CATÓLICA PORTO
EDUCAÇÃO E PSICOLOGIA

MMixte: A software architecture for Live Electronics with acoustic instruments. Exemplary application cases

Thesis submitted to the Portuguese Catholic University for the Doctoral Degree in Science and Technologies of the Arts

by

Maurilio Cacciatore

ESCOLA DAS ARTES
September 2018

CATÓLICA PORTO
EDUCAÇÃO E PSICOLOGIA

MMixte: A software architecture for Live Electronics with acoustic instruments. Exemplary application cases

Thesis submitted to the Portuguese Catholic University for the Doctoral Degree in Science and Technologies of the Arts

By Maurilio Cacciatore

Supervised by Prof. Erik G. Oña, PhD

ESCOLA DAS ARTES
September 2018

Table of contents

Index of Figures

Foreword

The following dissertation is in two parts; the first one is further subdivided in two sub-sections.

The argument stresses the significance of software architecture for mixed music, with special attention paid to solutions available in the market for middleware programming to manage electronics in concert situations.

Following a description of the history of technological evolution all through the development of mixed music repertoire in the past few decades, I move on to describe the principal theoretical solutions for software architecture and software solutions available.

On the basis of this research over the last four years, I have been working on a collection of software modules I have called "MMixte", all of which are dedicated to software programming within the software Max [1] The theoretical background behind MMixte enables, if necessary, the project's exporting to other software platforms, in the future. MMixte gathers experiences of mixed music, especially in France, from the last thirty years and rationalizes signal path within a software program. The order and rationality applied to the management of audio signal enhances programming quality, the simplicity of its graphic interface, and at times even the quality of the sound result. Whenever necessary, it makes debugging much faster. Most dissertations about live electronics take audio signal processing into consideration as one of the aspects relevant to this domain, but the topic of software architecture rationalization implementing old and new algorithms seems to have been neglected in the past few decades. Awareness of the significance of software architecture was voiced in the Nineties; live electronics, as we know it today, has reached its musical importance in the field of composition thus evolving, in its programming strategies, through force of habit. To me, work in rationalizing strategy seems necessary as it may be helpful to the community of live electronics composers and programmers. The MMixte project is online and has been for quite some time already: it is available to anyone wishing to use it; it does not in any way concern audio signal processing and focuses exclusively on the organization of all such components that make

[1] "Max is a visual programming language widely used in by artists and in academia for the development of ad hoc software for audio, visual media, and physical computing". www.cycling74.com.

the audio signal path efficient. Due to its nature, it is proposed as middleware, ready for use in addition to the possibility of being modified depending on specific needs.

In the second part of this dissertation, I illustrate some of the pieces I have written in these last few years, using MMixte to program their electronic sections.

As complementary development of my compositional work, I add an appendix for one of the prototypes I have developed through these years, one that is likely to become an object for further advancement in years to come.

Abstract

MMixte is a middleware based on Max for mixed music with live electronics. It enables programming for a “patcher concerto”, a platform, that is, for the management of live electronics in just a few minutes and with extreme simplicity. Dedicated to average and expert users, MMixte enables true programming of live electronics in very little time while also enabling easy adapting of previously developed modules, depending on the case and its needs. The architecture behind MMixte is based on a variation of so-called “pipeline architecture"; the analysis of the most widely used software architectures in the market and design patterns to program graphic interfaces has led to the conception of ways of organizing communication between various modules, the way they are being used and their graphic appearence. Analysis of other, “state of the art” module collections and other software programs dedicated to mixed music shows the absence of another work on software architecture for mixed music. Application of MMixte to some of my personal works shows demonstrates its flexibility and ease of adaptation. Computer programming for a piece of mixed music requires much that goes beyond just programming of audio signal processing. The present work seeks to provide an example of a solution to such needs.

Keywords

MMixte, Middleware, Mixed music, Max, Software architecture, Design pattern, Pipelined architecture, Computer score, IRCAM, Maurilio Cacciatore, ESB.

1. Mixed Music between the History of Composition and the History of Technologies

1.1 Birth and Development of Mixed Music

The term mixed music designates the co-presence of live acoustic instruments with acoustic digital resources from audio files or real time elaboration of information coming from musicians. It is customary to locate the birth of mixed music in 1939, the year when John Cage composed *Imaginary Landscape* n. 1[2] . For the first time ever, pre-recorded sounds were integrated in an instrumental score. K. Stockhausen's works from the Sixties were crucial in spreading this new paradigm of musical composition. By the Seventies, many composers gathered in a number of European countries, often working for national radio or research center-based electronic music studios implemented various combinations of acoustic and electro-acoustic resources in their works. Interaction techniques between musicians and machines occurred in the wake of technological development. Mixed music was, for quite a long time, the prerogative of just a few composers possessing the skills and the artistic means to work in electronic music studios where creation of tapes with pre-recorded sounds was possible; creation of a mixed music work presupposed the existence of a team, a work place, therefore a production budget. It was only in the last thirty years, with the mass computers distribution, that mixed music production became a reality outside research centers. The evolution of electronics in real time went through a similar fate over a shorter time span. Born in the Sixties, this sub-genre[3] had begun developing in the Seventies: indeed, IRCAM's contribution from the early Eighties on was crucial for the development of its techniques and technologies. Numerous American universities also contributed to forming research teams whose practical results in the conception of new hardware and

[2] J. Cage, *Imaginary landscape n. 1*, 1939, London, Editions Peters, EP6716.

[3] At the time, live electronics involved hardly more than amplifying a few acoustic sources with such an effect on audio streaming resulting from sending the signal into the machines available back then. The use of variable frequency oscillators was already possible in 1939. As a matter of fact, John Cage uses two of them in his *Imaginary landscape n. 1.* This, however, does not justify calling it a work of *live electronics*, since it does not involve the kind of audio elaboration required to differentiate diffusion from its source.

software equipment, enabled musical production centres to make new technologies available for composers to implement in their mixed music works. The works of composers thus functioned as practical applications of technological research also going through beta testing phases since, through intensive use, programming faults were detected just as programming efficiency and interaction between user and software was being tested. This relationship of beta testing between software programs and pieces of music first using it still applies today .[4] The construction of machines dedicated to such interaction in research centres helped define a new basis for new paradigms; only in the last twenty years, the power of commercial computers and the possibility of privately purchasing user licenses for professional software has enabled the creation of mixed music works outside production and research institutions. The conception of the works' electronic sections, the kind of electro-acoustic material used and the compositional techniques used to enable the marriage, so to speak, between instrumental and electro-acoustic resources within the score provided definitions for distinctive compositional trajectories:

- Deferred Time Mixed Music: instrumental pieces with audio files all through the piece.
- Music for Amplified Instruments: one or more microphones diffusing audio on a speaker chain capture instrumental performance. Audio is handled in terms of dynamics, generally with the purpose of enabling the perception of sounds too feeble in the real world for clear perception in a concert hall or to attempt balancing them out with those of classic instrumental playing. Although the use of reverb is technically real time audio transformation, it is considered rather as a necessary “adaptation” to the acoustic reality of the concert hall, and firmly remains a part of the characteristics of this repertoire, alive and well nowadays.
- Real Time Mixed Music: the musicians' instrumental play is captured through microphones and transformed as to dynamics (amplitude, equalization, signal compression), morphology (delay, freeze, granulation) and spectral components. Transformations in the spectral realm include all possible operations involving dynamics and morphology operating, however, from an FFT or other analysis algorithm instead of

[4] I have had the chance to contribute to the current version of *Antescofo midi*, algorithm for score following, when my *Concerto per Tastiera Midi, ensemble e live electronics* (2010-2011) was premiered at IRCAM in 2012. It was also at IRCAM that a new version of CataRT was implemented to enable some of the spatialization processes used in my piece *Tamonontamo* (2012). In both cases, programming algorithms was constantly being updated for about three months in view of improving their codes, thinking they were ready for public release at the time of the first performance of the works and presentation lectures.

handling original audio dynamics; this presupposes constant analysis of the audio streaming making up the basis of digital data subject to elaboration following a predetermined algorithm transforming input data according to calculated procedures.

- "False" Real Time Mixed Music: a diffusion technique involving the use of real time elaboration of an instrumental section in a pre-recorded studio; such audio streaming was recorded on analog support up to the early years 2000 [5] and simultaneously diffused with the live instrumental section, much like a mixed music piece with deferred time electronics. Such technique does have certain advantages:

  - the cause-effect relationship between the instrumental section and the streaming diffused through the speakers is maintained;

  - the computer's calculations noticeably decrease through the concert performance since, from a technical standpoint, one only dealing with diffusion of audio files with no digital elaboration at all Since real time elaboration is produced in the studio, problems that might arise, such as proper measuring of parameters, electro-acoustic chain stability and machine response, et cetera, may addressed without the interruption of instrumental performance becoming much of a problem;

  - reducing, if not avoiding, accidents inextricably bound to live performance and potentially causing qualitative changes in real time elaboration that might spoil, as a result, the composer's artistic idea.

Conversely, false real time can hardly benefit from the kind of expressivity which live electronics ensure in audio elaboration depending on instant streaming needed live on stage.

[5] The use of cassettes ADAT for professional digital music up to eight channels was common until the early years 2000; computers' significant increase in RAM has enabled streaming of long, multi-channel pieces directly from the computer.

1.2 The Interaction between Instrumentalist and Electronics

Pieces of deferred time mixed music do not require the ADC/DAC [6] signal transduction in the electro-acoustic chain since instrumental performance and audio[7] files are independent. For a long time, digital sounds were transferred in the studio on a magnetic support (tape) to be diffused from a player throughout the live performance of the instrumental section, the electronic section was a single audio file starting at the beginning and reaching the end of the composition[8]. In other words, there was no human, let alone artistic, interaction between the instrumental performer and the electronic part:

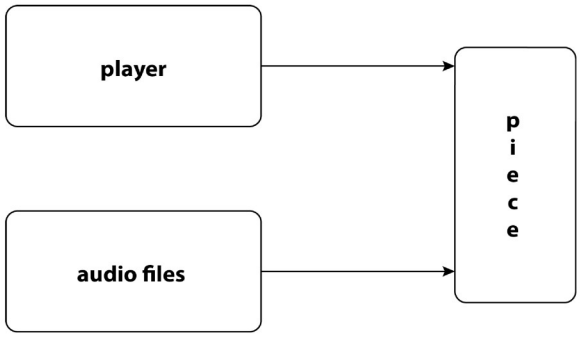


Fig. 1.1 deferred time mixed music interaction n° 1

Those in charge of tape playing machines or audio files and diffusion machines usually activate them from the mixing board. In many cases, it is the composer himself who attends to such tasks, in collaboration with other technicians.

The excessive length of the deferred time section was a major drawback from a performance standpoint. Because the section was fixed once and for all, pre-recorded and subject to no changes over time, live performers had no choice but to adapt to the electronics section doing their best to synchronize their own instrumental performance. This marked a sort of annihilation of the performer's interpretive dimension who was

[6] Analog to digital conversion / digital to analog conversion. For the sake of simplifying the argument, the need of amplifying acoustic instruments is excluded, in this case, as it would presuppose the need to place a microphone in front of the instrument in order to capture its signal.

[7] Conceptually, and as far as the present argument is concerned, the function of digital audio files as deferred time electronics is very much like that of tape; as a matter of fact, in our collective imaginary, we still hear talk today about pieces “for instrument and tape” to indicate the presence of audio files even though, technically, tape is no longer used.

[8] K. Stockhausen, *“Mixtur”* and *“Mikrophonie 1”*, in *La musica elettronica*, Milan, Feltrinelli, 1976.

asked, in such contexts, to perform without the possibility of managing the time of musical performance in terms of his own sensitivity. Time management through *accelerandi* and *rallentandi* is no doubt one of the most sensitive expressions of musical interpretation. The performer moves around the metronome whose values the composer sets in the score. These, however, are no more than a reference point since absolute respect would presuppose performing the piece in question with a metronome or a click track. If respect for tempo in the instrumental performance by the instrumentalist is relative to interpretation, the audio file's temporality is absolute insofar as the sounds are always reproduced in the same way, always at the same speed. Even when diffusing audio from a DAW which enables changes in dynamics and audio morphology with relative ease, audio file diffusion will always be stiff, making the performance with the instrumental section just as stiff because the performer can do no more than adapt and follow the tempo as best (s)he can.

Such a degree of rigidity increases proportionally to audio file length, thus also increasing the chances that a mistake might occur: if the instrumentalist were to lose track of synchronization with the electronics in deferred time without finding it again, the faulty result would equal the audio file duration. This has led to such demands as further splitting the electronic section in order to reduce to a bare minimum the chances of making a mistake resulting from inaccurate synchronization between instrumentalist and electronics. Following the birth of Digital Audio Workstations (DAW), the electronic sections for the pieces were not mastered and thus left at the mix level, using the DAW work session for last minute changes in equalization, compression, diffusion volume, and the sounds' position within the track as a whole. For both machine operator and instrumentalist, the score was the common reference point where annotations for the beginning of the audio file were included:

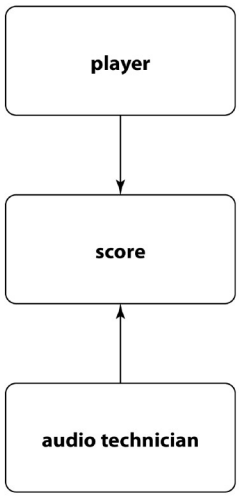


Fig. 1.2 deferred time mixed music interaction n° 2

The passage of the diffusion of tracks in deferred time from analog tape to a DAW session has easily enabled the cutting up of the deferred time electronics in several parts, taking advantage of eventual pauses in the diffusion of the electronics. The computer operator takes advantage of the pause to set the cursor towards the new group of audio regions to be diffused, ensuring diffusion restart at exactly the point indicated in the score.

In the case of music for amplified instruments (as well as in real time electronic music), one or more microphones capture the instrumental sound and a conversion of the signal from analog to digital takes place. After processing, in the mixer itself or by means of a computer, the signal is converted back to the analog domain and is sent to the loudspeakers. Man-machine interaction with a time latency below 100 ms between audio input and output[9] is usually referred to as live electronics.

The electro-acoustic chain required for music for amplified instruments with ADC/DAC conversion is illustrated in the following diagram:

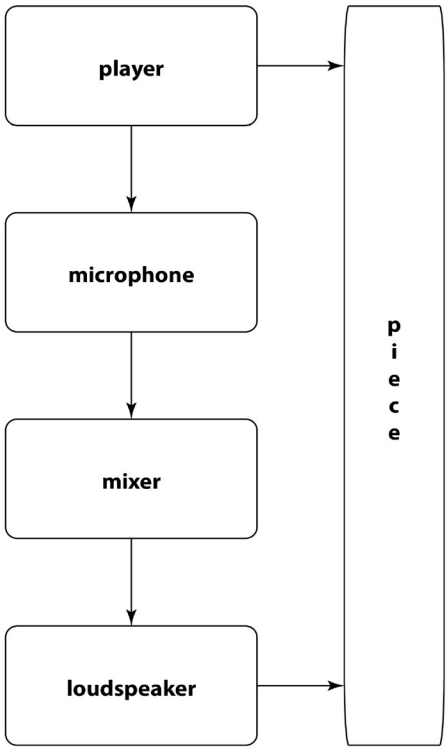


Fig. 1.3 amplified instrument music interaction n° 2

In the case of real time audio elaboration, the passage of audio through a processor (today's computer) will be necessary.

