how it achieves this. Then we will explain the architecture of this new version and its novel visualization and interactions possibilities. Finally we will discuss a few situations we encountered testing with musicians.

2. MATERIAL

Historically, OMax emerged from studies on *stylistic simulation* by Shlomo Dubnov and Gérard Assayag and Marc Chemillier’s research on improvisation modeling. It has since gained considerable attention from improvising musicians worldwide through dozen of concerts, workshops and master-classes.

2.1 Audio vs. MIDI

Both audio and MIDI streams can constitute a source for OMax learning. Though MIDI is already a flow of abstract data, it still needs to be segmented into consistent units to be learnt. In the case of a monophonic MIDI input, segmentation is trivial: a unit for a note. However a polyphonic MIDI input feeds a continuous and overlapping flow of notes to be separated into polyphonic chord-like *slices* (Figure 1). This slicing happens with the birth and death of significant events and has been described in [4]. It does not imply any specific labeling (or lettering) to tag the symbolic units to be compared and learnt.

In the case of an audio input, prior to any kind of grouping, information needs to be extracted from samples. We have in OMax two very different types of audio analysis which infer two different kind of *listening*. The first type of analysis is pitch extraction. For now, we are able to deal only with monophonic pitch extraction and use the YIN algorithm [8]. To make the output of yin more consistent and stable, we use a statistical analysis with concurrent voting agents gathering pitches over fixed windows [3]. Stable pitches are gathered and grouped into units when equal and consecutive to form notes. At this point, we are brought back to the simpler case of monophonic MIDI-like data.

We summarize the different steps to form consistent units for the different type of analysis in Figure 1. From an *audio* stream, *micro-units* are constituted with an initial *framing* and the *extraction* of a descriptor of the signal. Depending

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

NIME’12, May 21 – 23, 2012, University of Michigan, Ann Arbor.
Copyright remains with the author(s).

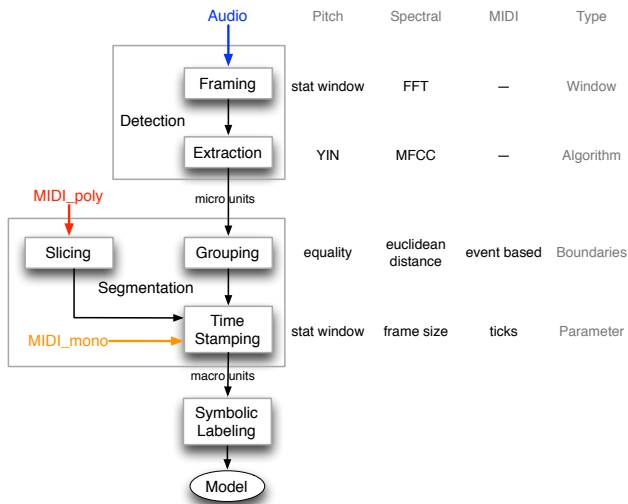


Figure 1: Analysis chain of the input

on the *algorithm* of extraction, different sliding *windows* are used. Then, in a second phase, *macro-units* can be put together by *grouping* similar and consecutive micro-units. The *boundaries* are defined by different criteria depending on the type of units. Then, to achieve the *segmentation*, we *time-stamp* these units linking them with the chunk of signal they encode — and we take into account the latency and time specific *parameters* of each methods.

2.2 Spectral Segmentation

The second type of audio analysis in OMax uses spectral descriptors. They allow the system to *listen* and play with wider types of sound including noises, percussions or different instrumental playing modes. Mel Frequency Cepstral Coefficients has been proven (for example in [7]) to be easily computable, very compact and reliable for recognition tasks. It is suitable to constitute our micro-units. However, MFCCs are vectors of floating-point multi-scale coefficients. It is thus necessary to have a clustering algorithm to put together consistent and meaningful macro-units.

Rather than using a rough quantization as in [6], we designed a weighted Euclidean clustering algorithm. This algorithm achieves both the macro-grouping and the symbolic labeling at once.

For every incoming MFCC vector, dropping the first coefficient (which represent the overall energy of the slice), we weight the remaining coefficients according to profiles to help enhancing the differences we want to discriminate. These profiles have been adjusted empirically along experiments with several instruments and playing styles. Then we compare the vector to the clusters already encountered by computing the Euclidean distance with their centroids and we determine (with an adjustable threshold) if it can be identified or if it constitute a new letter in our spectral alphabet.

2.3 Alphabets

This incremental clustering mechanism creates a growing alphabet of spectral letters in a multidimensional Euclidean space, meaning that the system is able to discover the symbolic units along the musicians playing. It allows us to have an ad-hoc definition of the clusters depending on the input material. Regions of the timbre space thoroughly explored by the musician will have therefore more clusters than other regions, less brought into play. On the other hand, pitch

classes and quantized spectral vectors constitute a fixed and predetermined alphabet.

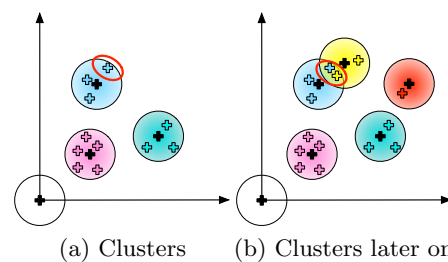


Figure 2: Example of mutation in spectral clustering

Another effect of this classification is the possibility of mutations according to the past of the learning. Depending on the material already encountered and known in the system — which means in our MFCC space, depending on the clusters already defined —, the same (or close) spectral vectors can be identified differently. An example of such a mutation is given in 2D Figure 2: vectors framed in red, although very close, are recognize differently depending on the moment they appear. The first vector (Figure 2a) is considered in the “blue” class while the second occurrence is closer to a more recently defined “yellow” cluster (Figure 2b). The appearance of the “yellow” cluster in between did not change the class of the previously encountered vector but it modifies the classification of forthcoming material. This also has a musical meaning: a specific playing mode can be considered as an accident and identified as close to an already heard mode if encountered only once. But the same playing mode, if developed by the musician may be rightfully creating one or more new cluster(s) — letter(s) in our spectral alphabet — to describe its slighter differences. Thus the mapping between signal units and symbols has become adaptive instead of being rigid, which reflects an important aspect of implicit learning in human interaction.

3. ARCHITECTURE

Further than being able to use different segmentations and alphabets, the whole software has been reprogrammed to adopt a very flexible and versatile architecture presented Figure 3.

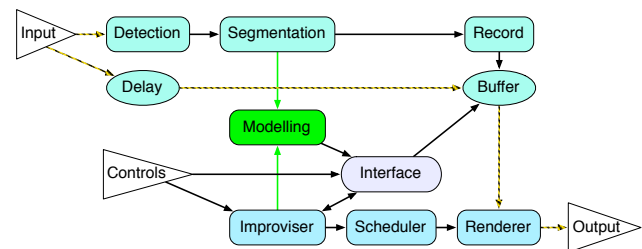


Figure 3: Functional diagram of OMax 4.x

3.1 Modularity

First of all, a modular approach has been taken to develop the different functions needed (Figure 3). The audio input is split in two streams. One is directly *recorded* into a *buffer* while the other enters a two stage process to constitute macro-units. The *detection* stage regroupes both the *framing* of the signal and the *extraction* of a descriptor to get micro-units. Then the *segmentation* stage is in charge of the *grouping* and the *time-stamping* to define macro-units

and date them (see 2.1 and Figure 1).

Thanks to a fixed *delay*, the analysis chain described in the previous paragraph has a retro-action on the recording to start and stop it consistently (mainly avoiding to record long silences).

Once labelled (see 2.3), the symbolic units are fed incrementally to the model which will be read and navigated by improvising agents. To create a new “clone”, the *improviser* is in charge of reading and jumping in the graph to create a new coherent path — a musical variation on what the acoustic musician played until now. The *scheduler* reading this path puts it back “in time” with the possibility of time-stretching or backward reading. Finally, the *renderer* effectively reads and crossfades the different *buffer* chunks corresponding to this new musical discourse.

3.2 Parallelism

The modularity of the new design allows now OMax to run in parallel different analysis and labeling and to acquire this way a multi-description model on a single input. The most typical setup making use of this is to run both the pitch and spectral analysis on a single audio input, building thus two Factor Oracles which refer to the same buffer and time-stamping.

Another very common option the new architecture allows, is to have several independent “clones” improvising on the same model. For that, we duplicate the whole generation chain, *improviser-scheduler-renderer*. Each *improviser* is able to have its own path on the common model of the input.

Besides these simple configurations, more sophisticated schemes are possible to create different musical linkage inside OMax. An example of these configurations is to have two “clones” playing the same computer-based improvisation — possibly at different speed or with transposition — ie. the same path in the model.

3.3 Versatility

Multiplying and combining at will the modules — the only limit being the power of the computer — we can shape and adapt our OMax setup to very diversified musical situations. From one to several inputs in parallels with one or more descriptions and models built on each of them and one to several “clones” improvising together or independently, the variety of arrangement allows us to start playing OMax almost as an instrument. We will see in 5 how OMax can now take its own musical part in different musical ensemble.

4. VISUALIZATION

On top of the redesign of the architecture, OMax 4.x adds to the software a whole new interface (thanks to Jitter, the graphical part of Max5). This interface is based on a visualization of the current state of one or two models being built. It takes the simple form of a growing horizontal and linear timeline representing what has already been learnt — time “flows” from left to right: left is farther in the past, right is the most recent element. Above and below this timeline can be shown some links of the Factor Oracle graph indicating the occurrences of repeated patterns. An example is presented Figure 4.

4.1 Live

Although unadorned, this feedback constantly given on the current state of the model revealed itself to be very efficient to locate and memorize on the fly musically interesting points. Seeing patterns of links appearing on the screen related to what the musician is currently playing allows to

associate musical passages with particular sections of the graph. And retrieve them easily later on.

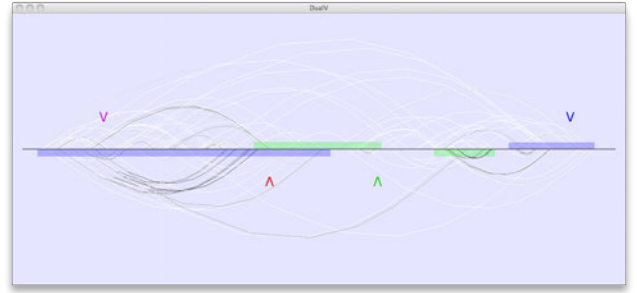


Figure 4: “Clones” and regions on the sequence visualization

While playing with OMax, the visualization is also a new interface to interact with the software. Indeed, as introduced Figure 4, the different “clones” OMax is able to play are also pictured on the display with moving arrows above and below the timeline. These arrows reflect the current position of each “clone” in the model and jump along the links when improvising new paths (see [2]). With the mouse, regions of the graph can be selected (green and blue sections on Figure 4) to constrain “clones” to specific sections of what has been learnt.

Many other features and functions that can not be detailed here have also been implemented to make the visualization as intuitive, interactive and useful as possible.

4.2 A Posteriori

Following the same visualization principles with different musical materials and visuals, we noticed that examining our model, a posteriori, could reveal important higher level structures. Moreover, this analysis may show, in real-time, these higher level formal structures with little graphical processing. For example, with colors to identify patterns and thickness to stress the durations, we can enhance certain aspects of the analysis and help discover interrelations of parts.

Figure 5 shows the analysis of a saxophone improvisation.

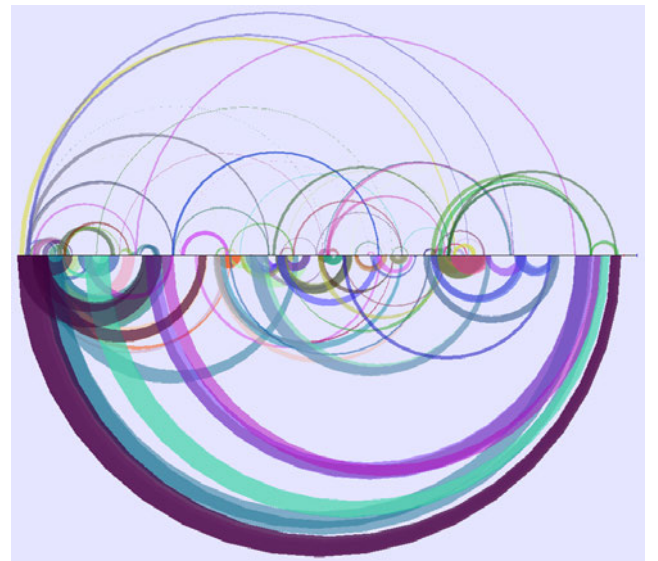


Figure 5: Visualization with colors and thickness

On the bottom half (below the horizontal timeline, which correspond to the model built on pitch), we can see with

the important number of arches, that the material introduced at the beginning of the session is developed for about a third of the total duration then the rest of the improvisation is very different (few links/arches connecting it with the beginning). Finally a short but strong recapitulation of the first material is played: many arches connecting the last moments of the improvisation with the beginning.

This way, the new interface of OMax may open possibilities for a simple yet efficient musicology tool to study pieces or improvisations and compare them. Automatic segmentation of formal high level structures in musical sequences has been experimented as a derivation from this visualization but is not presented in this paper.

5. OMAX IN ACTION

In the last two years, we had the opportunity to play with OMax in numerous and various situation. Here are some of the rough observations we could empirically make. We encountered mainly two kinds of musical ensembles. The first is a duet between an *acoustic musician* improvising and feeding the system and an *electronic musician* controlling the software. The second common situation is to include OMax (and the person controlling it) into a small musical group (usually up to ten musicians).

5.1 Duets

Naturally, the first idea to try out a software like OMax is to ask a musician to feed the system with an improvisation and play along with the computer based “clones”. This allows us to test at the same time how the system *understands* the musicians playing and how they musically interact. Multiplying this kind of experiments with very different instruments helped us refine the different analysis stages of the software. While multiplying the different styles showed us the musical possibilities and limits of such a dialog.

It soon appeared in this situation that the choices made by the person controlling OMax strongly influence the course of the session. Thus, the acoustic musician could notice very fast the interventions and the character of the electronic musician, and often referred to this partner as a mixed entity constituted of OMax and the person controlling it, rather than as the software by itself. Duets of short duration work very well most of the time and some characteristics of purely acoustic improvised duets can be found.

However, when musicians want to make it longer (and confront it with a public), they usually feel more confident (pre)setting narrative elements. They frequently start to predefine the type of interaction with the system, mostly in terms of when to play and what part of the learning to use (which could be identified to a music score for OMax player). Or they feed OMax with some prepared material (pre-composed and/or pre-recorded) and improvise with these. Some previous experiments have been done in this last direction such as in *Peking Duck Soup* (Bloch 2008 [6]).

5.2 Groups

The other kind of musical situation OMax has regularly been involved in could be qualified as “collective free improvisation” or “band”, that is a small group of musicians playing together, either improvising freely or with strong predetermined structures. In this cases, one or two instances of OMax were used and each of them could *listen* and learn from one or two of the acoustic musicians. Technically, the question of feedback of other musicians and OMax into the microphones gets really serious in these situations.

Despite the difference of nature between the acoustic instruments and OMax — which usually does not have its own sound source — the computer and its player could really find a place comparable to the other musicians in the group. And the work to build a whole concert program was in almost every aspects similar to the work with an ensemble of only acoustic instruments.

6. FURTHER DIRECTIONS

Both the entire renovation and the musical testing feed the research around the musical aptness of OMax.

The explorations already started around the notion of alphabet in OMax constitute a strong direction of our future work. It puts up questions about the consistency of these alphabets and ways to build finer and even more meaningful alphabets. On-going research about information geometry or other content-oriented machine learning techniques may nourish our reflexion.

Multiplying the alphabets and models built in parallel gives opportunities to exploit more or differently the knowledge of the system. The various descriptions of an input sequence may be queried concurrently or jointly to help drive the system towards more expressive and speaking musical paths. This database oriented approach could use some reinforcement learning technique in order to make interesting solutions emerge.

Finally, numerous musical situation suggested we should investigate the relation of OMax with the rhythm both in a broad (phrase leading, endings...) and in a strict (pulse, time signature...) sense. A strong anticipation mechanism may be a relevant approach to these problems.

7. REFERENCES

- [1] ALLAUZEN, C., CROCHEMORE, M., AND RAFFINOT, M. Factor oracle: a new structure for pattern matching. In *SOFSEM'99* (Milovy, Czech Republic, Nov. 1999), vol. 1725 of *LNCS*, Springer-Verlag, pp. 291–306.
- [2] ASSAYAG, G., AND BLOCH, G. Navigating the oracle: a heuristic approach. In *International Computer Music Conference '07* (Copenhagen, Denmark, Août 2007), pp. 405–412.
- [3] ASSAYAG, G., BLOCH, G., AND CHEMILLIER, M. Omax-ofon. In *Sound and Music Computing (SMC) 2006* (Marseille, France, May 2006).
- [4] ASSAYAG, G., BLOCH, G., CHEMILLIER, M., CONT, A., AND DUBNOV, S. Omax brothers : a dynamic topology of agents for improvisation learning. In *Workshop on Audio and Music Computing for Multimedia, ACM Multimedia 2006* (Santa Barbara, USA, Octobre 2006).
- [5] ASSAYAG, G., AND DUBNOV, S. Using factor oracles for machine improvisation. *Soft Computing* 8-9 (2004), 604–610.
- [6] BLOCH, G., DUBNOV, S., AND ASSAYAG, G. Introducing video features and spectral descriptors in the omax improvisation system. In *International Computer Music Conference '08* (2008).
- [7] BROWN, J. C., HOUIX, O., AND MCADAMS, S. Feature dependence in the automatic identification of musical woodwind instruments. *J Acoust Soc Am* 109, 3 (Mar 2001), 1064–72.
- [8] DE CHEVEIGNÉ, A., AND KAWAHARA, H. Yin, a fundamental frequency estimator for speech and music. *JASA: Journal of the Acoustical Society of America* 111 (2002), 1917–1930.

B.2 In Progress

2013, we wrote an article describing the result of the whole work presented in this thesis. This article is not finished and has not been accepted for publication yet. Our approach in this article is to present the new usage of OMax in playing context enabled by our research. We join this article to this thesis in this appendix. It briefly summarizes how OMax works then it details the new influence of the context and the new possibilities for the supervisor to control the content of the computer based generation. Finally it gives two examples of *real-life* utilizations.

Towards a Multi-Description, Context Aware and Playable OMax

Benjamin Lévy

IRCAM, CNRS, UPMC
1, place Igor Stravinsky
75004 Paris
`Benjamin.Levy@ircam.fr`

Abstract. OMax is a software which learns in real-time from a musician’s playing and is able to co-improvise with him/her. This paper presents new improvements to support automatic or manual adaptation of OMax’s generation to the current musical context. Thanks to multiple descriptions, semi-automatic selection and a weighting and constraints system we orient the dynamics, rhythm or musical content of the computer discourse. We briefly illustrate our research with a new prototype developed for two concerts in the Jazz scene.

Keywords: OMax, Improvisation, Machine Listening, Factor Oracle, Musical Context, Supervised Computer System, Music Generation

1 Introduction

Studies on style modeling through machine learning and statistical models have been turned over several years ago to generate new musical discourse which would exhibit many aspects of the original music [5]. The development of OMax software started in this field and improved over the years to be able to learn patterns and generate in real-time new variations for both MIDI and audio streams [3]. Several techniques and scenarios have been explored with OMax, feeding our research. And the participation to musical co-improvisations between acoustic musicians and supervised computer(s) have become the most typical use of the software.

Thanks to its *agnostic* and constantly learning core and with several methods to enhance the variety of its discourse, OMax (usually with its supervisor) has been able to relevantly challenge a large panel of musicians. However the computer-based generation still seems to lack important features of improvisation to widen its horizon, notably a grasp of the musical context and a certain versatility. This gap is the main focus of our current work. We start by describing the essential learning and generating core of OMax in a very concise and practical way. Then we present our work to bend and orient the musical discourse of the computer through automatic contextual annotations and new steering possibilities offered to the supervisor. We describe as well how these improvements have been set up in new prototypes and tested in musical situations.

2 Improvisation Core

The very first and strong hypothesis at the core of OMax is the *sequential* nature of a musical performance. This grounding does not assume anything about the content of the units in the sequences but allows us to use very well known and studied *statistical modeling* and in particular *context models* as the knowledge structure of our system. After testing classical Markov Chains then Variable Markov Chains (especially with Incremental Parsing and Prediction Suffix Trees [6]) we chose the Factor Oracle automaton as the backbone of our system [2]. The Factor Oracle graph is acyclic, linear in time and space and recognizes at least all the factors (ie. subsequences of any length) of the word (input sequence) it is constructed on.

2.1 Chaining Patterns

The two strongest assets of this structure in our musical application is the incremental algorithm, originally given in [1] and improved in [8], and the linear spine of $m + 1$ states and $m + 1$ transitions if the input word is of length m , which exactly represents the musical sequence given as input. Besides the linear (from states i to $i + 1$) and forward transitions, Factor Oracle presents backward transitions named *suffix links* which connect occurrences of repeated factors in the input sequence. We intensively use these suffix links as possible variation paths for generating new musical phrases on the learnt material (see [2,3] for more details). The suffix characteristic of these links— the same pattern is present as a suffix at both ends of each link and the link is labelled by the size of the common pattern named *lrs* (length of repeated pattern) — ensures a smooth concatenation, on the symbolic level, of two connected chunks. This *jumping* mechanism serves as the essential supply of variations in OMax.

One improvisation feature already appears in this succession of literal citations and variations: the rate of these *jumps*. It can be translated in the musical vocabulary into a *difference ratio* or a *distance* from the original material. We usually call it *continuity*, in regard to the audible length of the chunks and the overall fragmentation of the result. Less jumps means longer contiguous chunks from the original sequence in the computer generation, thus noticeably closer discourse to the original input.

2.2 Continuity

Even though the chaining of patterns is justified by our underlying symbolic model, perceptively, either with MIDI or in the audio domain, the connections between these patterns can sometimes sound rough or very surprising. Drawing the thin æsthetic line between a relevant innovative surprise and a rough undesirable collage is totally out of our research scope especially because it widely depends on the music content and intent — which are artistic choices. However, giving the possibility for a system such as OMax to interact with the musician(s) on a chosen spot considering his/her æsthetic line is the very way to make OMax

attractive and pertinent, especially in the situation of improvisation. Therefore, we added in the process of improvising (ie. *jumping* in OMax) a very large number of parameters to allow the system and its supervisor to smooth over (or make uneven) transitions and/or orient the content of the computer based generation.

Firstly, a threshold on the lrs of the suffix links to look for in the Factor Oracle (see 2.1) conditions the number of acceptable jumps. But besides the symbolic units fed to the graph, three musical qualities are also extracted from the input stream and can be used to weight the possible jumps and discriminate between them. The *energy* of the pattern, in the form of MIDI velocity or of the first MFCC coefficient when working on spectral description (see 4.1 and [9]), is computed. To even (or not) transitions, energies of the pattern we *jump from* and the pattern we *jump to* are compared: closer energies will result in smoother jumps. The same principle is applied with a *rhythmic coefficient* which characterizes the duration's ratio of the patterns at the *origin* and *destination* of the jump. It is close to a notion of rhythmic density and gives a hint on the tempo but has little to do with pulse or beat. Finally, a third parameter depends on the content of the symbolic units used to build the graph: it can be the octave if we extract and learn *pitch classes* patterns or pitch if we extract and learn *textural/spectral* patterns.

These three parameters are computed by default to automatically obtain smooth concatenations but their use is controllable by the supervisor of the software. He/she can choose to emphasize the importance of one or the other depending on the musical context and start *playing* OMax in this way.

3 Contexts Influence

To help adapting the computer based generation to its current musical context, we extended the notion of additional features and parameters presented in the previous paragraph, to a wider notion of contextual annotations of various kinds which may influence the choice of jumps and material through a selection and weighting system.

3.1 Energy

When extracting the current energy — or dynamic, in musical term — of the musician's playing, we can use that information not only to annotate the stream which is learnt, but also to influence the current discourse generated by the computer. Either with predefined scenarii like *follow the current dynamic* or *do the opposite of the current dynamic*, or under the control of the supervisor, we can musically lead the dynamic content by favoring jumps to musical material of desired energy. We thus compare the energy descriptor of the destination of the jump to the present (continuously updated) energy extracted from the real-time input, and weight the jumps according to the desired dynamic.

This mechanism does not ensure that the computer generation will exactly execute the musical choice but it will incline its content towards the material

with the closest quality attainable in the current knowledge of the machine; and this way it will use statistically more material of that quality. Even if the dynamics instruction is thus not strictly followed by computer, the permanent considering of the *external* energy — ie. from the acoustic musician(s) — and the comparison with the computers *internal* energy — ie. what the machine is going to play — mimics the permanent tradeoff a musician has to deal with when playing and especially when improvising with others. In that way, we make OMax perceptive to the *context* in which its discourse takes place.

3.2 Tempo and Phase

In the same direction, making OMax receptive to the rhythm and beat has been a strong appeal. Experiences showed us that analyzing rhythmic patterns with Factor Oracle and using it as the core mechanism for variations (see 2.1) does not ensure enough continuity (see 2.2) to build a musically coherent and apprehensible discourse — except with specific types of music.

However, using the work of E. W. Large on beat tracking with oscillators [7] and thanks to the implementation of L. Bonnasse-Gahot [4] and J. Chao we can represent the pulse with a running phase and a constantly updated tempo. These two values extracted from the audio can be used both to annotate the learnt material and to adapt the computer based generation along its playing. But giving the feeling of pulse requires a strongest mechanism than favoring jumps with the weighting system presented 3.1. Therefore we introduced simple constraints on the jump candidates to obtain a strict selection before rating the acceptable jumps. Essentially, we compare phase values and, using a threshold, we examine if the difference is small enough to avoid a break in the feeling of pulsation.

As explained for *energy* in 2.2 and 3.1, we also imagined two usage of the pulse annotation. Either we compare the phase value between the origin and the destination of the jump candidate, which will preserve the rhythm of the original material even when recombined by the computer. Or we can compare the phase of the destination of the jump candidate to the *current* phase extracted in real time. This second alternative will synchronize the computer generation to the external beat context, making OMax participate to the musicians pulse environment.

3.3 Guidance

From the weighting and simple constraint systems described for energy and pulse, we derived another way to give control to the supervisor on the musical content OMax generation. Indeed, the techniques used to adapt the dynamic to the current context can obviously be used to obtain arbitrary *nuance*. We also imagined different kinds of influence and constraints on the musical material. For example, when pitch description is available, we can either constrain or favor jumps leading to a chosen register of the instrument.

4 Supervisors Influence

Albeit we developed OMax as an improvisation tool, we noticed along the years very different usages. It goes broadly from the composer using it to transform and generate chosen musical material in search for new lines during his/her writing process, to the computer musician improvising freely on stage with one or several acoustic musicians. Even though each of these different usages may require specific developments or interfaces, we generally consider that giving more control over the software’s behavior to the user will make it more usable and *apropos*. In that sense we try to enrich the *playability* of OMax ie. the possibilities to manipulate the musical content of the softwares output.

4.1 Descriptions

Nowadays OMax is able to feed its Factor Oracle core with three types of symbolic data robust enough to ensure continuity (see 2.2); namely MIDI slices, notes — that is pitch, velocity and duration information — and spectrally coherent units as explained more thoroughly in [9]. Besides handling different kind of streams and/or instruments, the meaning underlying the symbolic knowledge model does make a perceptible musical difference when generating new phrases. Therefore we oriented our view on the improvisation core towards a *database* approach and enabled the query of one or several Factor Oracle (when available), corresponding to one or several descriptions of the input, to gather possible jumps, then filter, weight, order them and pick the *best* solutions.