[9] Roads Curtis, *The computer music tutorial.* Cambridge, The MIT Press, 1996.

The passage of audio through the mixer today is sometimes substituted by direct connections between microphones, audio cards and computers. In this case, the handling of audio dynamics is performed *in the box* through software plugins instead of being performed *out of the box* through assistance from dedicated machines. The present study shall not be delving into an analysis of the differences between the two types of chains since, from the standpoint of mixed music architecture given that both solutions lead to the same interaction scheme:

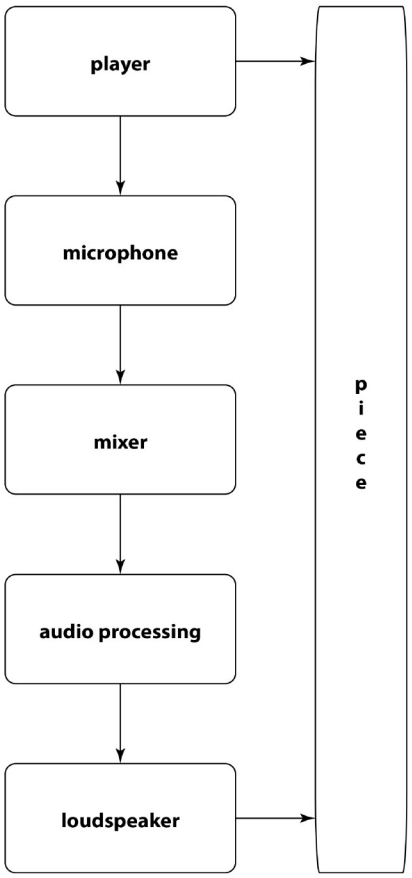


Fig. 1.4 interaction in real time music

1.3 Distinctive Traits of Computer Programming for Mixed Music

1.3.1 Midi Protocol

In 1981, Sequential Circuit technicians organized the first public showcase of Midi computer language and its first commercial application with the Prophet 600

synthesizer[10] which launched in 1982, also at Sequential Circuit's initiative. Although the GM2 protocol could not reach Midi standards as we know them today until 1997, it has long-since established itself as the digital protocol suited to data transmission in all direction between digital instruments, computers and controllers.

One of the first applications of the Midi protocol to a piece of mixed music was Philippe Manoury's *Pluton*[11] for Midi Grand Piano (nowadays, Disklavier) and 4X[12]computer [13].

The use of the Midi protocol in mixed music has enabled the practice of what may be defined as Manoury's most significant theoretical contribution as far as the development of mixed music is concerned. Since the end of the eighties, Manoury has been conceptualizing and putting into practice “virtual scores”, later called “computer scores”. The first piece officially adopting such a procedural approach was *Jupiter* [14] for flute and live electronics.

1.3.2 The Pedal

A large part of the mixed music repertoire for solo instruments and live electronics has relied upon assistance from Midi “sustain” pedals. Such pedals feature a spring mechanism bringing the pedal back into rest position when no foot pressure is applied. They send figures 0 and 127 in both possible positions: up and down. Original use of the pedal pertains to sending “sustain” (n° 64) control messages to a keyboard or a Midi

[10] Mikhaïl Malt, *Introduction à la norme Midi*. Cursus 1 in music computer science 2009/2010. IRCAM, Paris, 2010.

[11] Philippe Manoury, *Pluton*, Editions Durand, Paris, 1989.

[12] 4X is a circuitry card to be assembled in a computer for its expansion. It was produced in the eighties by Giuseppe di Giugno at Ircam. Dedicated to real time audio elaboration, it represents one of the first computers for the processing of real time professional audio. It was followed by the IRCAM Signal Processing Workstation (ISPW), at first implemented as an expansion of the NeXT computer. The server software in charge of managing the three DSPs proceeding towards digital computation of audio elaboration was called FTS (*Faster Than Sound).* The software program which then became independent in the Nineties, Max Opcode, actually was a transcription/adaptation of Max-ISPW FTS. various mathematicians and computer developers worked together on its software program: among them, Miller Puckette who produced the first version of FTS precisely as an object programming language for the use of computer 4X functions. The Midi protocol was already in use for the management of digital data for NeXT as well as 4X later on.

[13] Favreau, E., Fingerhut, M., Koechlin, O., Potacsek, P., Puckette, M., and Rowe, R. *Software Developments for the 4X real-time System*. Proceedings, International Computer Music Conference. San Francisco: International Computer Music Association, pp. 43-46, 1986.

[14] Philippe Manoury, *Jupiter*, Paris, Editions Durand, DA 505, 1987.

controller capable of transmitting this type of message. When applied to a digital music context, the digital message enables imitation of the effect obtained with the use of the grand piano tone pedal. In this case, the function linked to the “sustain” control message is not used for its original purposes. The pedal's mechanical characteristics are exploited to send an electrical impulse to the live electronics control software. This impulse then progressively sets a counter whose numbers correspond to the index numbers of the events of the electronic part. The sustain pedal is generally used by the same musician performing on stage. The advantage of assigning this tool for the responsibility of changes in software status to the player is that musicians can effectively synchronize instrumental playing with the event launch since the tool does both. If, at times, tight interlocks between musical events and pedal action are difficult for an operator from the mixing board, they are much simpler for an instrumentalist as the action on a pedal amounts to (familiar) instrumental action. Strikes on the sustain pedal are commonly notated in the score:

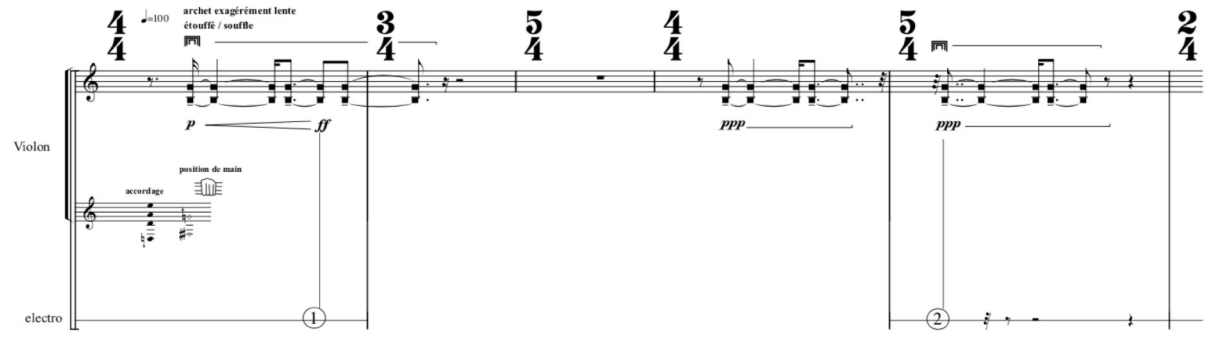


Fig. 1.5 Santiago Diez Fischer, *Loop’s definition* (2010), for Violin and electronics.

In the example of Fig. 1.5, numbers on the line below the one for violin indicate index numbers for the events of the electronic part to be activated as indicated in the score with a sustain pedal.

This system, however, does have a drawback concerning the visual side of the set-up: activating electronics events immediately following a single movement on the pedal makes the relationship between acoustic source and electro-acoustic speakers artificial. It is clear, for the audience, that something in the electronics has just occurred because the pedal was just activated. A number of events to be activated at a high temporal frequency is hardly suitable for this particular technical solution.

1.3.3 The Midi Keyboard

Easy programming of a number coming in the form of a Midi Protocol message control lies at the very root of the technique enabling integration of the protocol itself within software programming dedicated to mixed music. The Midi keyboard is another controller with numerous applications. It reproduces a grand piano keyboard (Midi Keyboards exist in a variety of sizes from twenty-five to eighty-eight keys) sending two sets of digital data: The note's (midi) pitch (a value between 0 and 127) and the note's "velocity" (a value between 0 and 127 too) indicating the speed at which the key is lowered, usually correlating with the intensity of playing. When the Midi keyboard is used as controller for event launch of the electronics, the key's "pitch" is the only number used, thus recalling the event of the electronics with the same index number (or, a different one, if need be). In such cases, as it often happens, the Midi keyboard is notated in the score to be played; it takes a musician able to correctly follow the tempo (or the conductor, if present, or through synchronization with the other musicians); in practical terms, those in charge of activating electronics events are musicians whose contribution counts just as much as the other musicians' on stage (Fig. 1.6)[15].

It is easy to see this technical solution's advantage, especially in pieces of lengthy duration including many events: programming up to 88 events (as many as the keys) can be done without any difficulties. Events can be arranged in ordinal sequence such as a chromatic scale or not; in the first case, sequential reading of the instructions in every event is meaningful because it is analogous to the evolution of changes in software status whereas the second one is not since sequential reading of the events linked to pitch numbers (indeed, used as event index numbers) has no link to the score events in the ordinal sequence. Reasons for one choice or another concern conditions beyond programming[16].

As we can see in Fig. 1.6, such a technical solution leads to the paradox whereby a musical section notated on the staff (since, in order to indicate the keys on a midi

[15] The controller part is notated as "Smpl." in the score.

[16] Extra-musical reasons leading to one choice or another might concern, for instance, the types of instructions or audio files to be activated when certain buttons are pressed. A composer might, for instance, prefer to activate a high sound with a key up on the high register just as be would, for a low sound, be activating a key in the lowest octave of the keyboard. If the person at the keyboard also needs to use other controllers, disposition (to the left, to the right or in front of the keyboard) can be a motivating factor leading to a preference for planning actions through one of the keys of the keyboard in the low, middle or high register. Stage motivations can also lend support to these kinds of choices grounded in musical but not, strictly speaking, computer-based reasons.

III - To be *invisible*

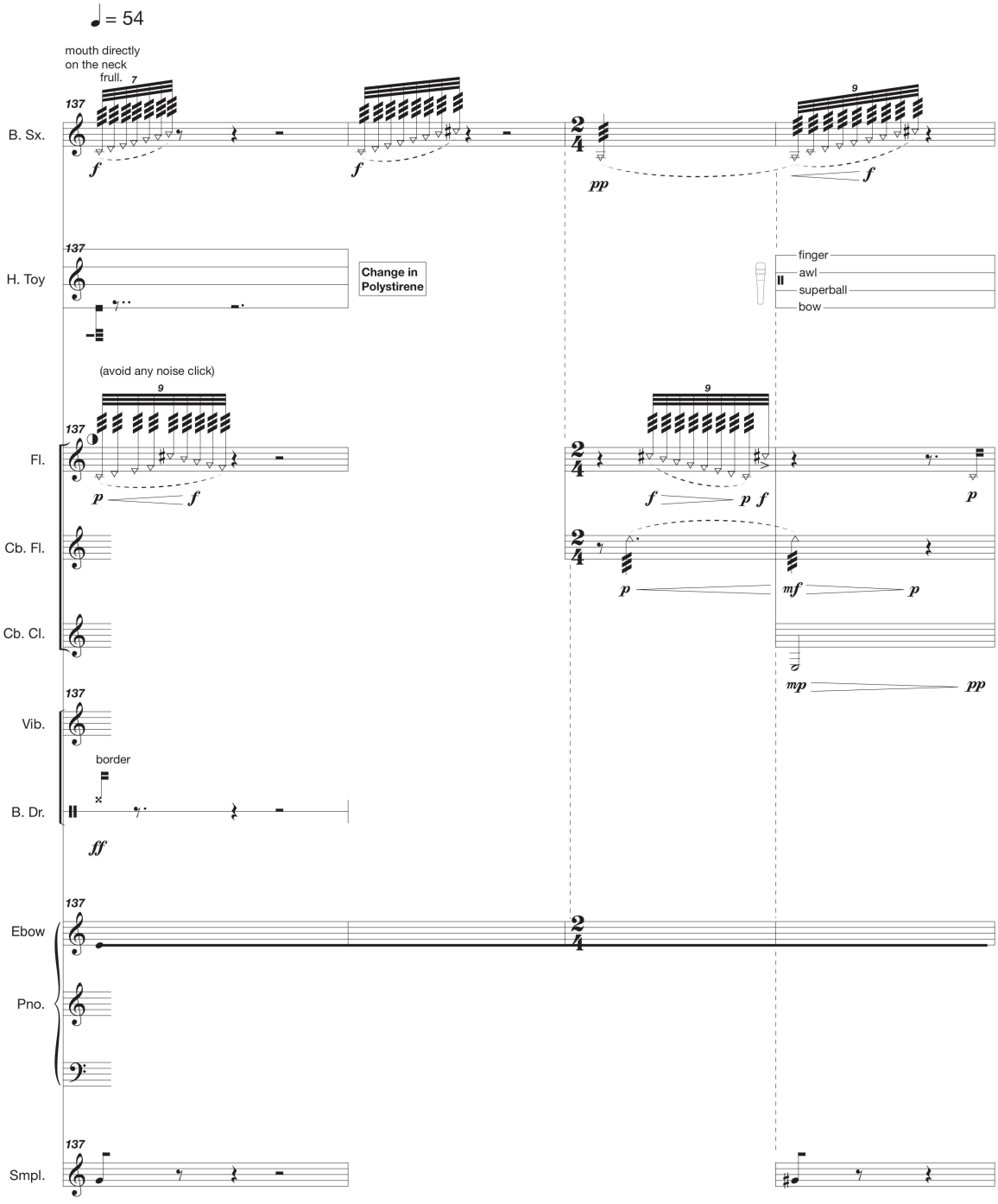


Fig. 1.6 Excerpt from Maurilio Cacciatore, *Meccanica della solitudine (2018)*, for barytone saxophone soloist, percussion co-soloist, ensemble, live electronics and stage setup, pag. 46. Edizioni Suvini Zerboni, Milan

keyboard, the notes corresponding to the keys are written not as pitch numbers but as if a conventional keyboard were used) carry mere symbolic and temporal value, not a musical one: in other words, written notes match computer sections, not music.

1.3.4 Sensors

Sensors are objects capable of generating small electrical currents or resistance to electrical currents. By means of interaction with the external world and through appropriate connection with some sort of transducer, they are capable of gathering information and converting it to a digital data format.

Their use is typical of interactive electro-acoustic installations where interpreting data from the outside world is necessary in order to obtain status changes of the set-up the installation is made up of.