This approach not only multiplies the variation possibilities, but also allows a musical choice through selecting the kind(s) of *listening* and variations (melodic, harmonic, timbral and combinations of these) OMax produces. For example, when playing with a saxophonist, both spectral and pitch descriptions may be simultaneously or alternatively relevant depending on the musical content and intent. Thus we developed a new unified architecture which lets the supervisor seamlessly combine or switch from one description to the other, without breaking the musical flow of the machine, in addition to control the jumps *rate* (2.2) and the smoothness of variations (3.1).

4.2 Regions

Following the unification process started with the accessibility to all the descriptions at once, we also generalized the *regions* system which permits to control the origin of the content used for the computer based generation in the history of the learnt material. Two regions *per description* were already available in the former versions of OMax. To augment their power and their versatility, we can now have as many regions as the supervisor needs, usable with any description and each of them can benefit from two *online* updating mechanisms — a *follow* option keeps a fixed-width window updated along the musician’s input while an *extend* option augments the width of the region to include the most recent material. *Avoiding* specific material is also a very musical way of choosing the

content of a musical discourse, so each region can as well be inverted to allow all the material *outside* the definite domain to be used.

From the long term experience with various musicians, we also noticed that alternating between material coming from the long term learning and elements from the very recent events added to the graph, is a pertinent process for introducing both *surprise* and *apropos* in the computer based generation. We generalized this process with an automatic switching module, which can alternate between any region (or no region at all), with a settable period and repartition over the period. Finally, these new regions can be combined as will, by taking their mathematical *union* or *intersection* to select several possible origin of the material recombined by the computer.

5 Applications

Along our research we built several prototypes of OMax to concretize and test different ideas and improvements around the software. We also regularly give concerts with these prototypes — when they are robust enough — ensuring that way the musical interest for the musicians and the audience.

5.1 ProtOMax

The last prototype we built implements all the functions we described in this paper with a very modular architecture in order to easily assemble different combinations of the new elements. Several constraints and weighting modules have been programmed for *energy*, *pitch*, *rhythm*, *pulse*... A new generic and versatile region module has been designed as well as the *follow*, *extend* and *region switch* modules to refine the possibilities of these regions. The diagram given Figure 1 summarizes the architecture of the new prototype. It is naturally also based on older modules especially for the learning part and it includes the graphical aspects and the visualization possibilities described in [9].

5.2 Concerts

Two recent concerts made use the new functionalities describe here, with two very different setups. Both concert were organized around a long formed jazz band playing mainly *pulsed* written music with composed parts and improvised solos or duets.

Nuit de l'Impro. The very first public performance with the new phase and tempo (3.2) possibilities of OMax took place during the night of the 12th October 2012 at the Conservatoire National Supérieur de Musique et de Danse de Paris thanks to Georges Bloch, head of sound department and his students. We had the opportunity to prepare rather long in advance this concert with Carine Bonnefoy's quintet (piano, trumpet, saxophone, double bass, drums) so

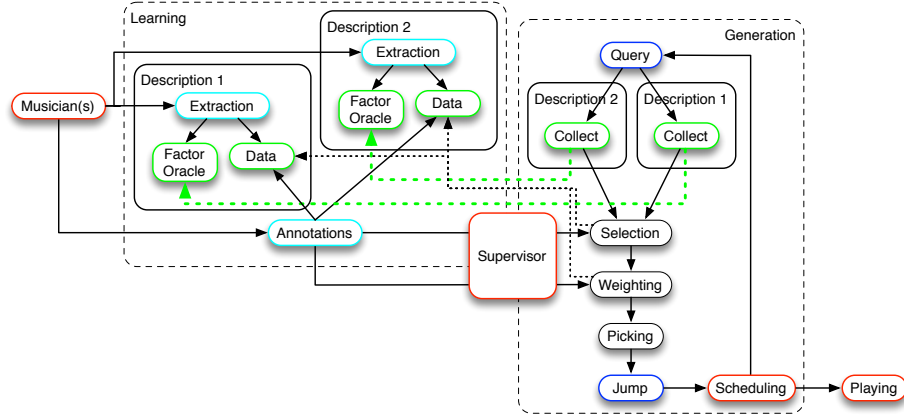


Fig. 1. Overall view on the architecture of the new OMax prototype

we recorded compositions from Carine’s in studio and gathered this way very precisely annotated and good quality recordings. During the public performance, OMax was loaded with this prepared material and didn’t learn on the fly. The challenge though, was to make OMax respect — for the first time — the very regular pulsation of a jazz band. Choruses with OMax solo and duets between OMax and a musician of the band were included in the compositions and OMax was prepared to forge improvised discourses mixing both the trumpet and the saxophone recordings, sounding like a hybrid instrument¹.

Compagnie Nine Spirit. More recently, the Compagnie Nine Spirit, long time contributors to OMax experiences, also helped us test the new prototype. In a bigger jazz configuration around the saxophonist Raphaël Imbert and his musicians (two guitars, two trombones, trumpet, double bass, drums and saxophone), OMax was used both in free improvisations and compositions from Raphaël in which the learning and the generation of the system may be planned. We gave two concerts with the band, May-June 2013 in Les Baux-de-Provence and Arles — in the framework of Marseille-Provence 2013, European Capital of Culture — where all the new annotations (especially *tempo* and *phase*) were learnt on the fly, on stage, and the new control possibilities were used to drive voluntarily the system and to automatically adapt and synchronize the computer based generation with the current musical context.

6 Conclusion & Future Directions

In this work we made OMax perceptive mainly to the dynamics and pulse aspects of its environment. We would like now to address the problem of harmonic

¹ a video footage can be found at www.dailymotion.com/video/xvcu1q
OMax’s interventions around 3:17, 6:55 and 9:00

elements. Some promising results came out of the research of L. Bonnasse-Gahot around SOMax — a corpus-based accompaniment variation of OMax — who uses a chromagram as information for harmonic context with great success. We may pursue this direction using the chromagram as annotation or core data.

In this work, we also gave the supervisor more control over the musical content of the computer base generation in terms of location, musical features and types of variations. This way he/she has more possibilities to *play* with OMax. However, for now, it is still up to the supervisor to give resolvable instructions and care about the consistency between weighting and constraints on the generation, the actual material learnt and the musical context. We need to explore errors and failure handling to obtain a more autonomous system. We have to consider cases of impossible constraints and automatic selection of annotations, this way we could entrust the supervisor with higher level musical decisions.

Acknowledgments. The work presented in this article greatly benefited from L. Bonnasse-Gahot’s work around SOMax in the framework of the SOR2 project, <http://sor2.ircam.fr>, funded by the French research agency, ANR. We would also like to thank Jérôme Nika and the RepMus team for their fruitful discussions and advice. Finally, thanks to Carine Bonnefoy & her musicians, Raphaël Imbert & his musicians and Olivier Corchia who helped patiently and actively for the testing of various prototypes and situations in studio and on stage.

References

1. C. Allauzen, M. Crochemore, and M. Raffinot. Factor oracle: a new structure for pattern matching. In *SOFSEM’99*, volume 1725 of *LNCS*, pages 291–306, Milovy, Czech Republic, Nov. 1999. Springer-Verlag.
2. G. Assayag and S. Dubnov. Using factor oracles for machine improvisation. *Soft Computing*, 8-9:604–610, 2004.
3. G. Assayag, G. Bloch, and M. Chemillier. Omax-ofon. In *Sound and Music Computing (SMC) 2006*, Marseille, France, May 2006.
4. L. Bonnasse-Gahot. Donner à omax le sens du rythme : vers un improvisation plus riche avec la machine. Technical report, École des Hautes Études en Sciences Sociales, 2010.
5. D. Conklin. Music generation from statistical models. In *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pages 30–35, 2003.
6. S. Dubnov, G. Assayag, O. Lartillot, and G. Bejerano. Using machine-learning methods for musical style modeling. *IEEE Computer*, 10-38:73–80, 2003.
7. E. W. Large and J. F. Kolen. Resonance and the perception of musical meter. *Connection Science*, 6(2-3):177–208, 1994.
8. A. Lefebvre, T. Lecroq, and J. Alexandre. Drastic improvements over repeats found with a factor oracle, 2002.
9. B. Lévy, G. Bloch, and G. Assayag. Omaxist dialectics : Capturing, visualizing and expanding improvisations. In *NIME*, pages 137–140, Ann Arbor, USA, Mai 2012.

Appendices C

Documentation of OMax Public Software

From 2010, a public version of the OMax software which benefits from the work presented in this thesis has been released and updated. This version is based on the architecture proposed in [chapter 9](#). However, it is limited to one Audio input with pitch and spectral description and one MIDI input with the virtual fundamental description of slices. The generation part has also been constrained to simplify its usage. For the Audio part, only four *improvisation logics* and five *players* are supplied. Each *improvisation logic* is associated with one *player*. An additional *player* with transposition and time-stretching possibilities is able to work with any *improvisation logic* and synchronize with one of the other four *players*. In the MIDI part, only two *improvisation logics* and three *players* are supplied.

We include in this appendix, the documentation of the public version of OMax that we distribute with the software. This documentation has been fully written from scratch during the PdD. It explains in a first part the principles of OMax and the software architecture. Then it thoroughly describes each module, its interface and usage.

O M a x

The Software Improviser



©Marin Lartigues

Version 4.5.x

OMax design and development by G. Assayag, G. Bloch, M. Chemillier, B. Levy with S. Dubnov
Documentation by B. Levy

© IRCAM, 2004-2012

Contents

1	Introduction	2
1.1	System Requirements	2
1.2	OMax Principles	2
1.2.1	Fundamentals	2
1.2.2	Style Modelling	3
1.2.3	Stylistic Reinjection	3
1.3	Quick Start	4
1.3.1	Audio	4
1.3.2	MIDI	6
1.4	Architecture	8
1.4.1	Overall view	8
1.4.2	Functions	8
1.4.3	Externals	9
2	Modules	10
2.1	The 2.5 worlds of OMax	11
2.2	Input(s)	12
2.2.1	MIDI	12
2.2.2	Audio	13
2.2.3	Pitch	14
2.2.4	Spectral	14
2.2.5	Common interfaces	15
2.3	Improvisation(s)	16
2.3.1	Core	16
2.3.2	Players	18
2.4	Visualisation	21

Chapter 1

Introduction

1.1 System Requirements

Hardware

OMax 4.5.1 has been developed and tested for Apple computers based on an Intel processor only. It has been extensively used on laptops of the Macbook Pro series, models going from Macbook Pro 3,2 (2007 with Core 2 Duo processor) to the most recent Unibody models.

Software

OMax 4.5.1 has been developed and used in Max/MSP 5 in its most recent (and last) version, 5.1.9. It has not been thoroughly tested on Max 6 yet, however no specific feature of Max/MSP 5 is used and positive feedback has been given on first tests. So it may work fine in Max 6 (without any guarantee).

1.2 OMax Principles

1.2.1 Fundamentals

OMax is an improvising software which uses on-the-fly learning and generating from a live source. It is capable of simultaneously *listening* to the audio stream of a musician, *extracting* information from that stream, *modeling* this information into a complex formal structure (think of a musical roadmap), then *navigating* this structure by recombining the musical material to create variation(s) or “clone(s)” of the input. OMax has no *a priori* knowledge or rules to guess or infer any specific analysis of the input based on a particular music theory (it is said *agnostic*). Thus its versatility and adaptability. OMax can either input and generate MIDI information or input and output an audio signal, in which case it uses a recording of the very musician’s sound, thus enforcing the “cloning” effect.

1.2.2 Style Modelling

“By Style Modelling, we imply building a computational representation of the musical surface that captures important stylistic features hidden in the way patterns of rhythm, melody, harmony and polyphonic relationships are interleaved and recombined in a redundant fashion.”. In other words, style modeling is the extraction of implicit rules from a musical material (often referred as the source). Whether the system infers the implicit rules of the source or mimics the surface without trying to guess these rules depends on the representation model used.

In this field, investigations have been done (in collaboration with Shlomo Dubnov) with dictionary based models such as LZ compression algorithm (Incremental Parsing (IP) methods) or Prediction Suffix Trees. Both methods build a tree encoding patterns occurrences and allows to calculate probabilities on the continuation of a given pattern.

In the years 2001-2002 the Music Representation team in IRCAM started to use a novel representation of pattern repetition, Factor Oracle, coming from the genetic research on patterns in genome. Since then, they studied and successfully used this representation for style modelling and music generation and built OMax, a real-time improvising system using stylistic reinjection.

1.2.3 Stylistic Reinjection

The musical hypothesis behind stylistic reinjection is that an improvising performer is informed continually by several sources, some of them involved in a complex feedback loop. The performer listens to his partners. He also listens to himself while he is playing, and the instantaneous judgment he bears upon what he is doing interferes with the initial planning in a way that could change the plan itself and open up new directions.

Sound images of his present performance and of those by other performers are memorized and drift back in memory from present to the past. From the long-term memory they also act as inspiring sources of material that would eventually be recombined to form new improvised patterns. Musical patterns are supposed not to be stored in memory as literal chains, but rather as compressed models, that may develop into similar but not identical sequences: this is one of the major issues behind the balance of recurrence and innovation that makes an interesting improvisation.

The idea of stylistic reinjection is to reify, using the computer as an external memory, this process of reinjecting musical figures from the past in a recombined fashion, providing an always similar but always innovative reconstruction of the past. To that extent, OMax, as a virtual partner will look familiar, as well as challenging, to his human partner. The interesting thing is to see how the human partner reacts to his “clone” and changes his improvisation accordingly.

1.3 Quick Start

1.3.1 Audio

1. Starting point to get OMax working (after checking the system requirements) is to **load the OMax.4.5.x.maxpat patch in Max 5**. This patch is highlighted in red in the OMax Patches&Objects folder. Or you can use the alias pointing to it present in the OMax4.5.x folder. It may take one or two minutes as the patch embeds a lot of others. It should print about 35 lines in your Max window with the credits for external objects and OMax.data, OMax.oracle and OMax.buffers declarations. However, none of them should be red (error message). Next steps will make reference to the capture presented Figure 1.1.
2. If you want to use OMax on a live input, **adjust the channels** (accordingly to your hardware and Max I/O mappings) then **check the Input toggle** (frames 2, top left corner of the patch).
NB: The first channel (named rec) is used to record the sound in a buffer (and replayed later on) while the second channel (named dtct) is fed into the detection (analysis) system. These two channel can receive the same stream if you use only one microphone.
If you want to use OMax on a sound file, click on the open message of the Play file box and choose an aiff file in the navigator popping up. Checking and unchecking the toggle (frame 2b, top left corner of the box) will start and stop the playing of your file once the audio is turned on.
3. Provided that you had no errors in the Max window when you loaded the patch, you can now **turn on the audio** with the “speaker” button (frame 3, bottom right side of the patch).
4. You can **check and adjust the level** of the sound arriving to OMax with the horizontal vumeters (and their overlaid sliders) in and on the right of the Record box (frame 4). You can also **monitor** this stream by checking the top left toggle of the Monitor box (frame 4b) — you may need to adjust the channels of your monitoring system with the numbers in the same box.
The Pitch and Spectral boxes **LEDs should start flashing** (frame 4) as the sound gets through. It indicates that they are detecting some activity and able to extract information from the audio. Otherwise you need to continue adjusting levels and/or detection parameters (see 2.2.3 or 2.2.4).
5. When you are ready, you can **start recording and learning** by checking the big toggle labelled Start Learning on the top of the patch (frame 5).
If you work with a sound file, you can stop and start reading the file again from the beginning with the toggle of the Play file box (frame 2b).
Model should start being built, Size and Hi Ctxt of the Dual_MIDI and Dual_SP boxes should start growing.
6. You can **activate the visualization** of the knowledge by checking the top toggle of the Visu box (frame 6) and looking in the Jitter window. At least an horizontal black line should appear and hopefully some white and grey arcs below and above this timeline proving that OMax is recognizing some interesting patterns.
7. OMax is now ready to improvise! **Check** the top left toggle of **one of the players and it should start playing** (frames 7). You can adjust the channel and volume of each of the players in the Output box right below.
8. Hit the Stop! button (frame 8, on the top of the patch) to **stop all the players at once**. And **reset** the Audio part of OMax **with the Reset button** underneath the Start Learning toggle (frame 8, on the top of the patch) and to be ready to start again a new improvisation.

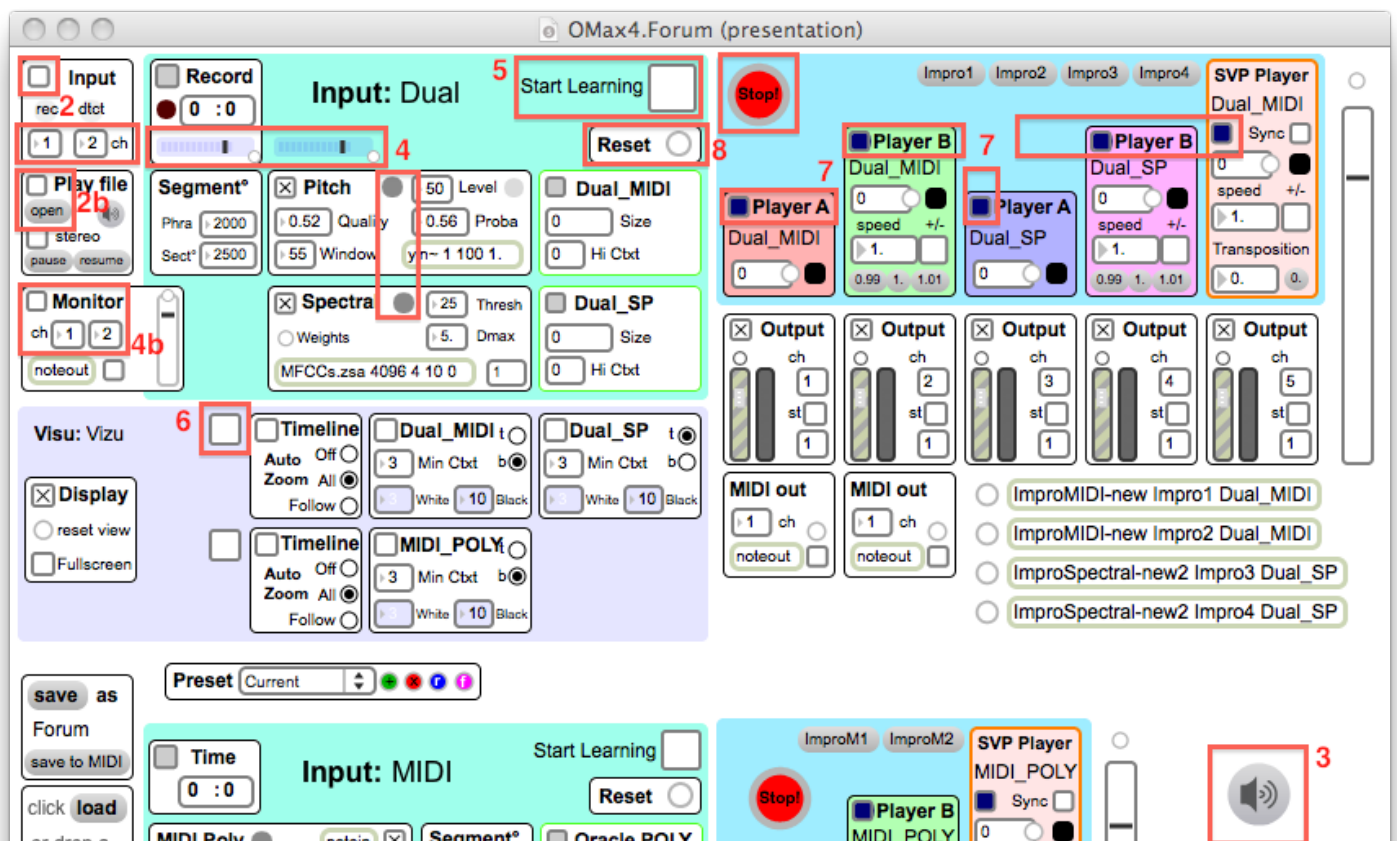


Figure 1.1: Audio Quick Start

1.3.2 MIDI

1. Starting point to get OMax working (after checking the system requirements) is to **load the OMax.Forum patch in Max 5**. This patch is highlighted in red in the OMax4.5 folder. It may take one or two minutes as the patch embeds a lot of others. It should print about 35 lines in your Max window with the credits for external objects and OMax.data, OMax.oracle and OMax.buffers declarations. However, none of them should be red (error message). Next steps will make reference to the capture presented Figure 1.2.
2. First you need to **define a MIDI input** to receive data from by double-click on the notein object in the MIDI Poly box (frame 2). You can also adjust the channel you want to listen to with the number box right below (frame 2). A value of -1 means *all channels*.
NB: if you want to play a MIDI file and feed it to OMax, you need to use an external software as [Sweet MIDI Player](#) to play and route the MIDI stream to Max/MSP.
The **LED** of the MIDI Poly box should start flashing as soon as you receive MIDI data.
3. You are now set to **start the learning** of OMax by checking the toggle labelled Start Learning (frame 3, on the top of the MIDI part of the patch).
4. You can **activate the visualization** of the knowledge by checking the bottom toggle of the Visu box (frame 4) and looking in the Jitter window. At least an horizontal black line should appear and hopefully some white and grey arcs below and above this timeline proving that OMax is recognizing some interesting patterns.
5. Before making OMax improvise, you need to **adjust the MIDI output of the players**. Double-click on the noteout box of each of them (frame 5) to decide if you want to use the default synthesizer of the Apple Operating System, *AU DLS Synth* or if you want to send the MIDI stream to another synthesizer you may have on your computer.
6. OMax is now ready to improvise! **Check** the top left toggle of **one of the players and it should start playing** (frames 6).
7. Hit the Stop! button (frame 7, near the players) to **stop all the players at once**. And **reset** the MIDI part of OMax **with the Reset button** on the left of the Stop! button (frame 7) to be ready to start again a new improvisation.

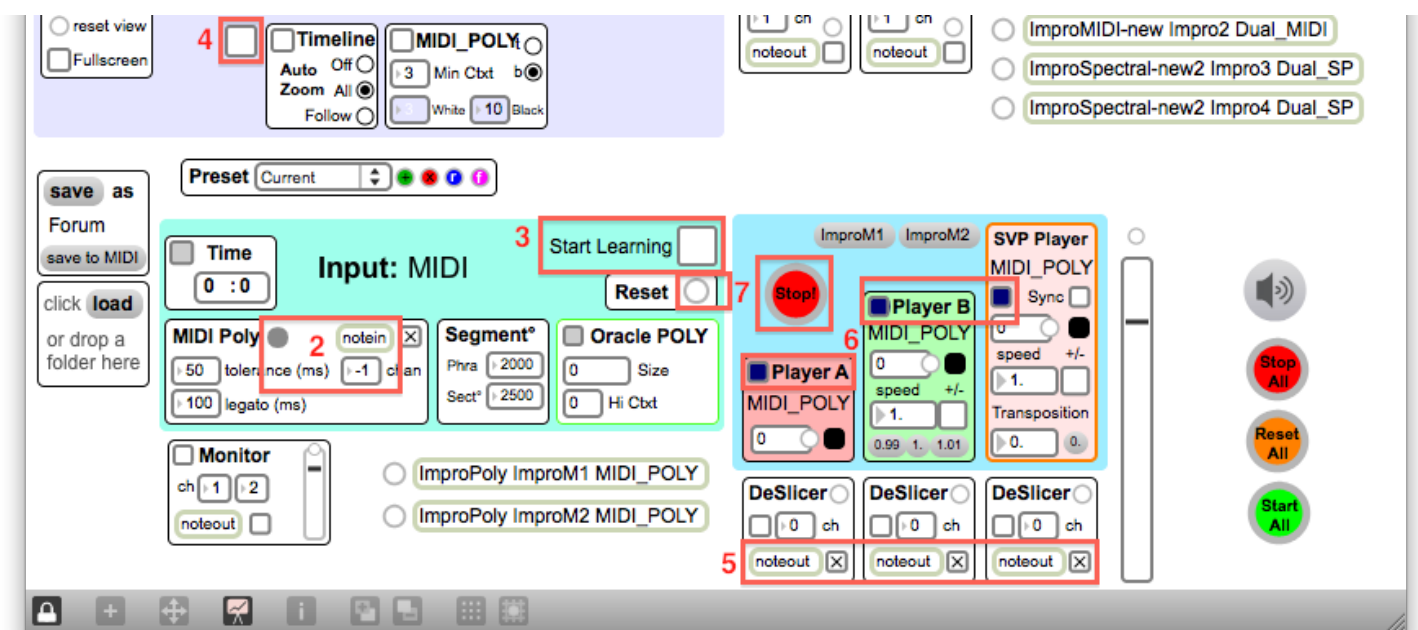


Figure 1.2: MIDI Quick Start

1.4 Architecture

1.4.1 Overall view

From a very distant point of view, OMax is built in two parts. A first ensemble is in charge of the listening and learning process and a second ensemble is the improvising part itself. The analysis is conceptually divided in three stages. A first stage extracts from the incoming stream some abstract description based on signal processing principles (which will be described later in this documentation). Once this **detection** has been done, a **segmentation** stage takes the abstract stream (of numbers mostly) from the detection to aggregate them into consistent units that have a symbolic coherence and a defined value.

For example, in the common case of pitch information, the detection stage will be an algorithm for fundamental frequency extraction. It will output a continuous stream of frequencies that need to be filtered and grouped by the segmentation to give a symbolic information of note defined by its onset, offset and pitch value (possibly given also with a velocity value).

The result of these two steps of analysis is a symbolic representation of the input stream that can be learnt in the pattern recognition model of OMax. This **modelling** is done by constructing the Factor Oracle graph previously mentioned.

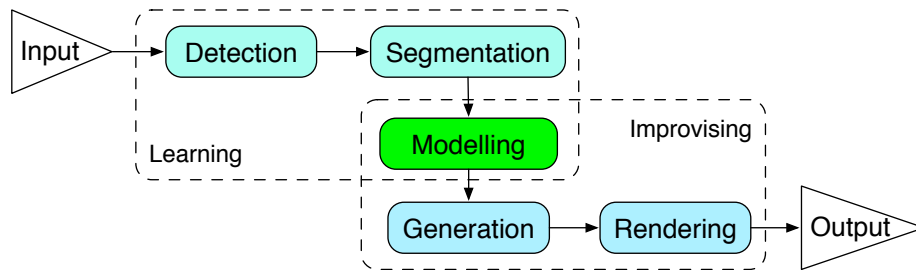


Figure 1.3: Overall architecture of OMax

Once the building of the model has started, the improvising part of OMax is able to navigate in this model to create a virtual improviser (often referred as a “clone”) which plays variations on the learnt sequence. A first block reads the model and generates a path for an improvisation. This **generation** block sends (usually along the navigation) the informations about this path to the **rendering** block which is in charge of retrieving the effective recorded elements (typically the audio chunks) to read and produce the sound of the “clone”.

1.4.2 Functions

In a more detailed view of OMax, we can describe the different parts previously presented with more blocks. As suggested, there is naturally a recording part in the learning ensemble of OMax composed of a **buffer** and a **record** agent. The rendering part of the improvising ensemble can also be divided in two different elements : the **scheduler** is in charged of putting the path (in the learnt knowledge) *in time* then the **renderer** read the recorded buffer to get the information to output the actual sound material.

The visualisation part is one of the main novelty in OMax 4.x (compared to the previous versions). It allows the user to see in real-time on the **interface** the current state of the model and interact with the improvising part, with regards to this state.