Digital data expressing the way the sensors work are generally computed according to the canons of Midi protocol (counting from 0 up to 127, compatibility with protocol control figures), facilitating their integration in a mixed music hardware/software system since computer programmers already possess the necessary know-how for the integration of the hardware section with the rest of the software. Sensors mainly in use (pressure sensors, speed meters, and passage sensors) need to be connected to a device capable of collecting and digitalising the data and to transmit it in turn to the software program.
One of the most used hardware solution in the past few years has been the Arduino board[17], prototype construction with Arduino presupposes specific skills in the fields of electro-technics, electrical engineering and other specific skills depending of the nature of the project.

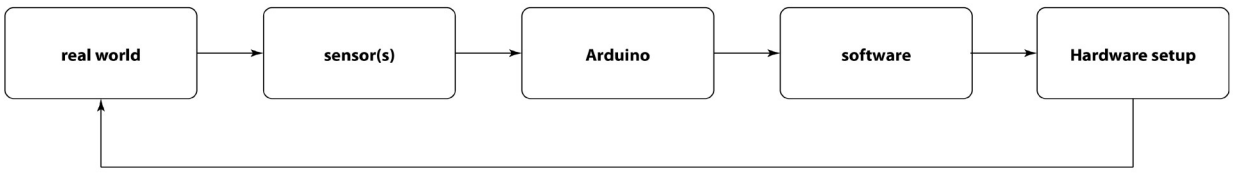


Fig. 1.7: interaction model with Arduino

[17] https://www.arduino.cc

1.3.5 The Computer Score

A computer score is a list of instructions split in groups called “scenes” or “events” whose purpose is to change status of the computer in charge of real time elaboration, to diffuse audio files and, more generally, attending to all functions needed for proper instrument/computer interaction.

These groups of instructions are recalled through index numbers progressively arranged and indicated in the score[18] .

Technically, a computer score might be defined as a list of presets. L’elettronics split in events cannot do without such a list; the alternative would be the manual status change of the software program, which makes sense from a live performance standpoint when the computer operator is on stage but obviously is not very practical when several actions must be simultaneously activated.

A computer score may contain the following instructions:

- DSP changes;
- digital data for parameters, internal to the software program;
- digital data for parameters external to the software program, whenever a bridge between audio software and third software is present;
- digital data referring to index numbers for audio file cue lists;
- file recall instructions of a different nature, external to the software program;
- instructions enabled by the platform hosting the audio software program;
- every command whose scripting is comprehensible from the platform hosting the audio software.

The corollary to the previous list is that a computer score only contains instructions and, with the variants for all cases depending on the platform chosen for programming, it is shown in a script.

[18] Depending on personal and editorial habits, ordinal numbers matching electronics events are sometimes written directly on the official score or on a secondary one called “computer producer score”.

1.4 Aesthetic Musings on Practices of Mixed Music Software Programming

The shift from “rigid” electronics to electronics split into events through computer score programming has empowered a truly interpretive dimension in mixed music, one where musicians on stage are not mere performers obeying musical indications in the score but a legitimate interpreter of music. The mechanical action of events launching is subject to musical playing and yet it is subordinated to instrumental performance and that is what makes such practice more convincing compared to the independence between musicians and electro-acoustic parts.

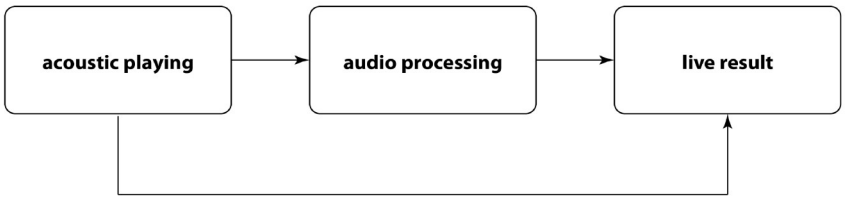


Fig. 1.8: diagram showing interaction in mixed music with live electronics

One of the distinctive traits of Manoury's early works and, in all fairness, all of French mixed music of the eighties and Nineties was the utopia of programming *expressive* electronics in such a way as to automatically adapt to extemporaneous instrumental playing, to diffusion sites, and to acoustic conditions. In other words, one of the objectives of what used to be considered proper real time electronics programming was the capacity of the software program itself to escape the cause-effect relationship so typical of stiff interaction and to approximate human interaction[19] (fig. 1.9).

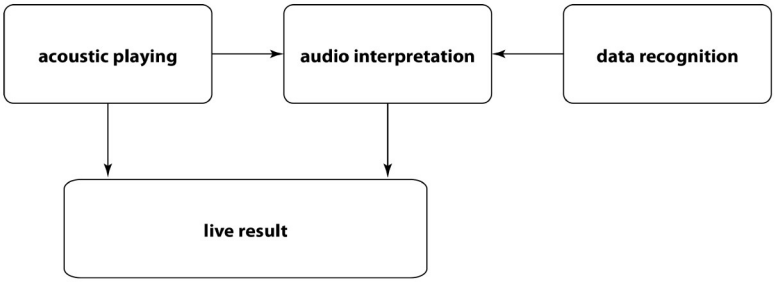


Fig. 1.9: Interaction model where parameters from the external world influence the parameters of the live electronics.

[19] See the concert program notes or the piece's first performance: http://brahms.ircam.fr/works/work/10482/.

Over the years, mixed music research has evolved, especially in the realm of software development: indeed, Max (then Pure Data, again, thanks to Miller Puckette) has perhaps become the most widely used programming tool for ad hoc systems for real time electronics. This software program's strength lies in visual programming, which leads directly to the possibility of creating graphic interfaces and implementing new functions in a number of ways [20], all of which depend on technical needs, on production paradigms [21] and on individual programming skills. A number of software programs for audio elaboration are produced as standalone applications programmed in Max which, since 2006, is also available in its *Max for Live* variant designed to work within Ableton's Live program.

[20] Until Max's version 6, it was necessary to implement a new code in order to create a new object. Now that the *"gen~"* package is available, ad hoc functions can be written internally through the software program only using default Max objects or functions.

[21] The choice of a strategy to adopt instead of programming a new code depends, for instance, on programming a public release of such a code or letting the composer/programmer use it exclusively.

2. Software architecture

2.1 Historical framework

The need for research in software structures was first acknowledged in the Sixties[22]. Since the Seventies, research has been paying considerable attention to *software design*. Its premise was that design as an activity is distinct from implementation, given its demands in terms of special notation, techniques and tools[23].
In the Eighties, the focus of software engineering research shifted away from specific software design and increasingly towards integrating designs and design process into the broader context of software process and management. As a result of such integration, implementation languages absorbed many of the notations and techniques developed for software design.
Research on software architecture was born in the Nineties. The term “architecture” replaced that of “design”, suggesting notions of codification, abstraction, standards, formal training, and style.
From the Nineties to the Two-thousands, software architecture definitely rose from a sub-discipline to a prominent domain in software engineering. Job titles as *Technical Architect* and *Chief Architect* now flourish in the software industry. A Worldwide Institute of Software Architects[24] and a great number of professionals are eager to emphasize their roles as architects in software engineering systems.

2.2 Software architecture definition

Trying to define a term as software architectures (SA) always implies potential danger in light of the absence of a widely accepted definition in the academic world; nor does the industry provide one. The number of SA definitions in the literature and on specialized websites is impressive. A community of software developers has collected all

---

[22] F. P. Brooks, Jr., *The Mythical Man-Month*, Addison-Wesley, Reading, M.A. 1972.

[23] G.D. Bergland, *A Guided Tour of Program Design Methodologies,* IEEE Computer, Vol. 14, No. 10, Oct. 1981, pp 13-37;
P. Freeman and A.I. Wasserman, *Tutorial on software design techniques*, IEEE Computer Society Press, 1976;
W.E. Riddle, J.C. Wilden and A.I. Wolf, *OROS, Towards a Type Model for software Development Environments, Proc, OOPSLA ’89,* New Orleans, Louisiana, October 1989.

[24] http://www.wwisa.org

definitions of AS[25] all of which tend to focus on different aspects of the topic in their attempt to give a definition from a specific point of view. The relationship between hardware and software, CPU engineering and software instruction, the relationship between hardware and programming language, programming style, purposes of use, and the level of opening or closing of the system are among the most common approaches to this definition. The ISO/IEC Standard 42010 "Systems and Software Engineering—Architecture description" defines software architecture as "The fundamental properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Three of the most widely used definitions in academic areas most often quoted in specialized journals:

- *"Architecture is defined by the recommended practice as the fundamental organisation of a system, embodied in its components, their relationships to each other and the environments, and the principles governing its design and evolution."*[26]

- *"The software architecture of a program or computing system is the structure of structures of the system, which comprise software elements, the externally visible properties of those elements and the relationships among them."*[27]

The third main definition makes such issues as scalability and distribution more explicit than the first two definitions:

- *"Software architectures go beyond the algorithms and data structures of computation; designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organisation and global control structure; protocols for communication, synchronisation, and data access; assignment of functional to design elements; physical distribution, composition of design elements; scaling and performance; and selection among design alternatives."*[28]

In an attempt to merge the above definitions, I shall be considering one last definition of SA:

---

[25] http://www.sei.cmu.edu/architecture/start/glossary/community

[26] ANSI/IEEE Std 1471-2000, *Recommended Practice for Architectural Description of Software-Intensive Systems.*

[27] L. Bass, P. Clements, R. Kazman, *Software architecture in Practice* (2nd edition), Addison-Westely 2003.

[28] D. Garlan, M. Shaw, *An Introduction to Software Architecture*, Advances in Software Engineering and Knowledge Engineering, Volume I, World Scientific, 1993.

- *"The architecture of a software system, shortly called Software Architecture, is the structure of the system, constituted by the parts of the system, the relations among the parts and their visible properties."*[29]

Various authors tend to identify architecture with its description. However, in the last definition provided, concepts are kept distinct. Software architecture works for defining how the sub-parts of a system work together, how is the hierarchy among these sub-parts and how the same structure can be modified or translated on another platform along the time. Software architecture description is the "philosophical" concept behind the realization of the software itself.

2.3 Architecture's purpose

Every software architecture should contain some features in order to apply to a practical realization. These way of conceiving an architecture has become a standard in the last 20 years, allowing the update of software also by third-part developers without the onset of troubles in previous users. Basically, these features are:

- **Modifiability**. The change of a sub-part does not face the change of other sub-parts nor their functioning.

- **Portability**. The software might be re-written for another platform or another hardware structure[30], existent or future.

- **Re-use**. Sub-parts of a previous software could be useful if implemented in a new software architecture. The conceiving of sub-parts that can be easily adapted to new projects makes faster the realisation of new software systems.

These concepts have been developed by large-scale software companies dealing in domains not related with music informatics. Although their first application did not concern mixed music, they can be applied for conceiving software architectures dedicated to music informatics. Composers usually re-use strategies or parts taken from previous works by their own and this way of work needs the creation of workspaces that

---

[29] C. Montangero, L. Semini, *Architettura software e Progettazione di dettaglio,* Dipartimento di Informatica, Università di Pisa, University Press, 2014.

[30] This feature is dramatically important nowadays: a large part of software provide an implementation for Windows and for Apple computers, as well as mobile application provide their version for Android and Apple platforms.

can be easily modified for their adaptation to new application. Portability is also important and it is a topic that also touches the domain of conservation of digital art.

2.5 Architecture styles

Architecture style is a set of principles implying ways of elaborating information and ways in which various software program sections interact with each other. Nevertheless, architecture style also implies the delicate shift between software and hardware, considering which method, sequence, priority and hierarchy computers elaborate information coming from the software section. Thus, there's a time when a software program's architecture style affects - and is itself affected by - the hardware's architecture the software was installed in.
Below is a list of architecture styles that might concern audio applications. Styles relevant to other kinds of different software compared to those relevant here are therefore excluded:

- **Pipe-Filter** style enables sequential data processing. Components are “filters” reading data streams on its inputs and producing data streams on its outputs. Connectors are *pipes* transmitting the output streams of one filter to inputs of another. Two or more filters can operate in parallel: one downstream filter can operate on the first data of the sequence di output of an upstream filter, while it continues elaborating its own input sequence. In the Pipe-Filter style, components and connectors behave in particular ways and should be configured so as to enable sequential data processing. Components, called “filters,” read data streams on its inputs and produce data streams on its outputs, typically applying local transformations to every

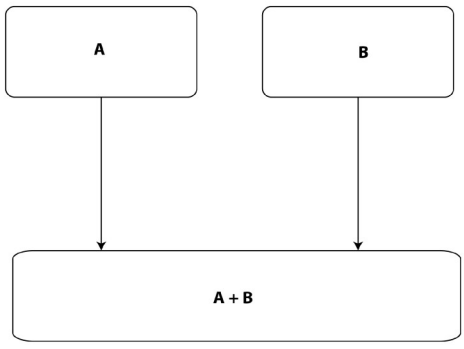


Fig. 2.1 Pipe-filter architecture style

element of the input streams and incrementally computing corresponding elements of the output streams. Connectors, serve as streams conducts transmitting outputs of one filter to inputs of another. Pipe-Filter style can be used to model part of a system with sequential data-flow which filter components will be transforming. Filter components function as a filter transforming flowing data. The connector functions as a pipe conducting data from a sink in one component to a source in other component. Components are sequentially connected.

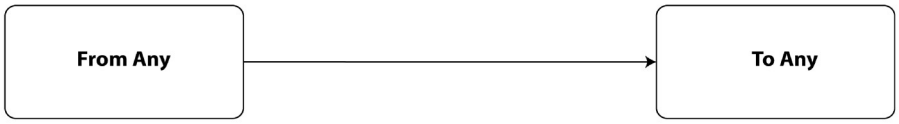


Fig. 2.2 Filter connector behaviour in a pipe line

- The **Client–Server** style enables components called “clients” to send requests to a component called “server”, and expect a reply. The servers' interfaces describe services (or, by and large, functionalities) offered; clients' interfaces describe used services. Clients initialize communication and prepare an answer from the servers. Clients must know server identity, while the converse does not hold; the client's identity is communicated with service query. Connectors represent an interaction protocol including a question and an answer in the base case. It can also prompt clients to begin a session with the service, to respect eventual regulations concerning the order of queries, or close the session.

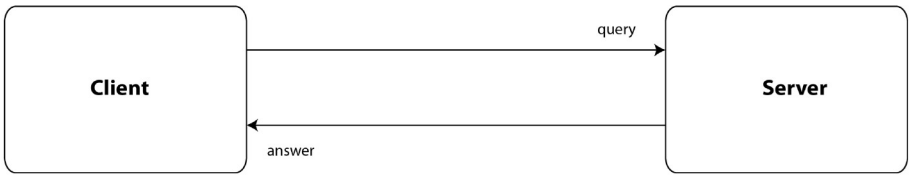


Fig. 2.3 Client-server architecture style

- **Feedback Control Loop** style allows central components and connectors configured to allow a central component control of several actuators through analysis of sensor information. FCL style is usually used to model a part of a system with a central controller to control one or more actuators by using data from one or more sensors.