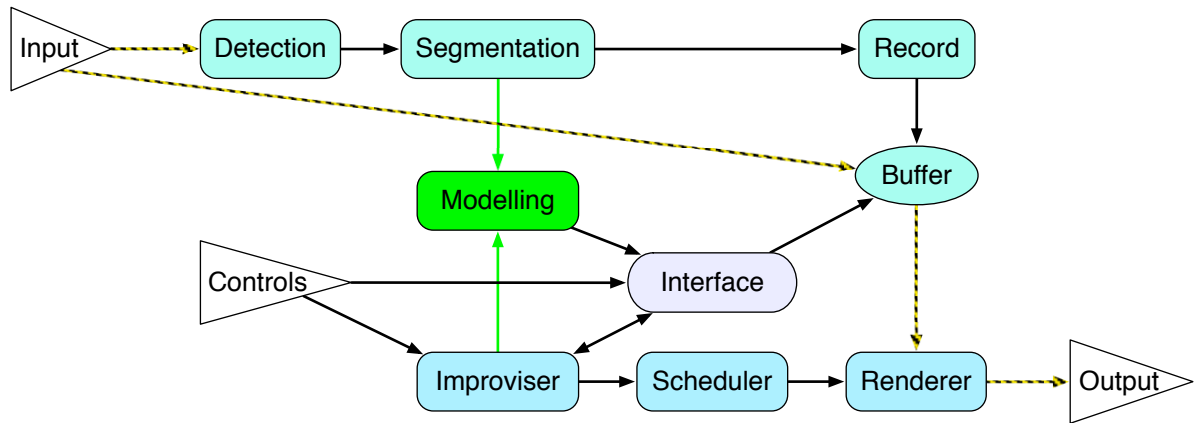


Figure 1.4: Functions in OMax

1.4.3 Externals

The second main novelty in OMax 4.x is the integration of OMax in the Max/MSP environment solely. On the diagram 1.4, with respect to Max/MSP conventions, black links represent message connection while yellow and black arrows are audio connections. The green links show the connections with the knowledge model which required specific external objects for Max/MSP developed in C/C++ especially for OMax. A collection of 9 external objects for Max/MSP is provided with OMax. They are all named

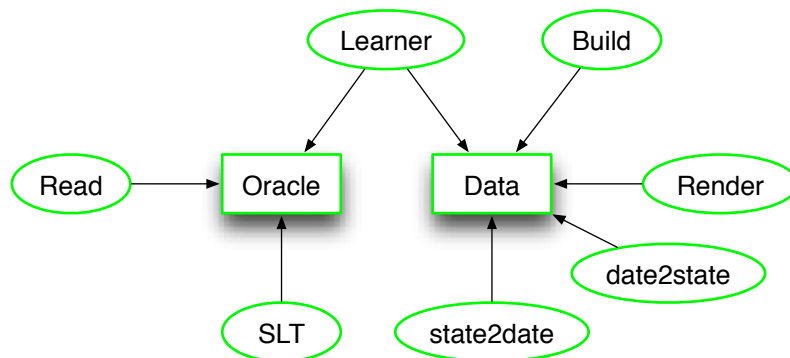


Figure 1.5: External objects for OMax 4.x

with the prefix *OMax*. They implement two data structures: one (**OMax.oracle**) is the Factor Oracle graph structure, the other one (**OMax.data**) holds the musical (symbolic) content of the input sequence. The other objects are used to interact with these structures, read them, write them and use their content.

Chapter 2

Modules

The third novelty of OMax 4.5.1 is the entirely modular structure of the software. Even though OMax is opened through a single and main patch in Max/MSP, each part of the software is embedded in its own abstraction that can be reused in many other configurations. Thus it is possible to construct with these elements an OMax patch suitable for specific needs or even reuse some part of OMax to build your own system. The modules of OMax are closely following the functions previously presented. Here is the diagram of the actual modules of OMax.

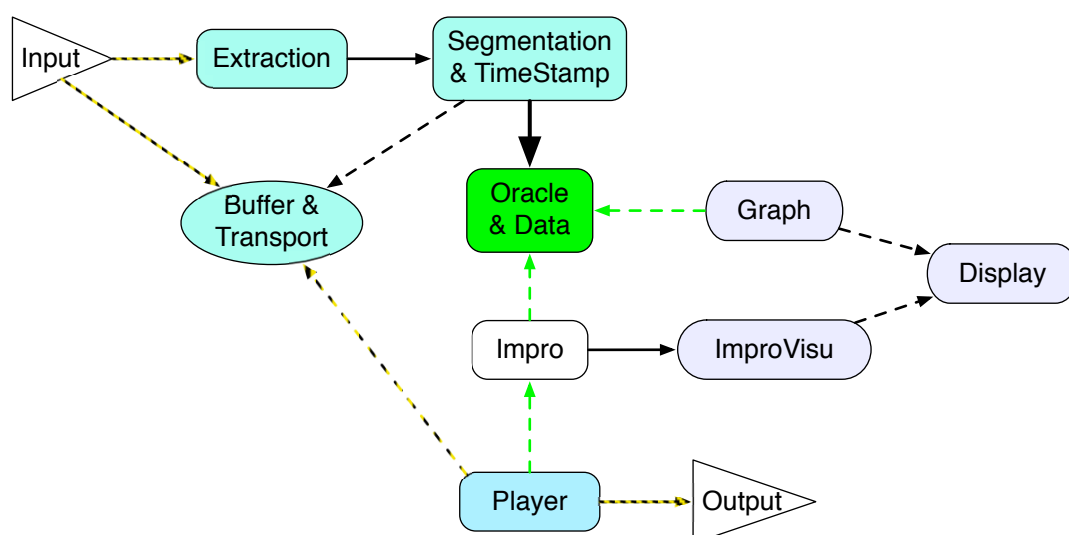


Figure 2.1: Modules of OMax

The **Extraction** module is in charge of computing the detection information on the input stream. It passes the result to the **Segmentation & TimeStamp** module which does the gathering of consistent and sequential description data to constitute the symbolic units and dates them with respect to the recording time. Timestamping is the link between the symbolic units describing the input and the actual recording of the stream made in the **Buffer & Transport** module.

Then, the symbolic information is organised and stored in the **Oracle & Data** module which is thus the core of OMax knowledge. The **Graph** module periodically reads the model and gives a visualisation of its state which will be displayed by the **Display** module thanks to Jitter.

The **Impro** module navigates in the knowledge model with respect to the parameters and constraints given through the interface and computes a new path to generate a “clone”. This path is graphed on the

display thanks to the **ImproVisu** module. Then the **Player** modules reads this path and with the actual recording recreates a sounding virtual improviser.

2.1 The two and a half worlds of OMax

OMax 4.x allows two types of input stream of two very different nature: **MIDI** data coming from a keyboard or any other MIDI instrument or **audio** coming from a microphone or the reading of a sound file. These two worlds are totally separated in OMax: the bottom half handles the MIDI part (Fig 2.2) while the top half of the main patch handles the audio part of OMax (Fig 2.3).

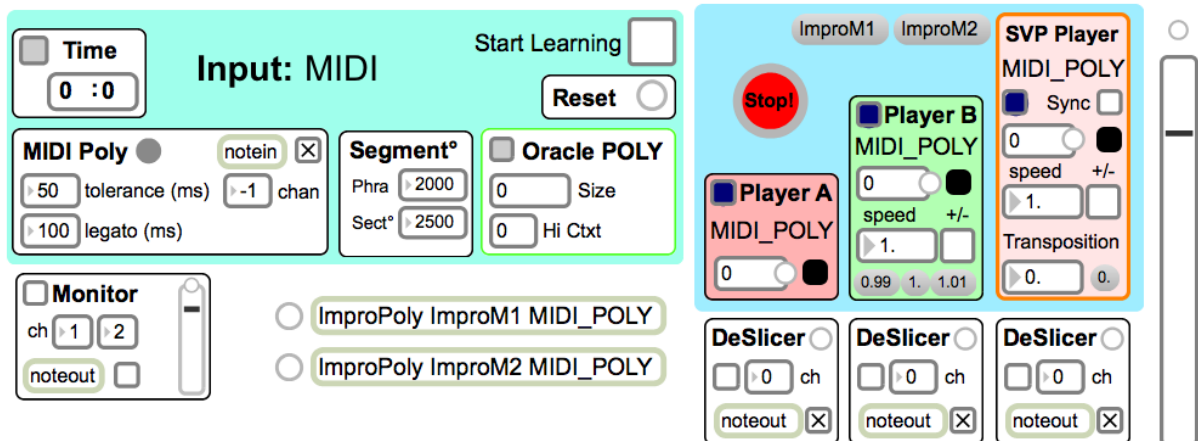


Figure 2.2: MIDI part of OMax

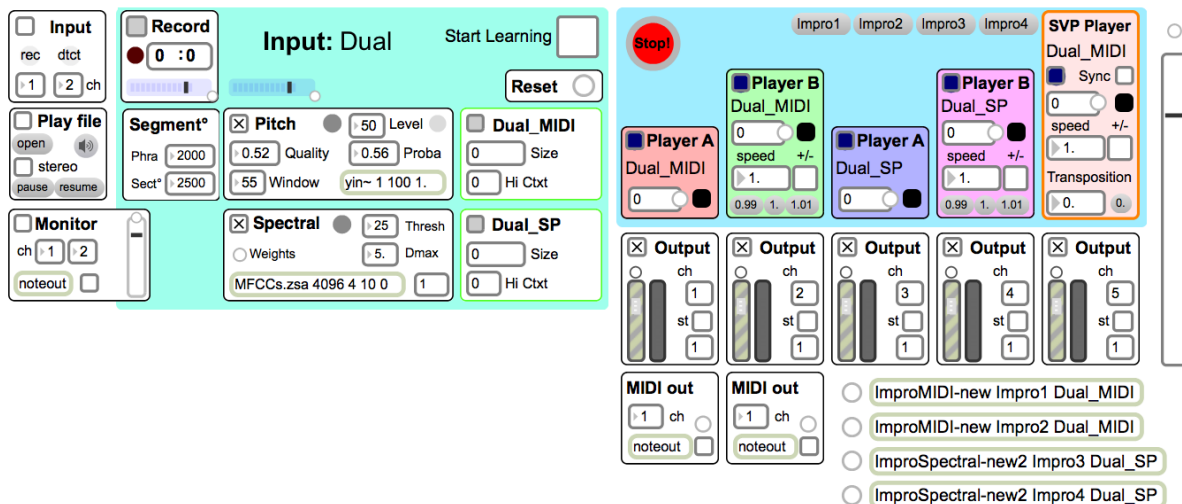


Figure 2.3: Audio part of OMax

Each one of these domain possess its own analysis module(s), data structures and improvisation parts. They share the same conceptual organization presented 1.2 and have the same kind of modules but no link. Only the Jitter display is mutualized for obvious performance and screen reasons.

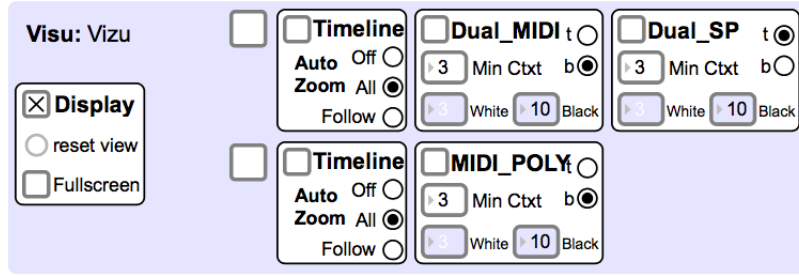


Figure 2.4: Visualization part of OMax

2.2 Input(s)

As previously mentioned, there are two types of inputs in OMax and three different analysis possible. We will present here how to use the modules concerning each one of them.

2.2.1 MIDI

The MIDI input of OMax is contained the part of the patch presented Figure 2.5. The parameters to get and analyze a MIDI stream are contained in the section framed in red.

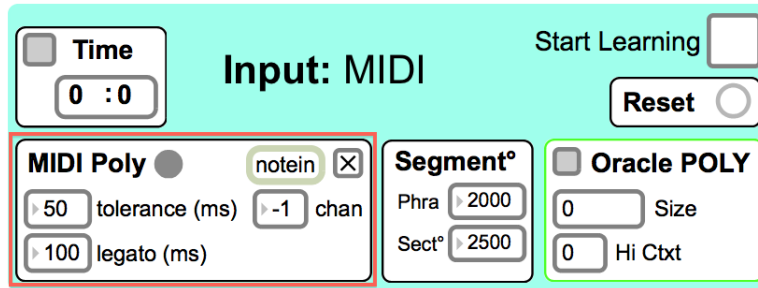


Figure 2.5: MIDI input section of OMax

notein ☒ : the notein box and its associated toggle is used to select and enable the receiving of MIDI data. Double click on notein to chose the MIDI port.

chan : the number labelled chan is used to select a specific MIDI channel to listen to. If the value -1 is put, the data from all the 16 channels of the port will be received.

tolerance : the tolerance parameter defines the time granularity (in milliseconds) to separate two MIDI events. Once a note has arrived, all the other notes arriving within the tolerance time will be considered as simultaneous and constituting a common chord. A note arriving after that time will be considered as a new event (possibly a new chord).

legato : the legato time (in millisecond) defines the separation between two notes (or chords) that are overlapping. If the overlapping lasts more than the legato time, then the resulting chord is considered as a meaningful event, otherwise, the overlapping is ignored and only the two notes (or chord) before and after are considered.

LED : a small grey LED (flashing green) indicates on the right of MIDI Poly if MIDI data arrives in the module.

2.2.2 Audio

In the current version of OMax, the analysis being done on an audio stream can rely on two different extractions: **pitch** content or **spectral** content. These two analysis run in parallel on the same audio stream so they share the same buffer and timeline but feed to different graphs/knowledge (Factor Oracle) to play with. The audio input of OMax can come from both a real-time stream of one or two microphones or the playing of a sound file. The modules to adjust the audio input are common for both the pitch or the spectral analysis.