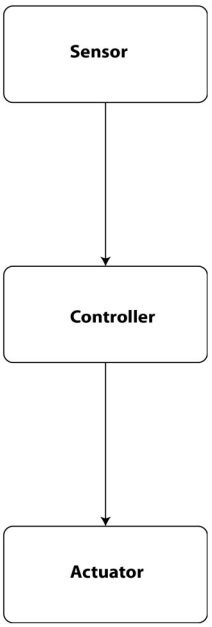


Fig. 2.4 Feedback Control Loop architecture style

- **Shared data** style focuses on access to data shared among various components. It integrates such components as databases, for instance, maintaining shared status and a set of independent components operating on data. Shared status, at least in a “pure” shared data system, is the only communication channel among components. A connector linking the data base with components operating on data can describe, for instance, an interaction protocol beginning with an authentication phase.

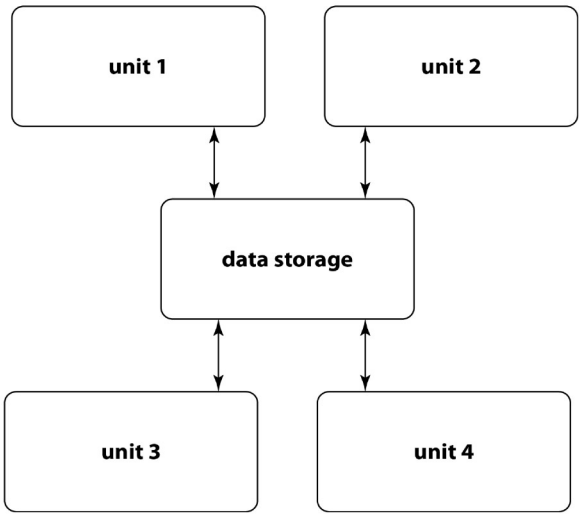


Fig. 2.5 Shared data architecture style

2.6 Design Patterns for Graphic User Interfaces

Regardless of the kind of software architecture being used, the relationship between the software program and the user is mediated by the graphic interface (GUI) through which software programs are enabled to function. If we wished to divide into further levels the shift from user to hardware physical components, we could define four levels which the conception of software architecture should be supporting:

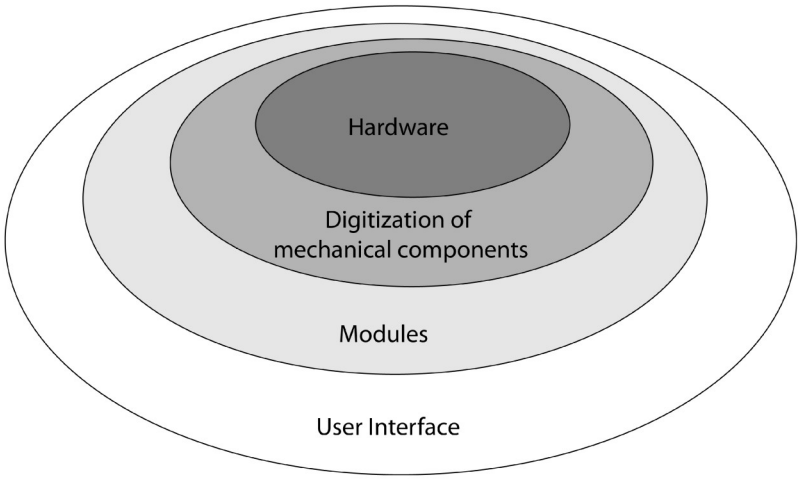


Fig. 2.6 Layered model

Users physically work only with the furthest out layer of the external interface; communication with the underlying layers is more or less implicit and it can be expressed through various kinds of interaction with the layer located further out. The different ways of having the GUI communicate with the user leads to the definition of several possible design patterns. Starting with the Nineties, when software program commercialization and therefore their usability having become crucial in programming user-oriented software, design patterns have been following a parallel evolution with programming languages. The GUI design pattern relies on a different chain integrating three main elements:

- **the module** is the software program's core whose algorithms gather, elaborate and render data from and for users;
- **the controller** enables users to input data;
- **the view** enables users access to software status and data output.

The arrangement of these three elements brings life to different kinds of design patterns, illustrated here below.

2.7 Architectural patterns

2.7.1 MVC architectural pattern

The Model/View/Controller (MVC) triad of classes, first described by Krasner and Pope in 1988[31], is used to build user interfaces in a software called Smalltalk-80. MVC is the first design pattern to be formalized and applied in hundreds of software programs following the graphic interface method instead of scripting as a communication channel between agent and calculator. Controls status change directly influences the model's way of working; this, in turn, sends visible output data through the graphic interface.

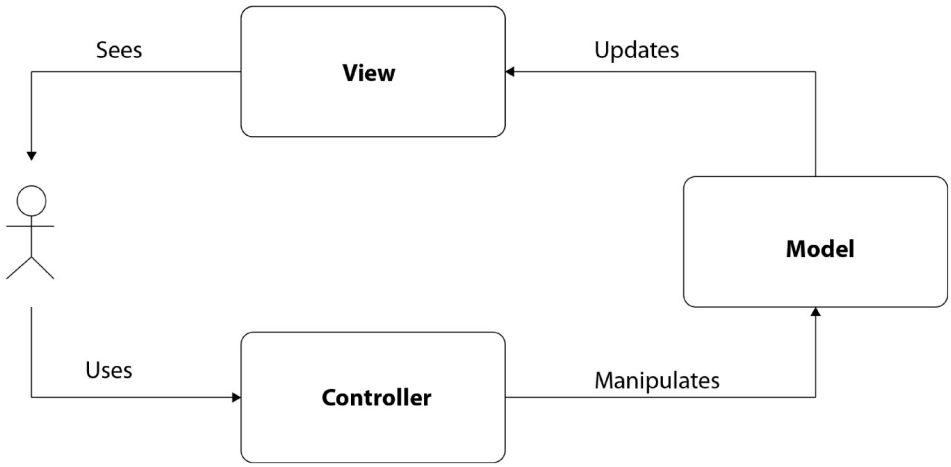


Fig. 2.7 MVC pattern

2.7.2 MVP architectural pattern

"Model–view–presenter (MVP) is a derivative of the model–view–controller (MVC) architectural pattern whose origin goes back to the early Nineties" [32]. It is used mostly for building user interfaces. In MVP, the *presenter* takes on the "middle-man" functionality; all presentation logic is pushed to the presenter. MVP pattern was developed to ensure easier automated unit testing and enhance concern separability in presentation logic.

---

[31] G.E. Krasner, S.T. Pope, *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*, Journal of Object-Oriented Programming, Vol. 1 Issue 3, Aug./Sept. 1988, pp. 26, 49.

[32] M. Potel, *MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java.*Taligent Inc., 1996

While it is the view's responsibility to display model data the presenter governs the way the model can be manipulated and changed by the user interface. This is where the heart of an application's behaviour resides. In many ways, an MVP presenter is the equivalent of the application model in MVC; most of the code dealing with the way a user interface works is built into a presenter class. The main difference is that a presenter is directly linked to its associated view so that both can closely collaborate, fulfilling their roles as suppliers for the user interface for a specific model.

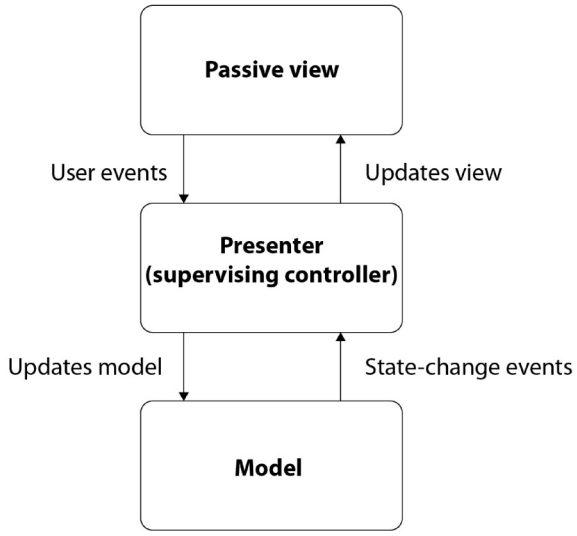


Fig. 2.8 MVP pattern

2.7.3 MVVM architectural pattern

In this design pattern, graphic elements coincide with controls. This design pattern is an improvement of the MVP adapting to those software programs used, in some way or other, in the *question/answer* form.

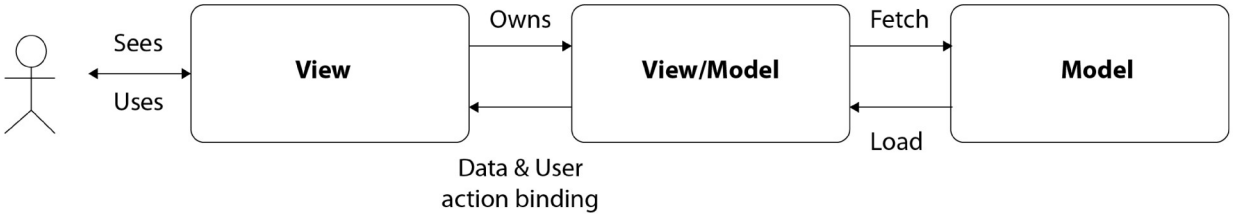


Fig. 2.9 MVVM pattern

2.7.5 PAC, HMVC architectural patterns

"The Presentation-Abstraction-Control pattern (PAC) is an interaction-oriented SA. It defines a structure for interactive software systems in the form of hierarchy of cooperating agents"[33]. Every agent is responsible for a specific aspect of the application's functionality made up of three components: presentation, abstraction, and control. This subdivision separates the human-computer interaction aspects of the agent from functional core and communication with other agents.

PAC recycles basic MVC pattern component characteristics, except in their being structured according to a functional logic where every component gains direct access to the model and univocally responds to a single output. This kind of organization presupposes a functional structure of SA general organization; in other words, components act locally according to a model model-view-controller but in the overall processes they play a defined, multi-layer hierarchical role lending itself to being implemented on several machines working in synergy, where each one plays a defined role in the SA general strategy. A design pattern of this kind unties the concept of SA from the possibility of it being implemented in a single machine; in other words, one potentially responsible for every shift from data input up to the resulting output and up to variable transformation. The key feature of this organization model is the hierarchy between machines and possible multi-tasking.

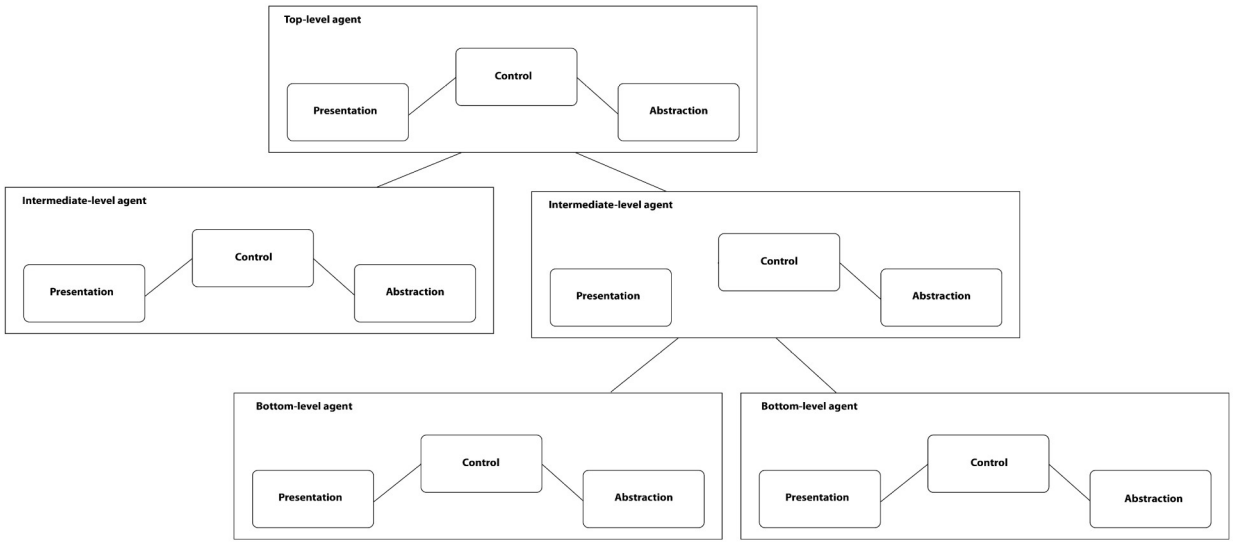


Fig. 2.10 PAC pattern

[33] Q. Kai, *Interaction-oriented Software Architectures*. *Software Architecture and Design Illuminated*. Jones and Bartlett Illuminated., 2009, p. 200.

HMCV pattern was presented in 2000 with the intention of bringing SA development into the field of nowadays so-called “widgetization”, “widget architecture” or, more specifically, “widget-server architecture”[34]. These have opened up the possibility, especially for less experienced users, to combine ready-to-use modules. Users program on their own at the SA bottom level, with no sense of any hierarchy hidden by the SA itself. This pattern does not present any substantial differences with PAC, as the author of a Java magazine [35] article has confirmed. The doubling of the same idea presented with the PAC pattern, conceived in 1987[36] and therefore thirteen years before HMVC, somehow represents the validation of this model as a need to further develop MVC patterns, and to bring the SA on a hierarchical structure capable of supporting multi-tasking and multi-threading.

[34] X. Zhiqing, W. Si, Y. Heqi, W. Zhenyu, C. Hao, Z. Chunhong, J. Yang, *A new architecture of web applications — The Widget/Server architecture.* Network Infrastructure and Digital Content, 2010 2nd IEEE International Conference.

[35] https://web.archive.org/web/20050205080537/http://www.javaworld.com/javaworld/jw-09-2000/jw-0908-letters.html

[36] Coutaz, J. PAC, an Object Oriented Model for Dialog Design. In Rullinger, H. I. and Shackel, R. (eds), Human-Computer Interaction - INTERACT ’87. Elsevier Science Publishers, 1987, pp 431-436

3. Software architecture solutions for real time audio

3.1 From mixer to digital audio software

Most commercially available software programs and visual programming languages used for acoustic and mixed musics are based on *pipeline* software architecture. Taking into account the entire ADC / DAC chain, data and audio streaming shift algorithms in almost linear fashion. Digital Audio Workstations such as Pro Tools, Logic, Digital Performer, Cubase, Nuendo, and a host of others devoted to the structure of every channel/track also draw inspiration for their graphics from audio processing sequences of analog mixers the input signal is entered in, modified in some of its components, and managed in panning to be sent as output to the machine with a final general amplitude (fig. 3.1). Such a scheme replicates assembly chains of analog machines for all of these steps. Thus, a mixer channel scheme sums up wiring rules applicable to audio processing machines. Today's digital mixers integrate communication protocols enabling digital parameter management of all data forwarding from other, external machines. Data communication might be relevant to input as much as output data since midi or osc [37] protocols, for instance, are both programmed for two-way data communication. In most cases, however, digital data are almost always sent as *input* in the mixer, gathering, that is, outside information affecting audio according to a cause/ effect relationship.

Dante protocol[38] enables communication via ethernet by turning the mixer into a component of a broader hardware architecture potentially including more machines communicating with each other in different rooms. Audio mixers' electro-acoustic chain is basically a model adapted in software programs for audio transformation for such post-production purposes as mixing and mastering.

---

[37] It is worth noting the extent to which integration of midi protocols parallels the market launch of digital mixers. New digital mixers often include the integration of the OSC protocol to enable software management from dedicated applications installed on smartphones and tablets.

[38] "Dante is an uncompressed multi-channel digital media networking technology, with near-zero latency and synchronization that has been adopted by more pro-audio AV manufacturers". https://www.audinate.com

The one aspect which DAW mostly develop is the audio transformation section generally implementing algorithms external to software programs (i.e. *plugins*) ready for recall and data forwarding so as to obtain such processing sequences as users wish.
The analog-to-digital transition makes parametric control of each step more fluid since it always occurs within the software program (acting as well as a hardware analog machine), ensuring multiple implementation of signal sendings in the middle or at the end of the processing chain. Since the *inboard* processing sequence is not pre-constituted, programming software for audio basically enable the same operations even more freely: users must not therefore create it each and every time. This enables, on the one hand, the actualizing of two principles:

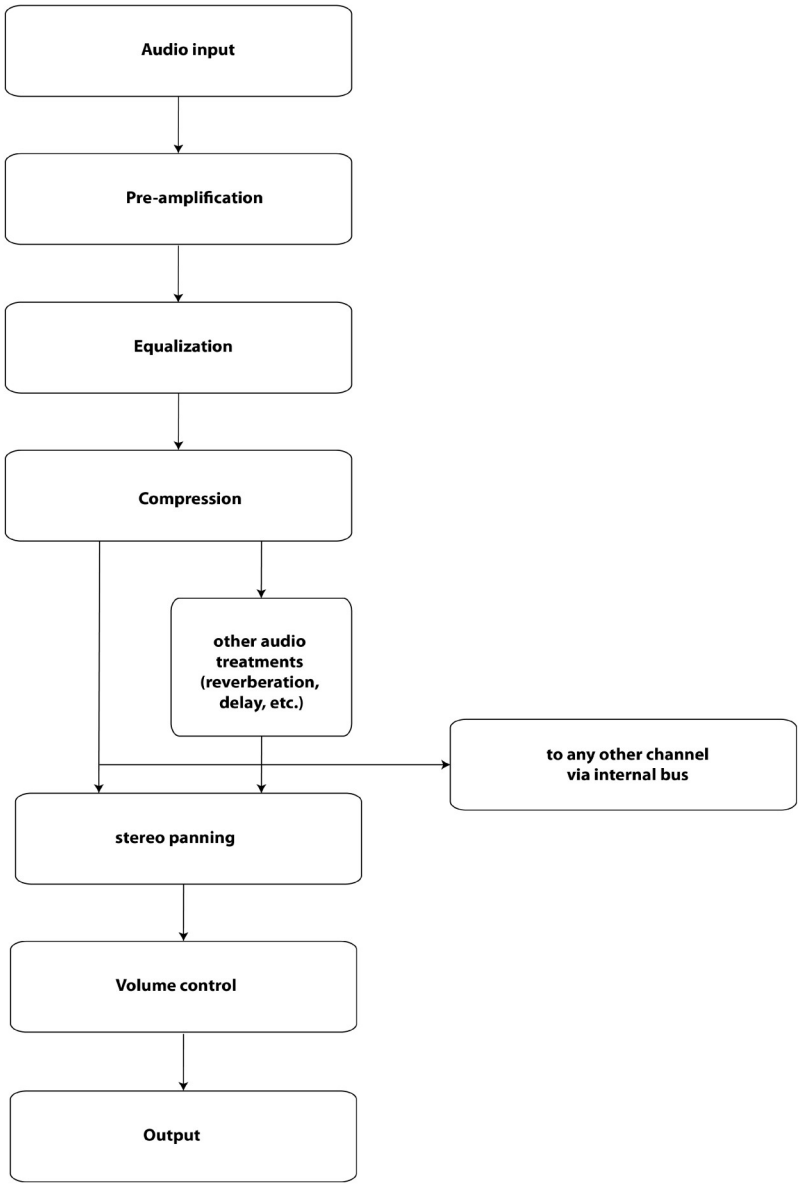


Fig. 3.1 Signal path in a mixer channel

- **economy**: only effectively used algorithms are implemented; only effectively used variables are activated;
- **efficiency**: the chain structure meets the project's local needs.

Obviously, such freedom presupposes skill in terms of how to structure a processing sequence. At any rate, the sequence of operations always leads back to an archetype of *pipeline* architecture, and a more or less complex one depending on the project yet unchanged in its substance.

3.2 Primitive architecture for acoustic and mixed music

By and large, the audio transformation is based on four fundamental steps:

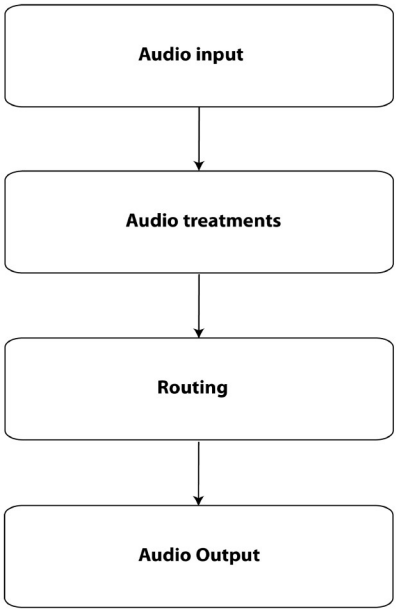


Fig. 3.2 Fundamental steps in pipelined architecture for digital audio

The introduction of *computer scores* since the end of the Eighties substantially established the concept of timeline in pipelined architecture for digital audio for acoustic music in real time and mixed music. Software status basically varies through instruction blocks triggered throughout the performance of a piece so it may change depending on the needs prescribed within a score or predictable external factors. A computer score always and only contains data; it is therefore capable of altering the status of variables from every *inboard* part following audio input and up to audio output (fig. 3.3).

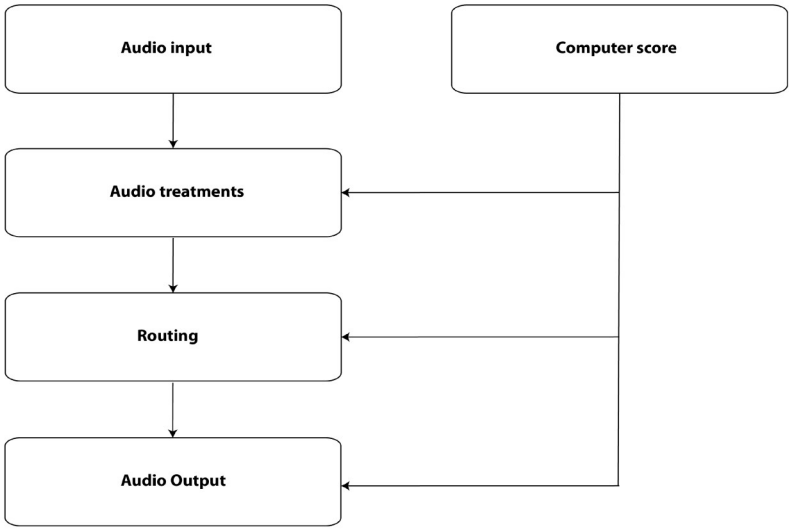


Fig. 3.3 Integration of the computer score

The management of the audio streaming and the computer score is done by the CPU. Producing a software system for real time audio management requires a number of intermediate components with the aim of efficiently performing the task of the software program itself.

Accessory components enabling software program control are likewise needed to ensure its governability. Indeed, a mixer would be ungovernable if it didn't have knobs and faders enabling interaction with the matching variables. Control implementation in design patterns regulating the software program's graphic interface meets the same control needs as outboard machines.

In practical terms, these are hybrid solutions since external control machines are connected to the software program: software functions retrieve the physical interface. These controllers enter data into any one of the points pre-programmed by the pipeline the programmer has built. They can also receive data from the software program matching parameter status. Other machines enabling data send may be responsible for changes in software status over time. This occurs with sensors and all those electro-mechanical machines potentially connected to the computer through digital protocols. At this point, this is what the architecture looks like:

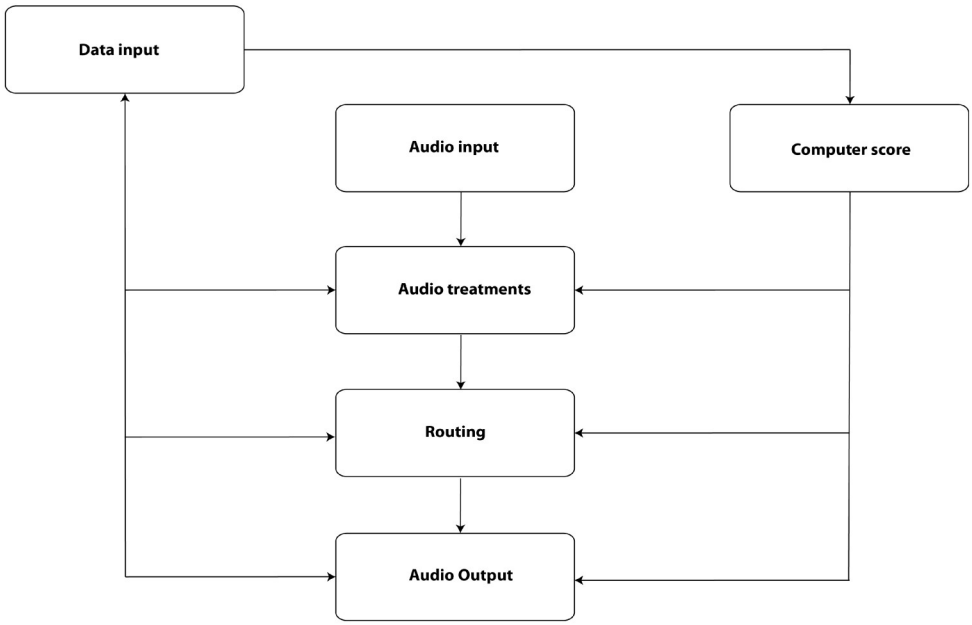


Fig. 3.4 Input data from the outside world

Outside input data can perform the same functions of computer score instructions. The difference is that data from the outside world coming through controls and sensor set-ups are in real time and therefore dynamically unpredictable; computer score data are instead pre-ordered and deterministic leading to the same kind of result whenever the same set of conditions is repeated. A forceful activation of instructions is due to timeline management. Such instructions are regrouped in sub-lists activated by recalling an index number, usually an ordinal and sequential one. Thus, lists of instructions are recalled through the so-called “stage figures” or “cue list”.

The activation block of the instructions lists is therefore mediated by a counter which turns out to be the module effectively communicating with the equipment sending data to the calculator (midi controller, HUI[39], GUI[40], etc. - fig. 3.5) or algorithms in the case of automatic activation as in *score following* (fig. 3.6):

[39] Human User Interface.

[40] Graphic User Interface

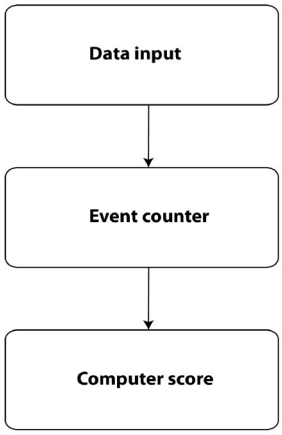


Fig. 3.5 event counter

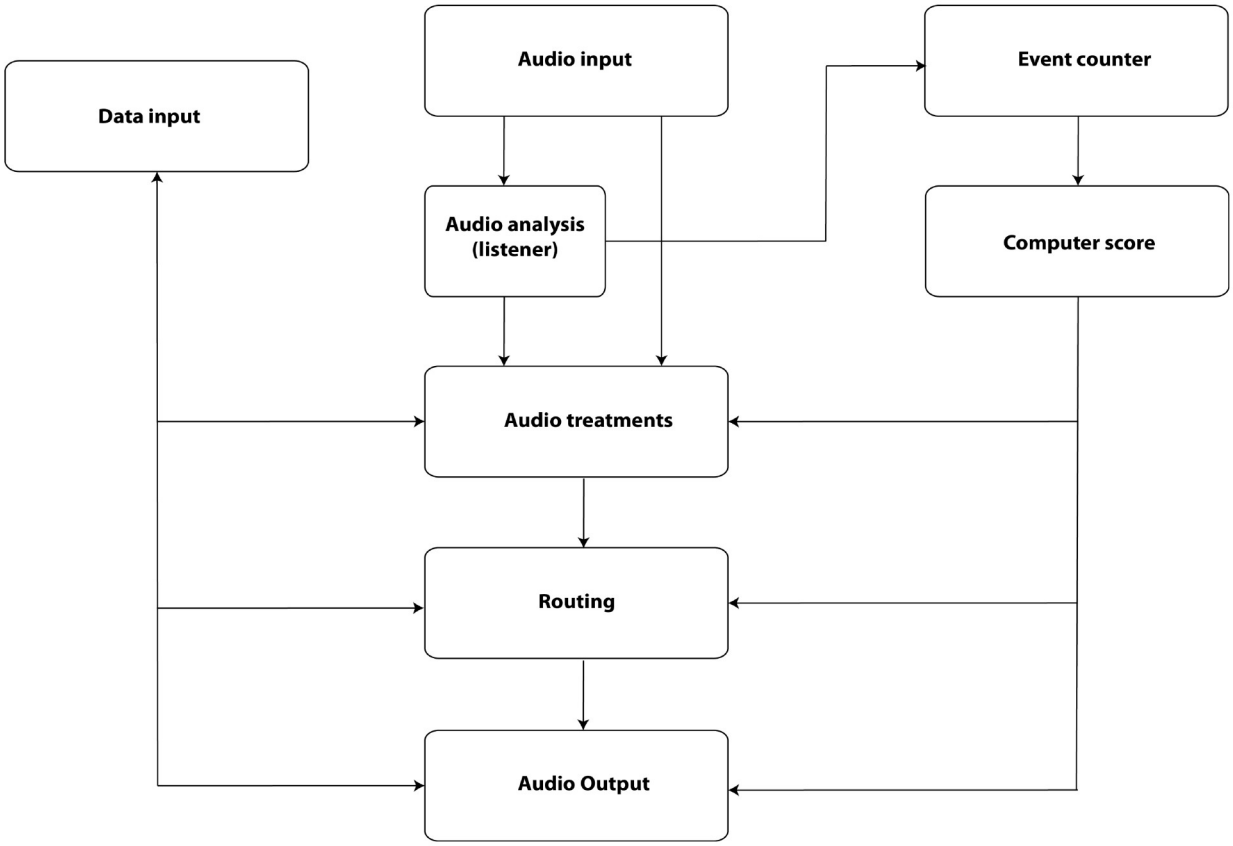


Fig. 3.6 Event activation through score following

Score following of midi instruments is a specific case:

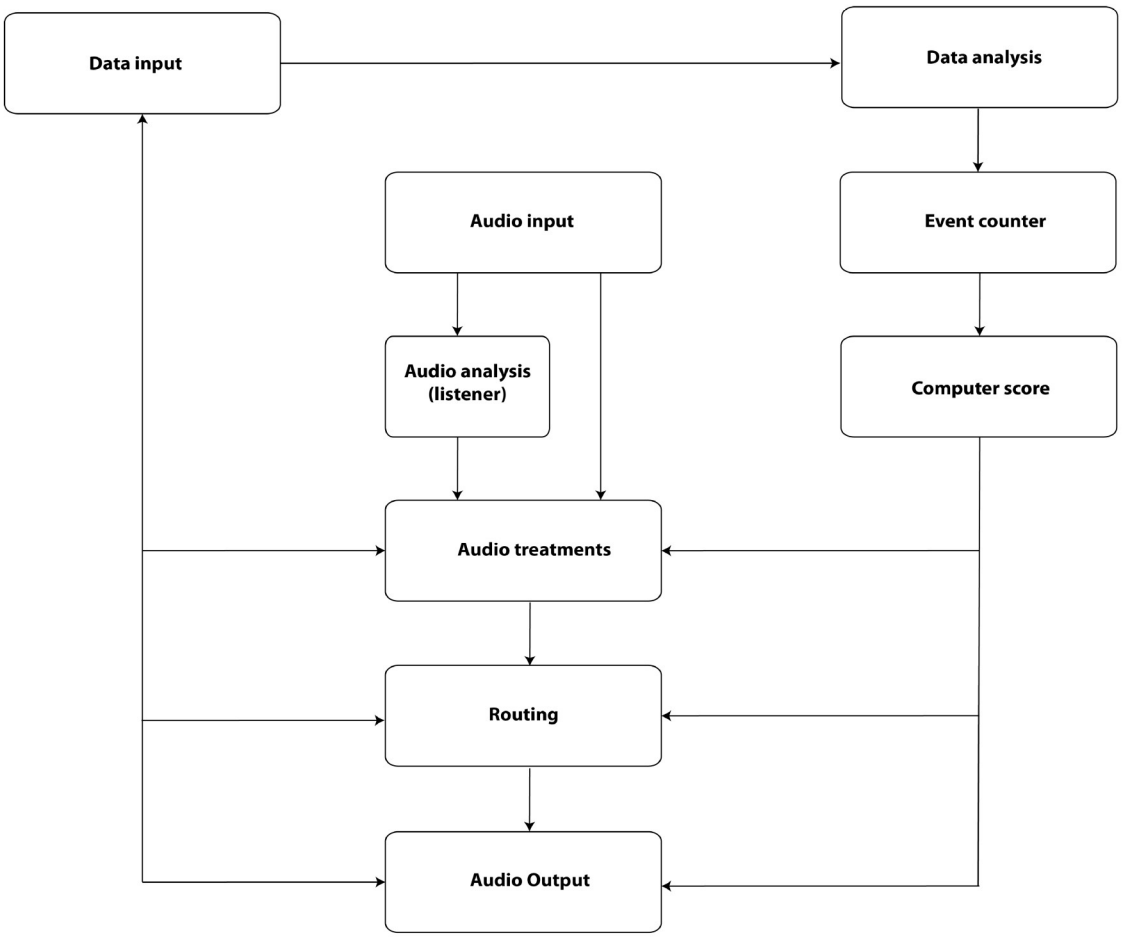


Fig. 3.7 Score following midi

A simpler case, used very often in the mixed music of the 90s, is that in which a midi keyboard, instead of being used as a musical instrument, is used to directly activate events. The activation of each event corresponds in this case to the numbers of the midi notes. In such case the further analysis of the data is not necessary. The model is ultimately that of the sampler; taking into account only audio files, a sampler calls one or more audio files inserted into a memory location associated with a key of the physical interface of the instrument.

3.3 Input sources classification

Audio input sources may be categorized according to three types:

- audio from a microphone: digital audio from an ADC conversion;
- audio file: deferred time digital audio recalled by the computer's memory;
- audio digital generators: various kinds of computer-generated audio on the basis of primitive functions (simple waves) or specific algorithms (as in musical instruments' physical models).

I can therefore speak of audio input *class*, since the three cases indicated occupy the same box within a software architecture for live electronics.

Likewise, data input can occur through:

- Human interfaces: physical or graphic interfaces (the two are often associated) for digital data input;
- midi musical instruments;
- sensors;
- other external machines using digital protocols (i.e. OSC or DMX).

I can therefore complete the previous architecture as follows:

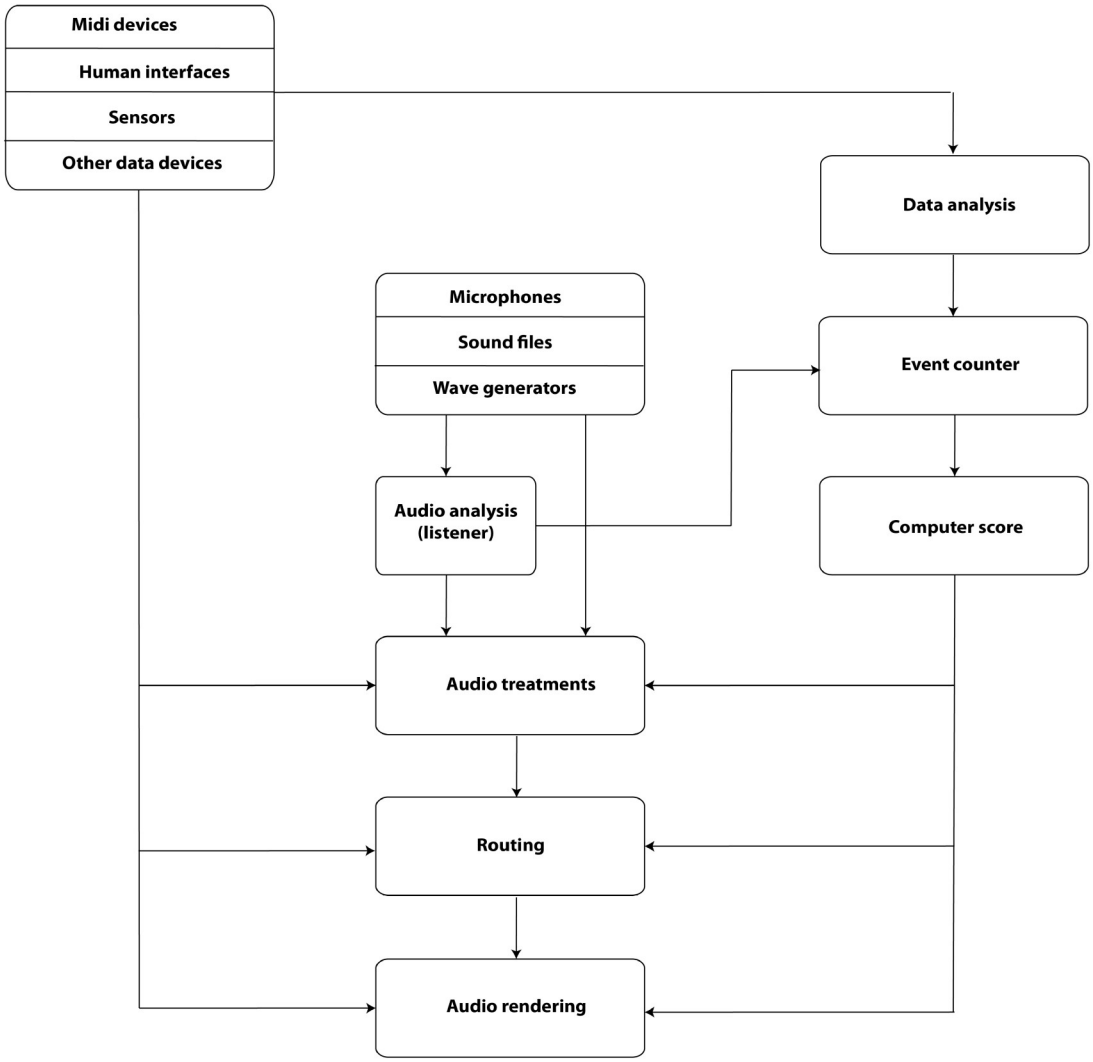


Fig. 3.8: Audio and data generalized architecture

3.4 Output components classification

The aim of a software architecture for acoustic music and mixed music is to have the audio output transformed according to certain procedures in given times. However, it might be necessary to have data output send information mostly for the application of multi-channel audio diffusion algorithms: Vector Based Amplitude (VBAP), Wave field Synthesis (WFS), Ambisonics, HOA, etc., *inboard* as well as *outboard*, that is, for external machines control. This is the reason why it is best to separate the channel routing phase from the audio output phase, which is presented as an *audio rendering* phase - as indicated in fig. 3.8. Routing consists of sending the effective channels in the software

program whereas the rendering adapts the routed channels to the existing ones in the diffusion system.

Audio rendering flows then an audio card physically connected to acoustic speakers. Hardware equipment can in turn be connected to other machines to send information which other software architectures manage, in turn, by working either in synchronization or independently of each other. We shall not be dwelling on such specific cases given that architecture hardware and audio machine wiring are topics that extend far beyond the scope of this dissertation.

Along with partial data output, audio rendering send can flow into a parallel software architecture hosted by another computer through communication protocols between machines: UDP[41], TCP [42] or other protocols currently in use or upcoming.

This is an interesting case as it presupposes the use of more machines in the same audio architecture. Locally, the architecture is always pipelined whereas the general model will tend, instead, to work like a client/server architecture where computer *A* usually sends information to computer *B*. The two (or more) computers may be synchronized to each other, depending on whether they are employed for the sole purpose of dividing the necessary calculations across several machines [43] or assembled with no such need, as in *laptop ensembles*. Various computers can be directed towards different audio cards or the same one, depending on location (whether in the same hall or in a different one) and also depending on wiring choices for diffusion set-up. The use of several audio cards requires the insertion of a synchronization system of the machines' clock in the software-hardware chain: one is usually configured as *master*, the other ones as *slave*.

Here is a possible example of architecture between two machines. Such architecture have a number of variations on local components depending on the kind of project: it is therefore only a general model only to be adapted on a case-by case basis where the adoption of this architecture is necessary:

---

[41] User Datagram Protocol.

[42] Transmission Control Protocol.

[43] This particular case is increasingly less adopted today since modern computers are powerful enough not to need to share the operation load across several machines. Until a few years ago such a solution was often necessary since CPU power was often not enough to guarantee reliable performances in situations featuring live electronics with real time audio analysis requiring a significantly high number of calculations.

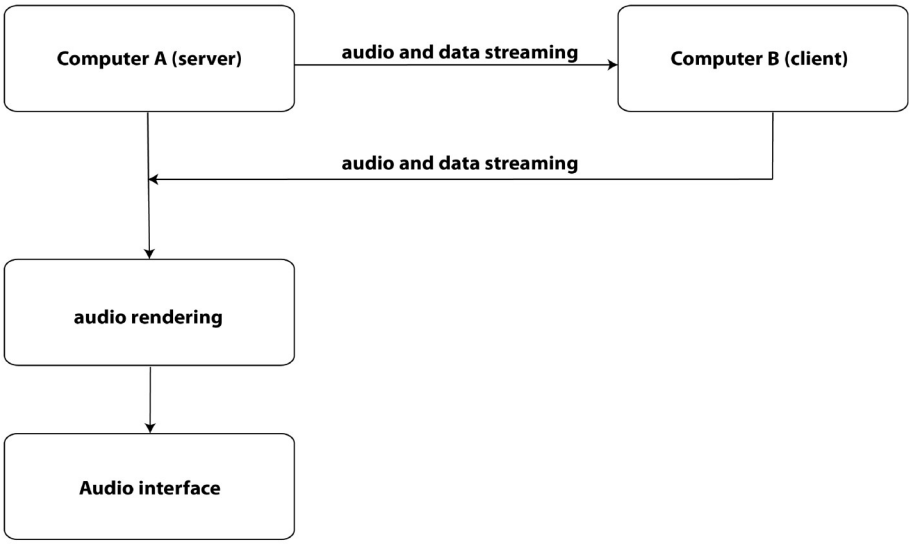


Fig. 3.9 Two-computer client/server architecture

In laptop ensembles signal and data send is reciprocal between various computers: they are therefore nodal points of a *flat* architecture, having neither master nor slave up to the audio output. It is only at this point that a master needs to be established in order to synchronize every machine's audio-clock on a single computation algorithm. For one last small, final part, we go back to a client/server system, albeit for a single operation. In the case of performances with computers located in different spaces (with audio send through ethernet cables or, as is more often the case, through the web) clock management for audio synchronization becomes crucial.

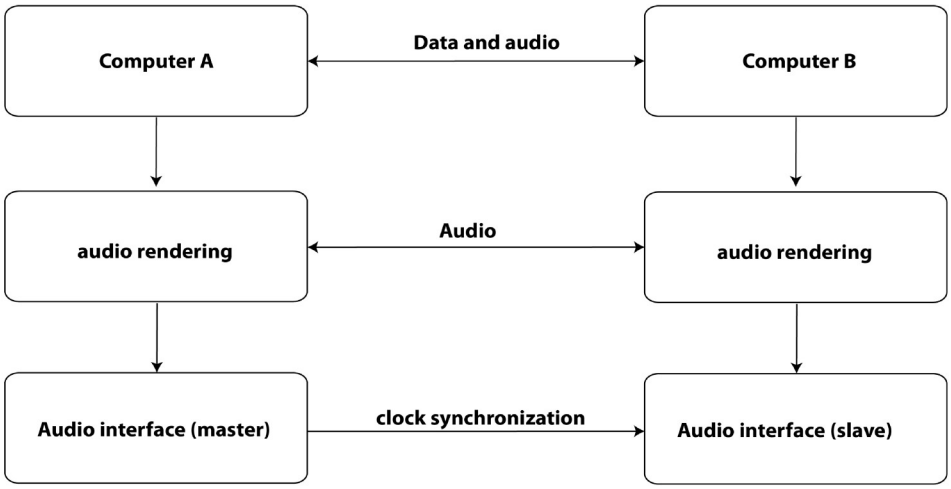


Fig. 3.10 Example of laptop ensemble architecture

4. Collection of Max objects and software programs for Mixed Music

In the past fifteen years, numerous software programs and Max collections have been released to simplify the programming process of live-electronics part coding of a piece of music. This area generally deals with acoustic and electronic sources interaction. All systems have evolved and now face different challenges and have different goals. This chapter examines five collections designed for Max use:

- BEAP modular
- BEASTtools
- Clef
- Jamoma
- Najo Max Interface

Three stand-alone applications for mixed music as alternatives to the use of Max are also described below:

- P-Soft
- Integra Live
- Usine Hollyhock

Max object collections ensure ready-to-use solutions for a number of data and audio processing needs through patcher programming phases. Stand-alone software programs enable programming computer sections of a piece of mixed music with possibilities which the software program itself makes available. They are always compatible with Midi and OSC protocols.
The attention which, in the last ten years, many university departments have devoted to these issues and especially to the release of these collections and software programs is evidence for the need to create software and middleware to make computer section programming of a piece of mixed music easier. These solutions differ significantly in terms

of degrees of personalization, possibilities to combine parts of the middleware itself or third parties and whatever else they might enable.