Input: if you have one or two microphones plugged on your computer, you can activate the “live input” patch by ticking the toggle on the top left corner and use them. Once you have turned on the DSP of Max/MSP, the two flashing green LEDs named **rec** (for recording) and **dtct** (for detection) will tell you if it gets any signal. You can adjust the channel for the recording and the detection separately with the two corresponding number boxes labelled **ch** (for channel).

The detection channel goes into the different extraction and segmentation stages while the recording channel is recorded into the buffer as explained in section 1.4.2. It is sometimes better to use two different microphones for those two processes. For example a close field mic, less sensible to the environment (if there are several musicians for instance) may be more efficient for detection while a farther mic will give more bass and air for a better quality recording.

Play file: if you want to use a sound file as audio input for OMax, you can use the small player embedded. Start by choosing the file with the **open** button. Then, once the DSP of Max/MSP is on, checking and unchecking the **toggle** (top left corner of the **Play file** frame) starts and stops the playing. You can also **pause** and **resume** the playing with the buttons on the bottom of the frame.

The **stereo** toggle lets you choose the routing of the channels if you file is stereo. When uncheck (default), both channels of your file are summed and fed identically into both the detection and recording channels. If you check the **stereo** toggle, the first channel (left) of your file is fed into the recording while the second (right) is routed to the detection.

Two horizontal **VUMeters** framed in red on figure 2.6 are overlaid with two **sliders** which allows you to adjust the volume of both detection and recording channel. The small **bang** button at the bottom right corner of each of them is use to come back to the default (127) value. In the **Record** module, beside seeing and adjusting the recording volume, the red LED let you know when OMax is actually recording and the **numbers** (refreshed four times per second) indicates the present duration of the recording.

Lastly, the **Monitor** modules allows you to listen to the audio fed to OMax. You can chose the channel you want to monitor on with the two (one for the recording channel, one for the detection channel) number boxes labelled **ch** and adjust the volume with the vertical **slider** on the right (the **bang** button on top brings back the slider to the default value).

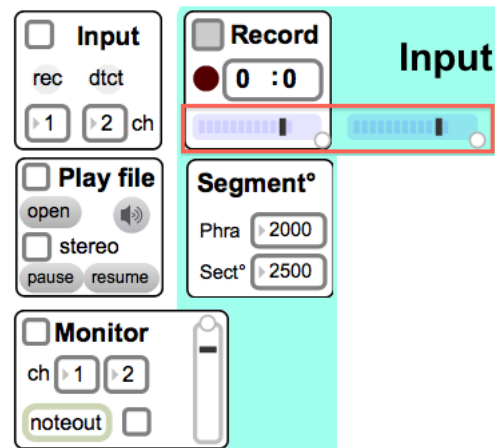


Figure 2.6: Audio Input of OMax

2.2.3 Pitch

Pitch extraction and segmentation are done in the module framed in red on figure 2.7.

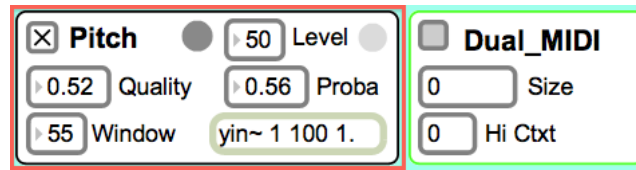


Figure 2.7: Pitch detection in OMax

Level : to be able to adjust separately the volume of the signal used in the pitch and the spectral detection, you can use the number box labelled Level.

Quality : the `yin~` object outputs along with the pitch stream a quality stream corresponding roughly to the saliency of harmonic frequencies. It is used in OMax as a *relevance* or *consistency* threshold to select dominant pitches. Pitches with a quality under the value given in the number box will be ignored.

Window and Proba : a statistical mechanism is included in the pitch detection module to improve the relevance of extracted information. Raw pitches are gathered during a certain time window (in milliseconds) and the so *cooked* pitch is output only if its probability over this window is above the Proba parameter (between 0. and 1.).

2.2.4 Spectral

Spectral detection in OMax relies on descriptors named MFCCs for Mel-frequency cepstral coefficients. If you do not want to use presets supplied with the patch, you can adjust manually the parameters shown in the red frame of figure 2.8.

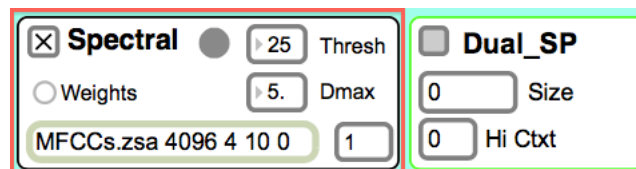


Figure 2.8: Spectral detection in OMax

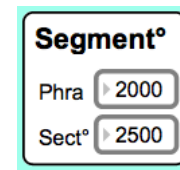
Thresh : the first coefficient of MFCCs represent the overall energy of the spectrum. As with quality in pitch, we can exclude spectral slices of too weak energy (as background noises for example) with the Thresh number box. The higher you put this threshold, the less sensible the detection will be.

Dmax : an adaptive clustering is implemented in this detection to recognize similar pitches. It uses a distance to compare the spectral content of the frames. The Dmax parameter adjusts the maximal distance for two frames to be considered as similar. Practically, it means that if you decreasing Dmax you gives the detection a finer resolution (differentiation) but may drastically decrease the quality of recognized patterns in the model as well. Increasing Dmax will agglomerate more different spectral contours as one type of spectrum which may augment the number of recognized patterns in the model but lower their relevancy.

2.2.5 Common interfaces

Silence segmentation

For both MIDI and audio input, OMax also performs a two level higher scale segmentation based on silences. The input stream is cut into phrases and sections based on the durations (in milliseconds) given in the number boxes labelled **Phra** and **Sect°**.

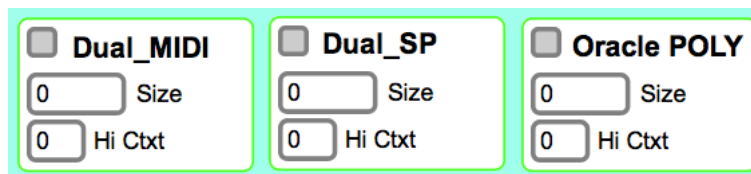


The interface is titled "Segment°". It contains two sliders: "Phra" with a value of 2000 and "Sect°" with a value of 2500. Both sliders have a small triangle icon to the left of the value box.

Figure 2.9: Silence segmentation

Current state of the model

Lastly, each detection gives you a feedback (presented figure 2.10) on the current state of the model once you have started the learning (see 1.3 to learn how to start). **Size** indicates the number of states in the Factor Oracle graph ie. the number of elements in OMax knowledge model of the input sequence. And **Hi Ctxt** (for highest context) displays the length of the longest repeated pattern recognized in the model. This gives you a clue on the current state of the learning and also some information on the adjustment of the detection. Typical values for highest context should be above 3 and not higher than 15 except if the source itself has some long repetitions.



The interface is divided into three sections: "Dual_MIDI", "Dual_SP", and "Oracle POLY". Each section contains a checkbox, a "Size" slider, and a "Hi Ctxt" slider. All checkboxes are unchecked, and all sliders are set to 0.

Figure 2.10: Current state of the Factor Oracle

2.3 Improvisation(s)

The improvisation part of OMax strongly relies on the Factor Oracle model used to analyze the source material (see 1.2). Variations or “clones” played by OMax are created by navigating this model and jumping from time to time to diverge from the original material. While the principle of this navigation are common to the different inputs and analysis done in OMax (see 2.1) some specialization are needed to adapt the result to the different material and gain in smoothness and musicality.

2.3.1 Core

In OMax 4.x the navigation strategy follows the functional diagram presented opposite (figure 2.11).

The original sequence is read for a few consecutive elements (ie a few notes or a short duration) that we name **continuity** then, it is time to diverge from the original sequence and the system decides to **jump**.

OMax has a small anticipation system to be able to find a good transition (variation) so it searches and **collects** the different solutions over a short **searching window**.

It **selects** next all these solutions based on different criteria both automatic (as a **taboo** system to avoid strict loops) and user based (as **regions** which allow to choose the part of the past serving as source). And discards solutions not following these rules.

Remaining solutions are **described** musically with parameters which strongly depends on the type of analysis and each solution is **weighted** with a mixture of these parameters. So the pool of solutions is ordered according to this weight.

Finally, one of the solution is **drawn** (among the best), validated and planned as the next jump. The destination of this jump starts a new continuous sequence and buckles the cycle.

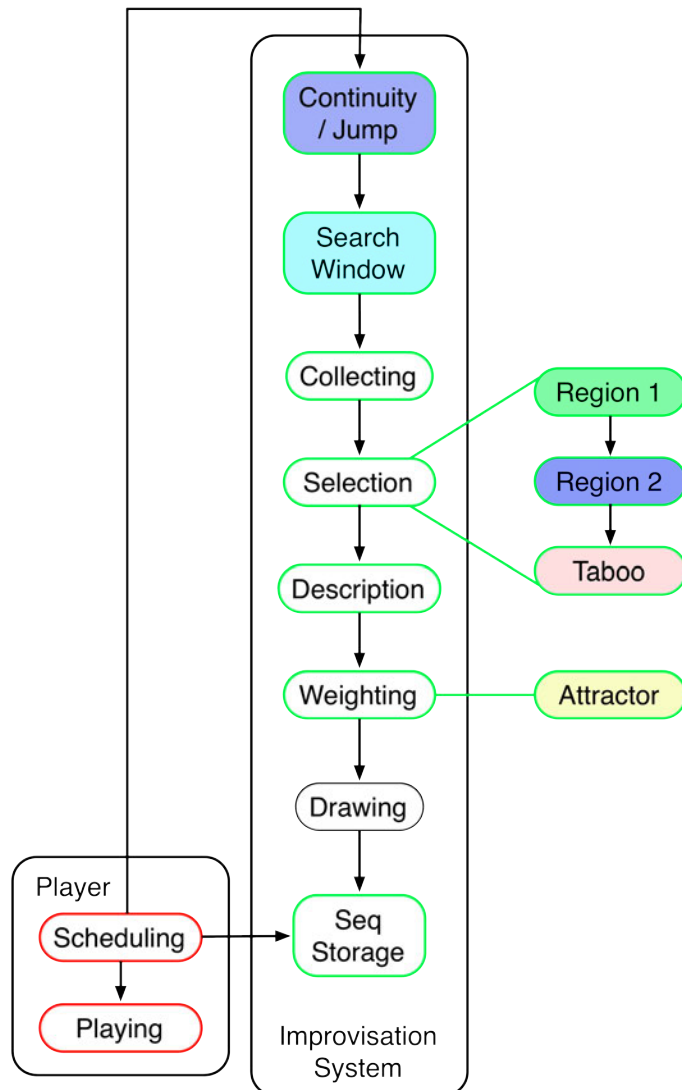


Figure 2.11: Improvisation & Player System

Interface for this common mechanism is presented figure 2.13. There are six of these improvisation core in the current version of OMax. Four of them are attached to the audio input (Fig 2.12a), two of them deal with MIDI data (Fig 2.12b). You can open and interact with them by double-clicking on their box (showed figure 2.12).

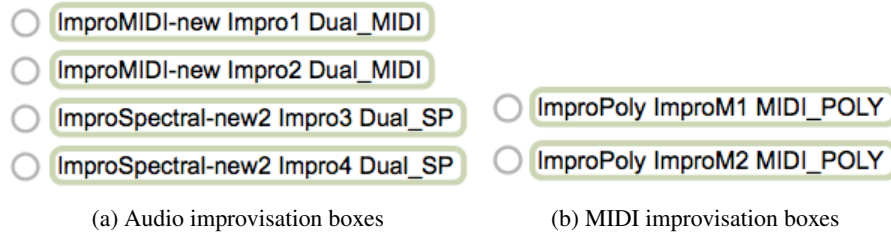


Figure 2.12: Improvisation cores boxes

Each one of these core is reading and improvising on one type of analysis only, indicated by the second argument given to these boxes. Dual_MIDI stands for pitch description, Dual_SP stands for spectral description (Dual being the name of the audio input) and MIDI_POLY is the model of the MIDI input.

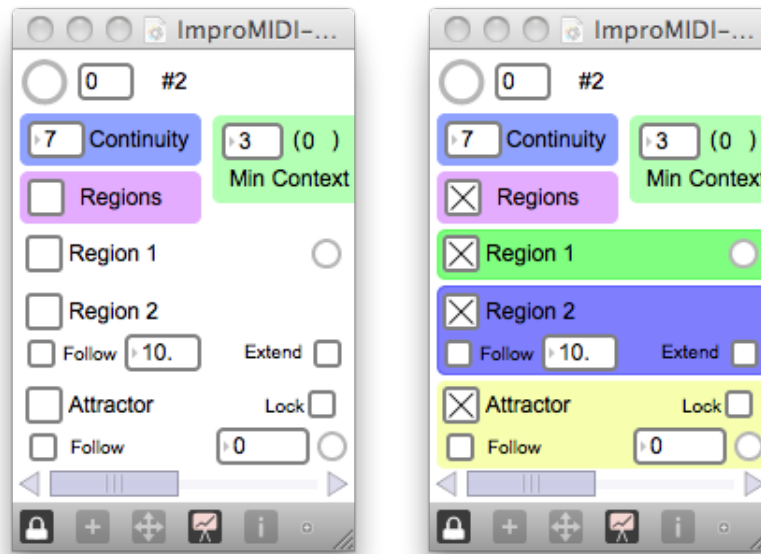


Figure 2.13: Common Improvisation Interface

Continuity : acts as a kind of *variation rate*. The higher, the longer consecutive chunks will be (so the lesser OMax will diverge from the original material).

⚠ Typical value for continuity strongly depends on the type of analysis.

Min Context : acts as a kind of *smoothness* parameter. The higher, the smoother divergences from the original material will be. However, forcing higher contexts makes also possibilities of jumps rarer. In brackets is recalled the highest context currently found in the model.

⚠ Putting a higher value than the one in brackets will stop OMax to vary from the original.