4.1 BEAP Modular

Developed at the Berklee College in Boston, Berklee Electro Acoustic Pedagogy Modular fig. 4.1 is a Max module collection fully compatible with Ableton Live through the Live Max library. Matthew Davidson, a former employee of Cycling ’74, developed this project with a team under his supervision. Graphics are fully compatible with the latest Max and Live versions; the blocks quality sets limits to customization possibilities to expert Max users only.

“This collection bridges Live's user-friendly interface with the unlimited possibilities of patching through Max for Live. The use of the BEAP modules supplies to the need of patching, running the concept to build a modular and customizable synthesizer in Live through Max”[44].

Fig. 4.1: BEAP modular modules

[44] https://www.ableton.com/en/blog/beap-powerful-modules-max-live/

Since the release of Max 7, BEAP modular blocks can be recalled through an icon on the Max window frame. The collection covers modules for control, data and audio processing, mixing, diffusion and sequencing. Users can daisy-chain blocks through input and output connections.

4.2 BEASTtools

BEASTtools is a fully modular cross-platform environment for 2- and 8-channel format sound exploration and processing. It was developed by Daniel Barreiro, Pete Batchelor, Eric Bumstead, James Carpenter, Alex Harker, Jonty Harrison and Chris Tarren with special thanks to Ben Thigpen.

The system includes a number of processing 'tools', each with specific functions, and the possibility for daisy-chaining so as to form a work environment. BEASTtools also features facilities to accept live audio streams, host VST plugins, up to 8-channel recording, and controlling channel routing for different software programs and applications"[45].

This collection's concept is similar to BEAP modular's: a collection of modules works in *bpatcher* frames for users to place on a Max patch. Users can daisy-chain blocks; processing in each module is defined in the collection while the final chain depends on users' choice. Each module is open and may be modified or integrated by other modules. This system requires some basic patching skills.

Fig. 4.2: BEASTtools ensemble overview

[45] http://www.birmingham.ac.uk/facilities/aestudios/research/beasttools.aspx

4.3 CLEF

CLEF (CIRMMT Live Electronics Framework, 2009-2014) is a Max-based modular environment for composition and live electronics performance, developed by Marlon Schumacher of the CIRMMT/McGill University. Ideas for the development of this package draw inspiration from the Integra Live's GUI. CLEF bridges features of a module collection with the hard architecture of stand-alone software. It has no external objects, libraries or complied codes; users work in a standard Max patch. The package works as a collective application and provides a specific menu within Max applications. This enables easy recall of the main windows to add modules, store and play. “CLEF separates domain model from data access layer and user interface, relying on dedicated technologies: OpenSoundControl for data access, Max dictionaries for module description and *pattr* for storage management”[46].

The system requires division of the electronics in cues and events much like *qlist*. Data storage is separated from audio processing. A number of sub-windows make the controller section more user-friendly: the most relevant features allow drawing lines in a timeline to define parameters and the main routing can be displayed as a flowchart [fig. 4.3]. A collection of .vst objects completes the package. Since none of the patches is editable, users cannot customize the collection.

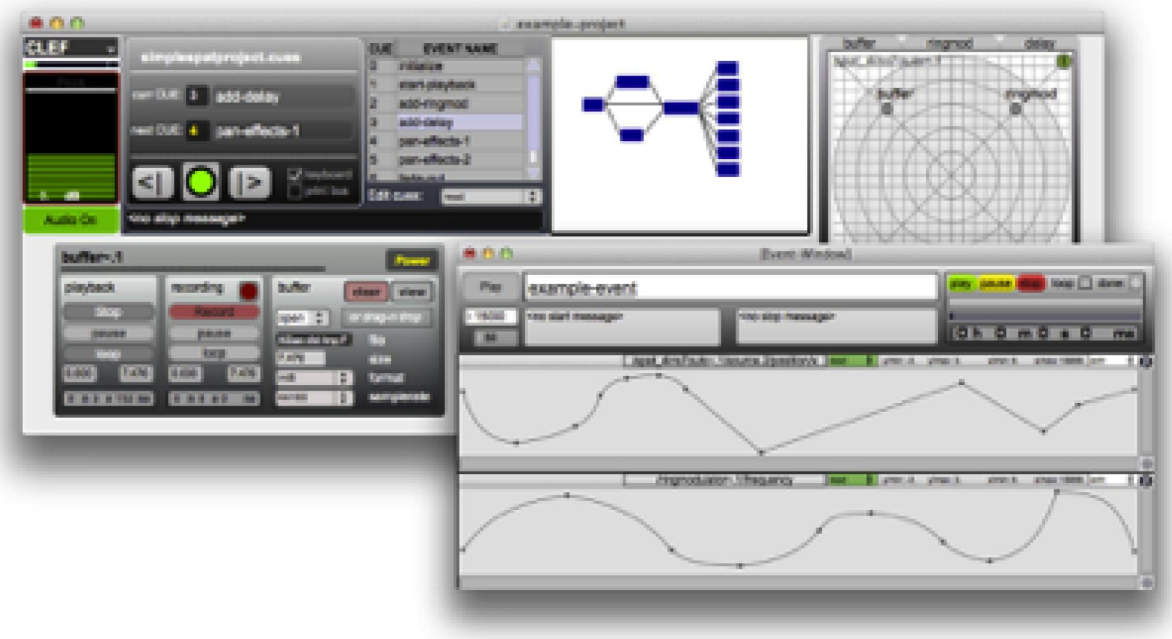


Fig. 4.3: Main panel and CLEF widgets

[46] http://clef.sourceforge.net

4.4 Jamoma

An open source project released as a Max package, Jamoma (2003-2016) was developed by a team of programmers and supported by Norwegian and Canadian institutions. As a work in progress, it welcomes third party contributions. The last official release is the 0.5.7, tested for Max 6. Like the previous ones, this one owns a series of abstractions to be filled in some bpatcher objects and chained as desired. Graphics enable users with a basic knowledge of Max to create audio processing. The patch collections covers control, audio and video processing. The version 0.6 is in alpha state and stears the project into another direction. An own syntax and way to recall and use modules appear on the overview of this release. Users may use existing modules or create new abstractions following the logic of this system. More extensive background in patching is required.

Jamoma Core provides a set of layered C++ frameworks and extensions to create an object model, preparing it for advanced purposes such as audio and graphics. Jamoma Core can be used with a wide range of hosting environments. So far, development was geared mainly towards the use with Cycling'74 Max, but example code exists illustrating how it can be used with other environments such as PureData (Pd), AudioUnit plugins, and iOS. Communication between environments runs through the OSC protocol.

Jamoma is downloadable in its version to be used in Max as well as in the version suitable for PureData. C++ object codes are publicly made available through free downloads in order to enable further development of both new and existing modules. Upcoming updates ought to expand the collection for communication with CSound, SuperCollider, Open framework and VST[47] plugins.

[47] *Virtual Studio Technology* is a plugin standard developed by Steinberg. https://www.steinberg.net/

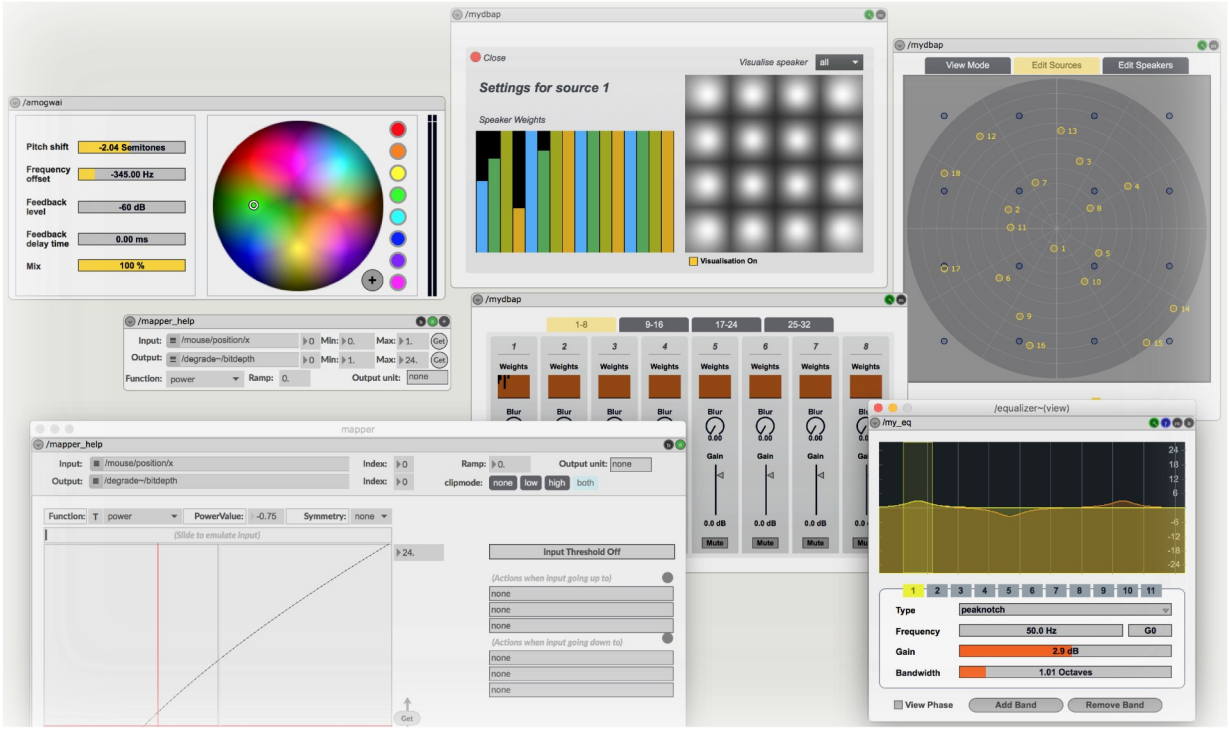


Fig. 4.4: A few Jamoma modules

4.5 NAJO Max Interface

Najo Max Interface (also called Najo Modular Interface) (fig. 4.5) is a free-download Max package of audio modules by Jean Lochard (Ircam, Pedagogy Department) with contributions from other IRCAM[48] colleagues of his. It offers easy access to many of the processing techniques developed at IRCAM without requiring exceptionally strong patching techniques. Najo Max Interface versions prior to 2014 were released as a stand-alone application. The present v2.4 March 2018 release is totally organized by modules and strives for similar goals as other packages. In NMI it is not necessary to create a bpatcher to implement a module. NMI modules can be directly instantiated in a patch by choosing from a menu. The system works in a standard Max patch. Modules may be modified and merged with custom parts.

[48] *sogs~*, *psych~*, *supervp.scrub~*, *supervp.trans~*, *yin~* by Norbert Schnell SuperVP engine by Axel Roebel irc.shell, *spat.oper*, *spat.spat~* by Thibaut, Carpentier
moogfilter~ by Thomas Hélie, Jean Lochard, Thibaut Carpentier, *najo.multipan* by Jean Lochard, « *The jimmies* » by Zack Settel. NMI documentation, 2018.

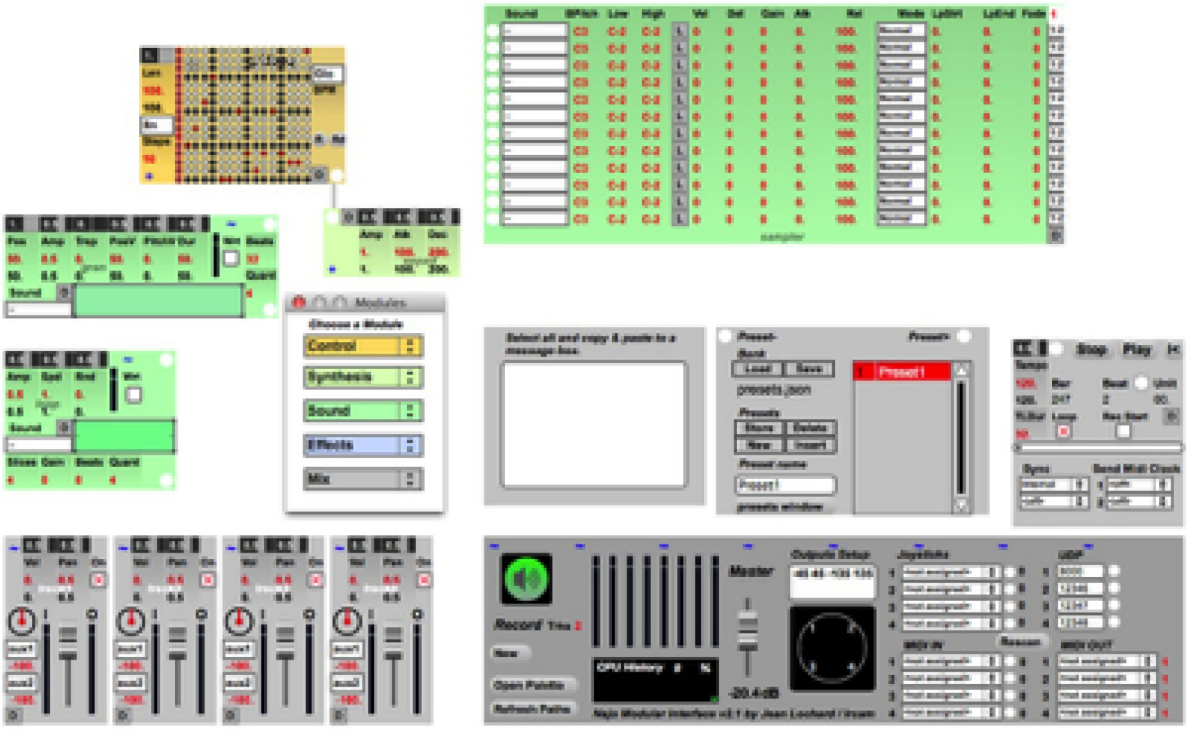


Fig. 4.5: NMI modules and main menu (at the center)

This collection includes more than ninety modules providing solutions for audio synthesis, reading samples, sound processing and data control. Users can recall modules and daisy-chain them freely. The focus of this collection is set on non-expert Max users and provides a *ready-to-use* series of modules for control, synthesis, sound diffusion, processing and output mix. The starting project patch, the division by function on the menu and a number of I/O modules graphics challenge users to build their own modular workstation. The package can be used for both studio creations and for live processing. Users connect modules building their own audio processing chains. The standard output block offers as default the possibility of spatializing sounds up to four channels. The package provides for free some Max externals normally released in commercial license[49].