Regions : this toggle globally activate or deactivate the region mechanism. Defining the regions is done either directly on the visualization (see 2.4 §2) or automatically with the follow or extend mode in the case of region 2 (see region 2 below).

Region 1 : enables and disables the first region shown in green on the visualization.

Button : the bang on the right of Region 1 line sends the reference of the middle of green region to the attractor.

Region 2 : enables and disables the second region appearing in blue on the visualization.

Follow : besides selecting manually (see 2.20) the region 2, you can have it automatically set to the last x seconds – x being set by the number box next to the Follow word – by ticking the Follow box.

Extend : like the Follow (but exclusively with it), right bound of region 2 can be set automatically to the most recent event learnt in OMax while the left bound is set by the user. This allow to have a second region extending towards the present but limited in the past.

Attractor : aside from regions that are strict constrains, an attractor can be placed on the timeline (see 2.20) to help OMax “clone” to vary around a specific element of the past. The main toggle (next to the word Attractor) activate and deactivate this effect. The number box allows to place it. △ Range for the attractor goes from 0 to the most recent event learnt which depends of course from the input and the analysis.

Follow : automatically puts (and updates) the Attractor to the most recent event learnt.

Lock : is useful when the attractor position has been set through Region 1. If you activate the Attractor with Lock ticked, the attractor will help the “clone” to get inside the green region. As soon as it is inside, the Attractor will be replaced by the Region 1. The effect is a smoother (but longer) transition of the “clone” to get into Region 1 thanks to the Attractor.

2.3.2 Players

Core patches presented in the previous paragraph (2.3.1) generates improvised sequence however they do not play them. Players, located in the right half of the main OMax patch are in charge of effectively playing the “clones”.

There are five players for the audio part of OMax (Figure 2.14a) and three for the MIDI part (Figure 2.14b). There are also three types of players in OMax: players of **type A**, players of **type B** and **SVP** players.

In the default configuration of OMax, each player, except SVP players, is reading (and polling) a different improvisation core. For example in the audio part, the first player on the left (red) is reading the first improvisation core (named Impro1 and navigating the pitch description, Dual_MIDI, of the input), the second player (green) reads Impro2, the third (blue) reads Impro3 (navigating on Dual_SP, the spectral description of the input) and the fourth (magenta) reads Impro4 core. The players color matches the color of the core being read.

The same thing is also true for the MIDI part: the first player reads the first improvisation core (named ImproM1, in red) and the second player reads the ImproM2 improvisation core (in green).

SVP players are versatile and can read indiscriminately any improvisation core (of the same input). You can switch from one core to another with the message on the top labelled with the name of the cores: *Impro1*, *Impro2*, *Impro3*, *Impro4* for the audio part and *ImproM1* and *ImproM2* for the MIDI part.

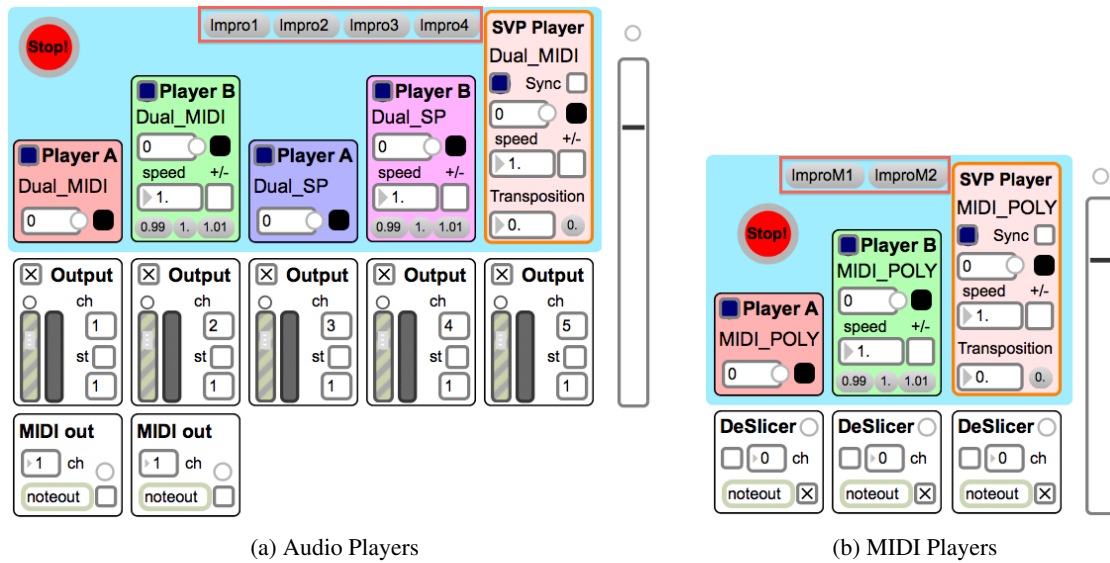


Figure 2.14: Players

Player A

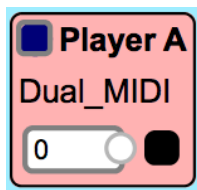


Figure 2.15: Player A

Player B

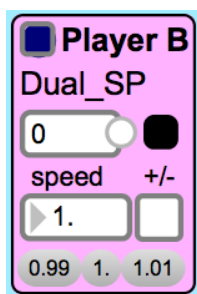


Figure 2.16: Player B

Players of type A are the simplest player possible. They start and stop with the **toggle** in their top left corner (on a dark blue background). The background of this toggle flashes (with a light blue color) whenever a variation (from the learnt input) occurs. The **number** indicates how many states (notes, chords or spectral slices) has already been read. And the **black button** on the right of the number box lets the player find a smooth ending then stop there.

Players of type B works the same way as players of type A: start and stop **toggle** is in the top left corner on a blue background, flashing when a variation occurs. The **number** indicates the size of the sequence already played and the **black button** triggers a smooth ending.

Besides these basic functions, players of type B provides a variable speed option. The **floating-point speed box** is a multiplicative factor going from 5. to -5. Normal speed is 1., 2. means two times faster, 0.5 half the normal speed (ie. two times slower) and so on. Negative speeds automatically ticks the **toggle** box labelled +/- and means reading the sound backward. However these changes of speed are link with a “natural” transposition exactly as when rotating old vinyls faster or slower.

SVP Player

SVP players make use of the Super Phase Vocoder developed at IRCAM and allows on top of the regular functions of the other players to transpose and/or change the speed independently. Start/stop **toggle**, smooth ending **button** and speed **box** work exactly as in Players of type B. Another **floating-point number box** allows to transpose finely the whole clone from 18 semi-tones below to 18 semi-tones above the original tuning (a value of 1. meaning one semi-tone above).

A SVP player can also be synchronized on another player (of type A or B) and thus used to harmonize a clone with a transposition for example. You need first to choose on which of the other players you want to synchronize with the buttons labelled *Impr*1...4 (or *Impr*M1...2 in the case of MIDI section) framed in red on Figure 2.14. Then you can tick the *Sync* **toggle** to synchronize the SVP player. You can check that they are indeed well synchronized by watching the **number box** showing the state being read.

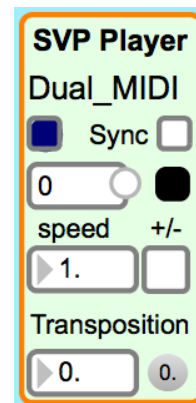
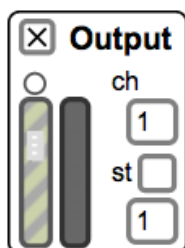
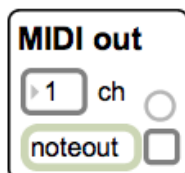


Figure 2.17: Player SVP

Output



(a) Audio Output



(b) MIDI Output

Figure 2.18

Below each of the players, you will find a small audio output patch which will let you adjust the volume with the left-hand side **fader**. The **view meter** lets you know the loudness of the clone (before applying the volume of the fader). You can choose the channel (*ch*) with the top **number box** on the right-hand side and decide to duplicate the output to another channel by ticking the stereo (*st*) **toggle** and addressing another channel with the second **number box**. You may also deactivate the whole audio output of the clone with the top left **toggle**.

This is useful in the case of a “clone” on the pitch description. Indeed this description of the audio input is able to emulate a monophonic MIDI stream. You can then play the “clones” through MIDI with the patch presented Figure 2.18b. Enable the MIDI playback by ticking the **toggle** next to the noteout box of the *MIDI out* patch. Choose the channel (*ch*) with the **number box** and the MIDI destination by double clicking on the noteout box. Use the **button** on the right to flush all MIDI notes.

In the case of the MIDI part of OMax, the output is named *DeSlicer*. It lets you choose the destination of the MIDI stream by double clicking on the noteout box which may be deactivated with the **toggle** on its right. By default, the output of MIDI OMax uses the same channel(s) as the input stream it learnt but you can override this by choosing a channel with the **number box** labelled *ch* and ticking the **toggle** on its left. The **button** in the top right corner flushes (ie. stops) all the MIDI notes on every channel.



Figure 2.19

2.4 Visualisation

Version 4.x has added a novel visualization to OMax. The state of the internal knowledge (Factor Oracle) can be represented in real time thanks to Jitter (the graphical part of MaxMSP), see Figure 2.25 for an example. There is only one (jitter) rendering window (named Vizu); as a result, you can display only one timeline (ie. one of the two inputs — Audio or MIDI — of OMax) at a time. However, when visualizing the audio input, both pitch and spectral modelling can be represented above and below the timeline.

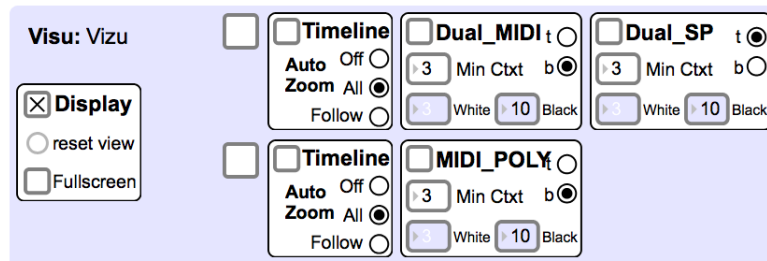


Figure 2.20: Visualization Control Interface

Display

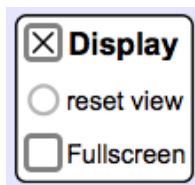


Figure 2.21

The Display module is in charge of refreshing and handling the Jitter window. You can start and stop computing the graphics with the **top left toggle**. The **button** labelled reset view re-centres the view on the default origin of the display. Finally, you can put the visualization fullscreen with the **bottom left toggle**. Press the *Esc* key to come back to the window mode.

Timeline

The Timeline module allows to show an horizontal line representing the elapsed time of recording and learning (growing in realtime). The **toggle** enable and disable this rendering. This module is also in charge of adjusting the zoom of the display. If you turn the Auto Zoom function *Off* with the first option of the **radio button** on the right, the display will simply stay still as it was. Clicking on the *All* option or pressing the *spacebar* of your keyboard will set (and automatically refresh) the zoom to display the whole knowledge. The *Follow* option will slide the display horizontally towards the right as the learning is growing.

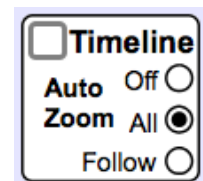


Figure 2.22

Along the timeline and thanks to the Display module — which is also in charge of converting the mouse position into time position in the recording — you can select regions (region 1 in green, region 2 in blue) on the visualization. Simply click, hold and release the mouse to extend the region 1. Defining a region 2 works the same way with the shift key pressed while selecting. Depending whether you selected a region above or below the timeline, this region will refer to (and be used to control) the pitch or the spectral model (see 2.4, radio button).

Links

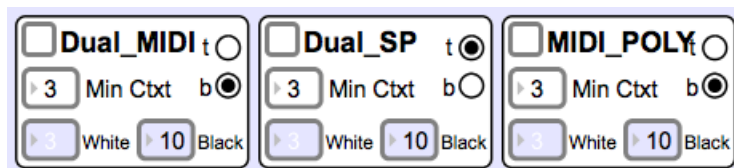
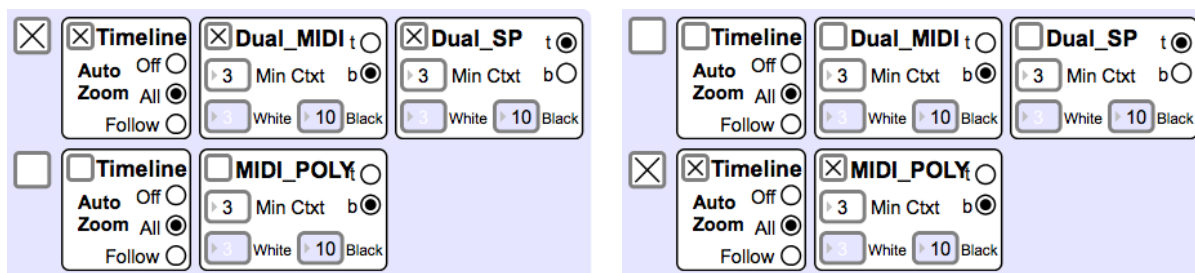


Figure 2.23: Control of the visualization of the links

The main visual information about the state of the model is given by tracing the links of the graph. This is controlled, for each graph being built, with the interface presented above (Fig 2.23). The **toggle** enable and disable this function, the **radio button** on the right of the box controls if the links (for each graph) is displayed above (*t* for top) or below (*b* for bottom) the horizontal timeline (see 2.4). You can choose the minimal quality of links to render with the **number box** labelled Min Ctxt, a higher value will display fewer but better quality links (and may allow you to save some CPU). The quality (context length) of the links are shown with a grey scale going from white (lower contexts) to black (higher contexts). Changing the Min Ctxt value will also set the **White number box** to the same value adjusting this way the grey scale to your display. However, you can also change these parameters by hand by indicating in the **Black and White number boxes** your bounds for the grey scale. It may be useful if you want to project the visualization on a large screen or if you work in specific lights conditions which dims the differences in colors.



(a) Full Audio Visualization

(b) Full MIDI Visualization

Figure 2.24: Toggle Shortcut for Visualization

A handy shortcut is provided in the interface to activate/deactivate all the features at once. The **bigger toggle** on the left of the Timeline boxes will check all the other boxes of the same row. Because there is only one Jitter rendering window, both bigger toggles are mutually exclusive so that you can visualize the two models built on the Audio input **or** the model built on the MIDI input but not both.

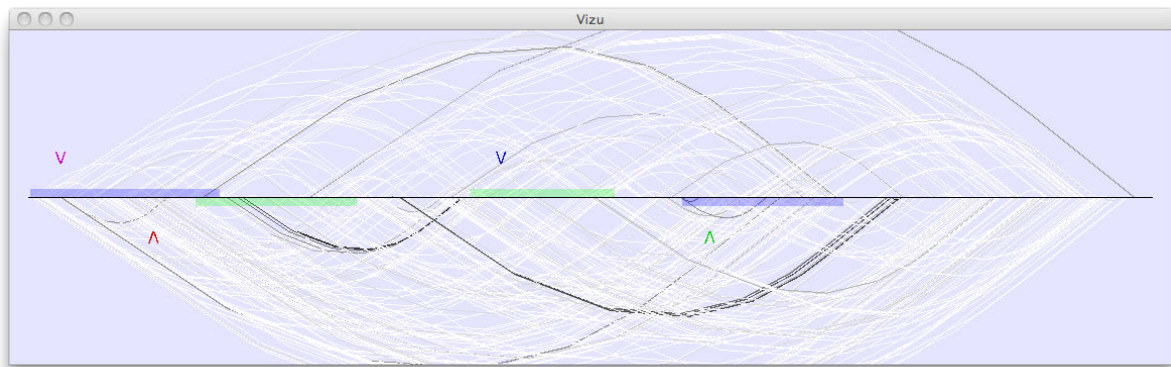


Figure 2.25: Example of the Visualization Window with Regions

Bibliography

OMax

- [LBA12] Benjamin Lévy, Georges Bloch, and Gérard Assayag. “OMaxist Dialectics : Capturing, Visualizing and Expanding Improvisations”. In: *NIME*. Ann Arbor, USA, 2012, pp. 137–140.
- [Ass09] Gérard Assayag. “Habilitation à diriger les recherches - Algorithmes, langages, modèles pour la recherche musicale : de la composition à l’interaction”. Thèse de doctorat. Université Bordeaux I, 2009.
- [Rob+09] Yann Robin et al. *Projet de recherche OMAX (étape I) : Composer in Research Report (2009)*. Tech. rep. Médiation Recherche et Création, 2009.
- [BDA08] Georges Bloch, Shlomo Dubnov, and Gérard Assayag. “Introducing Video Features and Spectral Descriptors in The OMax Improvisation System”. In: *International Computer Music Conference*. 2008.
- [AB07] Gérard Assayag and Georges Bloch. “Navigating the Oracle: a Heuristic Approach”. In: *International Computer Music Conference '07*. Copenhagen, Denmark, 2007, pp. 405–412.
- [ABC06a] Gérard Assayag, Georges Bloch, and Marc Chemillier. “Improvisation et réinjection stylistiques”. In: *Le feed-back dans la création musicale contemporaine - Rencontres musicales pluri-disciplinaires*. Lyon, France, 2006.
- [ABC06b] Gérard Assayag, Georges Bloch, and Marc Chemillier. “OMax-Ofon”. In: *Sound and Music Computing (SMC) 2006*. Marseille, France, 2006.
- [Ass+06] Gérard Assayag et al. “Omax Brothers : a Dynamic Topology of Agents for Improvisation Learning”. In: *Workshop on Audio and Music Computing for Multimedia, ACM Multimedia 2006*. Santa Barbara, USA, 2006.

Improvisation, Oracle

- [CDA11] Arshia Cont, Shlomo Dubnov, and Gérard Assayag. “On the Information Geometry of Audio Streams with Applications to Similarity Computing”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 19-1 (2011).
- [Con+10] Arshia Cont et al. “Interaction with Machine Improvisation”. In: *The Structure of Style*. Ed. by Shlomo Dubnov Kevin Burns Shlomo Argamon. Springer Verlag, 2010, pp. 219–246.
- [Man+10] Fivos Maniatakis et al. “On architecture and formalisms for computer assisted improvisation”. In: *Sound and Music Computing conference*. Barcelona, Spain, 2010.
- [DA08] Shlomo Dubnov and Gérard Assayag. “Memex and Composer Duets: computer aided composition using style modeling and mixing”. In: *Open Music Composers book 2*. Ed. by G. Bresson J. Agon C. Assayag. Collection Musique Sciences ircam Delatour, 2008.

- [CDA07a] Arshia Cont, Shlomo Dubnov, and Gérard Assayag. “Anticipatory Model of Musical Style Imitation using Collaborative and Competitive Reinforcement Learning”. In: *Anticipatory Behavior in Adaptive Learning Systems*. Berlin, Germany: Martin Butz, Olivier Sigaud, and Gianluca Baldassarre, 2007, pp. 285–306.
- [CDA07b] Arshia Cont, Shlomo Dubnov, and Gérard Assayag. “GUIDAGE: A Fast Audio Query Guided Assemblage”. In: *International Computer Music Conference*. Copenhagen, Denmark, 2007.
- [DCA07] Shlomo Dubnov, Arshia Cont, and Gérard Assayag. “Audio Oracle: A New Algorithm for Fast Learning of Audio Structures”. In: *International Computer Music Conference*. Copenhagen, Denmark, 2007.
- [CDA06] Arshia Cont, Shlomo Dubnov, and Gérard Assayag. “A framework for Anticipatory Machine Improvisation and Style Imitation”. In: *Anticipatory Behavior in Adaptive Learning Systems (ABiALS)*. Rome, Italy, 2006.
- [RAD06] Camilo Rueda, Gérard Assayag, and Shlomo Dubnov. “A Concurrent Constraints Factor Oracle Model for Music Improvisation”. In: *XXXII Conferencia Latinoamericana de Informática CLEI 2006*. Santiago, Chile, 2006.
- [AD05] Gérard Assayag and Shlomo Dubnov. “Improvisation Planning and Jam Session Design using concepts of Sequence Variation and Flow Experience”. In: *Sound and Music Computing 2005*. Salerno, Italie, 2005.
- [Aya+05] Mondher Ayari et al. *De la théorie musicale à l’art de l’improvisation : Analyse des performances et modélisation musicale*. Ed. by AYARI Mondher. Paris: DELATOUR-France, 2005.
- [AD04] Gérard Assayag and Shlomo Dubnov. “Using Factor Oracles for machine Improvisation”. In: *Soft Computing* 8-9 (2004). Ed. by M. Chemillier G. Assayag V; Cafagna, pp. 604–610.

Style Modeling

- [Dub+03] Shlomo Dubnov et al. “Using Machine-Learning Methods for Musical Style Modeling”. In: *IEEE Computer* 10-38 (2003). Ed. by Doris L. Carver, pp. 73–80.
- [DA02] Shlomo Dubnov and Gérard Assayag. “Universal Prediction Applied to Stylistic Music Generation”. In: *Mathematics and Music: A Diderot Mathematical Forum*. Ed. by Feichtinger H.G. Rodrigues J.F. Assayag G. Berlin, Allemagne: Springer, 2002, pp. 147–158.
- [Ass+01] Gérard Assayag et al. “Automatic modeling of musical style”. In: *8èmes Journées d’Informatique Musicale*. Bourges, France, 2001, pp. 113–119.
- [Lar+01] Olivier Lartillot et al. “Automatic Modeling of Musical Style”. In: *International Computer Music Conference*. La Havane, Cuba, 2001, pp. 447–454.
- [ADD99] Gérard Assayag, Shlomo Dubnov, and Olivier Delerue. “Guessing the Composer’s Mind : Applying Universal Prediction to Musical Style”. In: *ICMC: International Computer Music Conference*. Beijing, China, 1999.
- [Dub98] Shlomo Dubnov. “Universal Classification Applied to Musical Sequences”. In: *ICMC: International Computer Music Conference*. Ann Arbor Michigan, USA, 1998.

Student works

- [Lév09] Benjamin Lévy. “Visualising OMax”. Master II ATIAM. UPMC-IRCAM, 2009.
- [God04] Jean-Brice Godet. *Grammaires de substitution harmonique dans un improvisateur automatique*. DEA ATIAM. Paris, 2004.
- [Lau04] Cyril Laurier. *Attributs multiples dans un improvisateur automatique*. DEA ATIAM. Paris, 2004.
- [Sel04] Carl Seleborg. *Interaction temps-réel/temps différé*. Mémoire de stage [DEA ATIAM]. 2004.
- [Dur03] Nicolas Durand. “Apprentissage du style musical et interaction sur deux échelles temporelles”. DEA ATIAM. UPMC-IRCAM, 2003.
- [Poi02] Emilie Poirson. “Simulation d’improvisations à l’aide d’un automate de facteurs et validation expérimentale”. DEA ATIAM. UPMC-IRCAM, 2002.
- [Lar00] Olivier Lartillot. *Modélisation du style musical par apprentissage statique : Une application de la théorie de l’information à la musique*. DEA Atiam. Paris, 2000.

Principles and Architectures for an Interactive and Agnostic Music Improvisation System

Abstract

Since the beginnings of computer science, computers have been used fruitfully for the generation of new music. In the last decades, their utilization to create generative systems moved from the implementation of tools for composition towards the invention of reactive programs aimed at participating in the much more unpredictable and challenging situation of musical improvisation. The work presented in this thesis focuses on the conception and realization of such a software, capable of pertinent interaction with acoustic musicians in a collective free improvisation, that is an improvisation without any predetermined knowledge of structures, rules or style. It is extended at the end of our work with considerations on emerging properties such as pulse or a broad notion of harmony. The OMax project proposes to approach the problem of non-idiomatic improvisation by learning and mimicking the style of a musician with an agnostic and incremental knowledge model. We take this computer system as our work basis and examine carefully three aspects: the conceptual principles of the system, the software architectures for effective implementations and the *real-life* usage of this system in numerous testing and concerts situations.

Besides a thorough study of all the conceptual elements of the system based on anthropomorphic decomposition of its parts, our main contribution is the design and realization of several variations of the OMax system. We first propose to use dual structures to store the literal information extracted from the input stream of a musician and to hold the knowledge model built on this information. We illustrate this architecture with a novel real-time visualization of the model. We claim that duplicating all the processes that lead up to the building of the knowledge model enables the computer system to listen at once to several aspects of the ongoing music with their own temporal logics captured in the different model instances. Running this way a multiple descriptions and multiple inputs modeling of the on going musical content greatly improves the pertinence of the computers response. The study of the generation mechanisms of the system enables us to put forward a new database oriented approach to the collection of these models. Our work has been also strongly coupled with the testing of our prototypes with several leading musicians. The musical feedback gathered along these numerous musical experiences lead the work presented in this thesis and opens up many directions for further research on the system, notably the exploration of automatic decisions based on the pertinence of the different descriptions at a given moment of the improvisation or a larger use of prepared material to a *composed improvisation* perspective.

Résumé

Depuis les débuts de l'informatique, les ordinateurs ont été utilisés fructueusement pour générer de nouvelles musiques. Au cours des dernières décennies, l'utilisation de systèmes génératifs s'est progressivement tournée des outils pour la composition vers l'invention de programmes réactifs pouvant participer à une improvisation musicale, situation bien plus imprévisible et stimulante. Le travail présenté dans cette thèse se concentre sur la conception et la réalisation d'un tel système informatique, capable d'interagir musicalement et pertinemment avec des musiciens acoustiques dans le cadre de l'improvisation libre collective, c'est à dire de l'improvisation détachée de toute structures, règles ou style prédéfinis. Nous étendons ce cadre à la fin de notre travail en y intégrant l'utilisation de propriétés émergentes telles que la pulsation ou une notion large d'harmonie. Le projet OMax propose d'aborder le problème de l'improvisation non-idiomatique par l'apprentissage et l'imitation à la volée du style d'un musicien à l'aide d'un modèle de connaissance agnostique. Ce système sert de base à notre travail et nous en examinons attentivement trois aspects : les principes conceptuels du système, les architectures logicielles permettant une implémentation efficace, et l'usage réel du système dans de nombreux tests et concerts.

Outre une étude fouillée de tous les éléments théoriques du système suivant une décomposition anthropomorphique de ses différentes parties, les contributions principales du travail présenté dans cette thèse sont la conception et la réalisation de plusieurs nouvelles versions du système OMax. Nous proposons en premier lieu l'utilisation de structures duales pour contenir d'une part les informations extraites du flux d'entrée d'un musicien et d'autre part le modèle de connaissance construit sur ces informations. Nous illustrons cette architecture avec une nouvelle visualisation en temps-réel du modèle de connaissance. Nous prétendons que la multiplication de tous les processus menant à la construction du modèle de connaissance permet au système d'écouter en parallèle plusieurs aspects de la musique se déroulant. Chacun de ces aspects possédant sa propre logique temporelle mène à la construction d'une instance différente du modèle. Nous obtenons ainsi une modélisation du discours musical basée sur plusieurs descriptions de plusieurs entrées ce qui permet d'augmenter considérablement la pertinence de la réponse de l'ordinateur. L'étude menée sur le mécanisme de génération du système nous permet de proposer une nouvelle approche de la collection de modèles de connaissance vue comme une base de donnée. Notre travail a été fortement associé à des tests réguliers des prototypes du système avec de nombreux musiciens de premier plan. Les réactions collectées au cours de ces nombreuses expériences musicales ont fortement orienté le travail présenté dans cette thèse et ont ouvert de nouvelles directions de recherche autour de ce système, notamment l'exploration de la prise automatique de décisions éclairée par la pertinence des différentes descriptions du flux musical au cours de l'improvisation ou une plus grande utilisation de matériau musical préparé dans le cas d'*improvisations composées*.