4.6 Integra Live

Integra Lab of the Birmingham Conservatory in UK developed Integra Live. This software program was originally part of the EU-funded Integra Project, directed by

[49] http://forumnet.ircam.fr

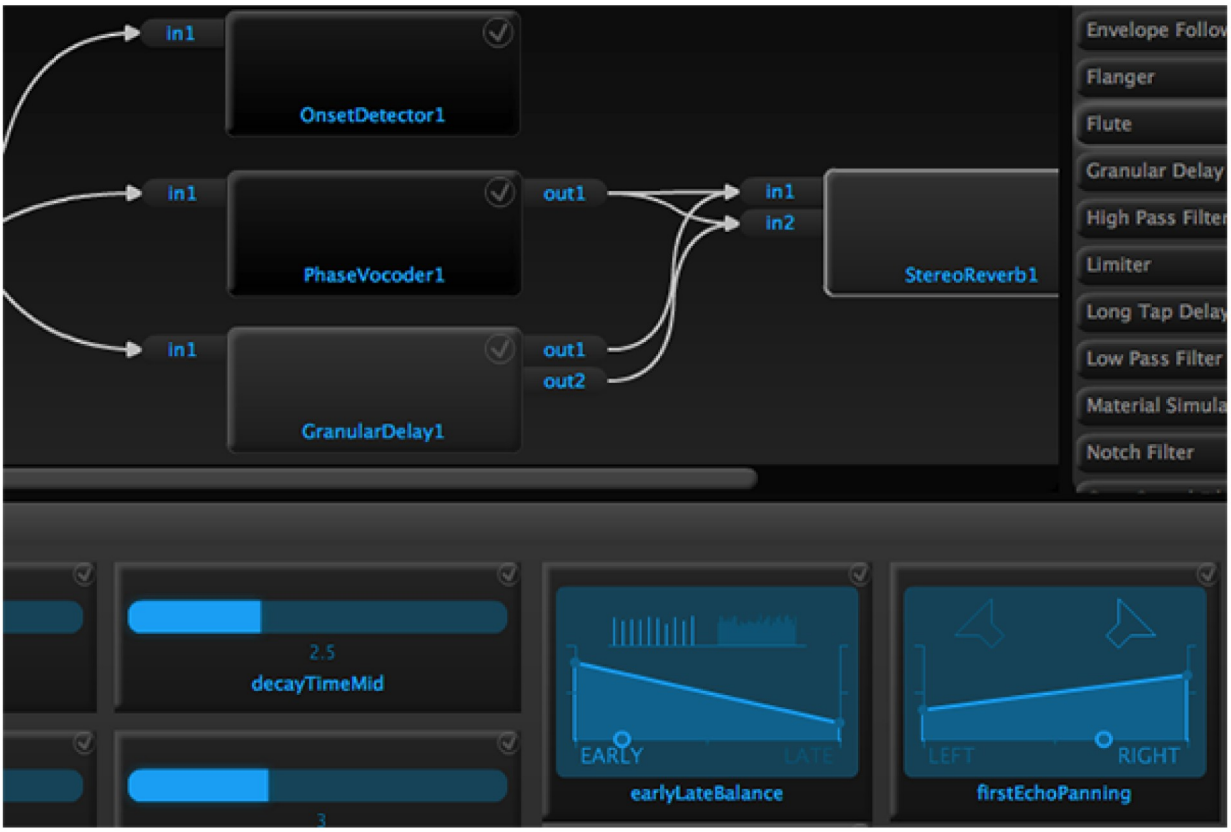


Fig. 4.6: window with blocks connection in Integra Live

Lambert Coccioli of the Birmingham Conservatory and supported by the European Union Culture 2007-2013 program. Integra Live features built-in scripting facilities based on the Lua programming language[50]. Scripts can be used to set and get parameters and perform a range of procedural operations on them. As a stand-alone application, it provides a modular succession of processing, hierarchically organized into three levels: tracks, blocks, and modules. Integra Live can also host several tracks in a single file. This software program works through two main views: Arrange View for editing and the Live View for performance. The Arrange View is the application's main view. It enables audio processing blocks to be created and arranged on a musical timeline.

A block is a container for adding and connecting modules. Blocks are used to organise modules into musically meaningful groupings; they represent what other platforms designate as events or scenes. Tracks enable multiple block activity at the same time.

[50] Lua supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode with a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping. http://www.lua.org

The Module View is a sub-view of the Arrange View. A module can be used to process, generate or analyze musical sound in IL. Modules contain standard effects and the most common processing. Modules parameters communicate via Midi. All Modules are locked. An extra-package can be downloaded to build custom Integra modules. IL includes all operations in a master timeline. “The master timeline provides a reference point against which musical ideas can be organized. Timeline progression can be linear, where the ordering and duration of blocks corresponds to their ordering in performance, or non-linear, where the playhead moves to arbitrary points on the timeline with some blocks being activated indefinitely or stopped and started through user interaction”. Scenes are used to create user-defined progressions through musical time. The graphic interface is user-friendly and the software provides to possibility to customize the colors.

4.7 P-Soft

Fig. 4.7: P-Soft software interface

This software program developed by Alexandr Mihalic is promoted by the MUSINFO[51] French organization; its beta version is required for International Composition

[51] MUSINFO Association was founded by Mikhail Malt, Alexander Mihalic and Laurent Pottier following the creation of the research group Musinfo (the first group created in the AFIM) with a vocation for reflection, publication and teaching on the theme of formalization of music (calculation of music).

Competitions running for five years [52]. “This software program works as an extension of acoustic instruments. It is distributed with the objective of letting users program an environment for effects processing and write scores for acoustic instruments in real-time. This software program is inspired by the LXP15 effects processing device, or one of its algorithms”[53], to be more accurate.

P-Soft runs as a blocked Max/MSP patch in presentation mode. The main windows host three panels, matching the application's main function: pedals, effects editor and parameters.

Pedals simulate real pedals via slider to trigger the audio. Each “pedal” is linked to a parameter to be chosen from the Effects editor window. Parameters can be freely assigned and change colors when activated (the whole interface only uses three colours: black, yellow, and green). The parameters panel contains the basic settings for audio, MIDI and banks storage. Parameters also provides I/O output settings. The amount of tools for audio processing provides delays, filters and reverb. P-Soft uses Max as a platform to build a closed environment on the concept of stand-alone software. Midi communication is possible as long as it follows strictly the path described in the documentation.

The project behind programming this platform lies in the availability, for the community of musicians, of a long series of pieces from the mixed music repertoire of the last fifty years, gradually being “transcribed” for performance via this platform.

The *Sampo* hardware interface enables retrieving and diffusing mixed music through this work of digital restauration and update of accompaniment technique of electronics to live instrumental performance. It also enables microphone and speaker wiring (not included in the physical interface) as well as the volume pedals controlling the parameters which performers are free to change manually.

The current version of P-Soft has a limited number of audio effects since its use is specifically designed for performance of mixed music pieces from the past where live processing possibilities were extremely limited: most mixed music pieces were essentially diffusions of audio files with amplified live instrumental sections.

The audio processing chain architecture available in P-Soft is clearly explained in the instruction manual included with the software program:

---

[52] http://www.musinfo.fr/fr/creation/concours/concours-2018

[53] http://www.musinfo.fr/index.php/en/2014-contest/software-and-manual#Software

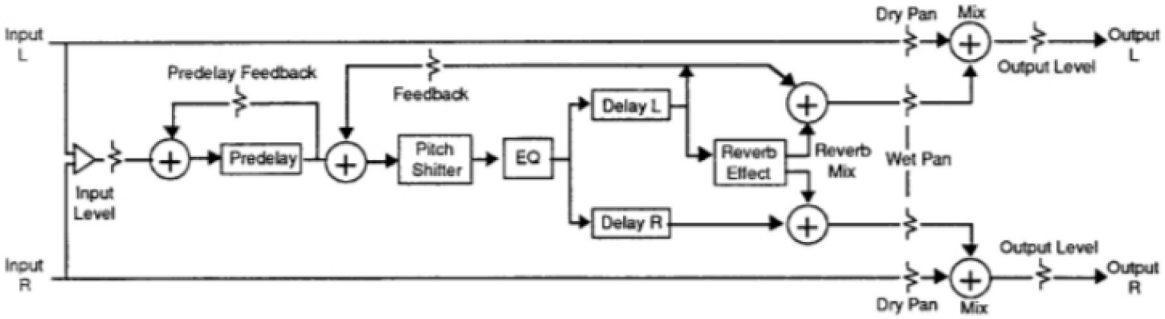


Fig. 4.8: Processing architecture in P-Soft

Fig. 4.9: Sampo instrument

Electronics for a piece may be downloaded via the internet and installed in *Sampo*, ready for use by performers. At the Ircam 2018 Forum, Musinfo's team announced that starting in 2019 P-Soft will be able to host user programmed audio processing external to the software (assuming that such additional modules are programmed in Max in order for them to ensure compatibility).

In June 2018 a piece of mine called *Pulse(s)*[54] for Alto Saxophone and Sampo was premiered in Bourges by Serge Bertocchi within the frame of the festival “Music and technology”.

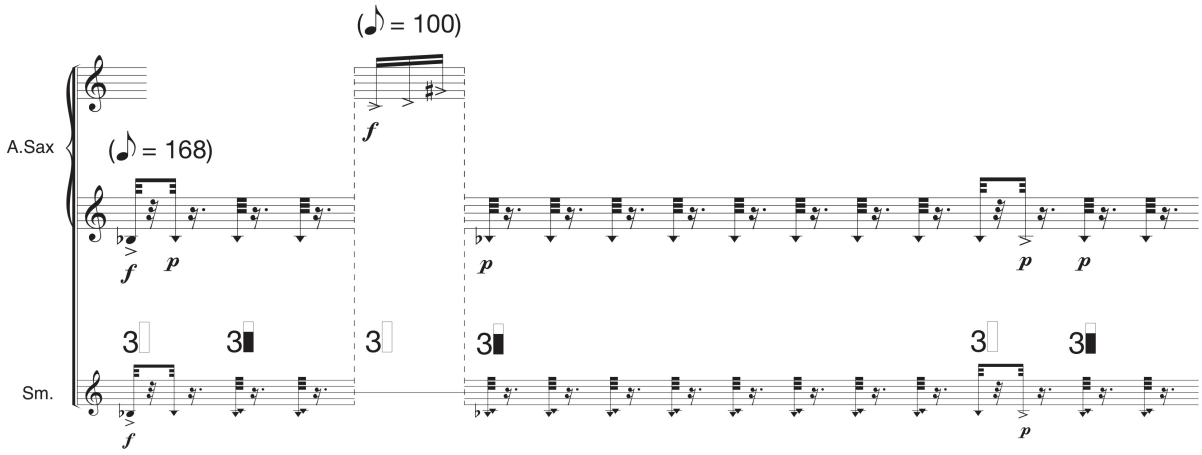


Fig. 4.10: Excerpt from *Pulse(s),* for alto Saxophone and Sampo, pag. 2

4.8 Usine Hollyhock

A complete environment for audio processing, Usine Hollyhock was developed by Oliver Sens and the Sensomusic team. Like Integra Live, the software program features a specific graphic interface and organizes all features through several panels and layers. Usine Hollyhock supplies model routing in the HMVC design pattern through the Object Programming. The main panels bridges the Modules with the View and Controls, but their combination is up to the user’s mental model (fig. 4.11).
The software has a free limited version and a commercial release providing more features. Usine Hollyhock is a true DAW dedicated to mixed music. It includes five-hundred-and-eighty modules for audio, MIDI, video, DMX, OSC processing and creation of interfaces for personal control. Like Integra Live, Usine Hollyhock modules may be programmed following a timeline grouping pre-programmed processing in the guise of scenes.

[54] M. Cacciatore, *Pulse(s), for alto saxophone and Sampo*. Milan, Edizioni Suvini Zerboni, S. 15667 Z., 2018.

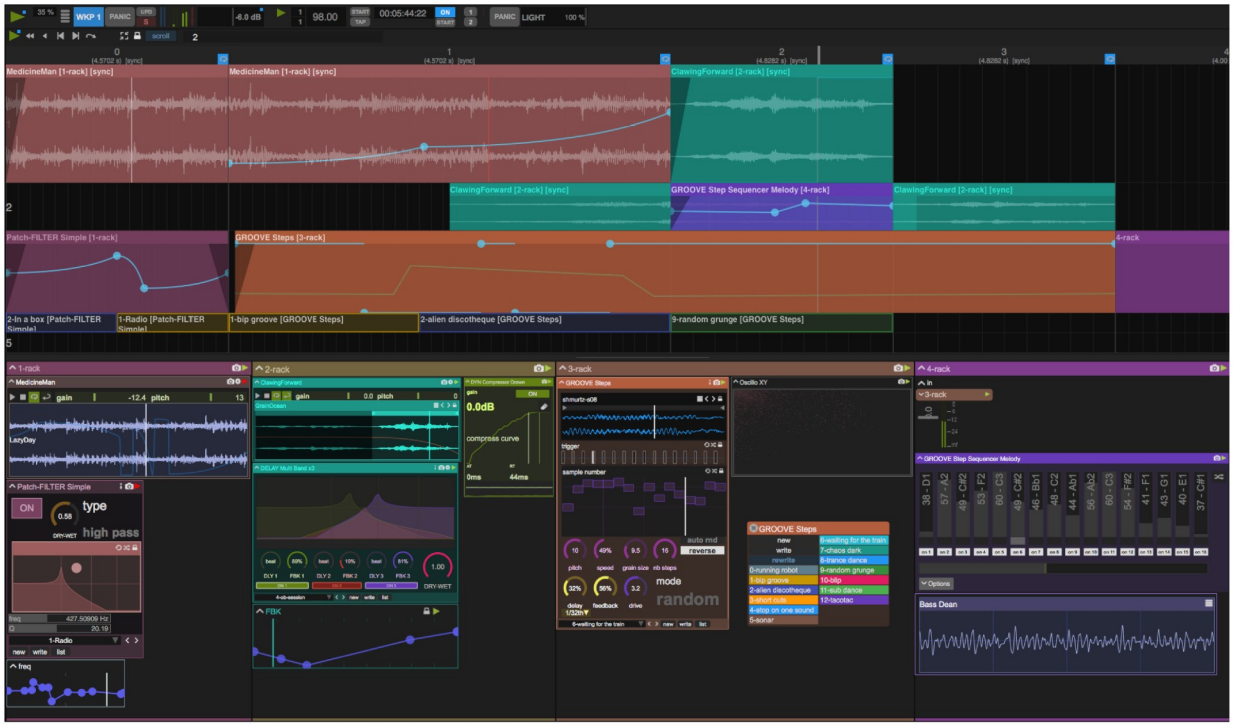


Fig. 4.11: An overview of Usine Hollyhock

4.9 Quality comparison

Figure 4.12 examines modular collections usable in Max or in other compatible platforms. These collections' strategy is quite similar as well as the features offered by each of them. It also offers audio processing under the guise of pre-programmed modules available to users through a user-friendly graphic interface. In it, users only work with the parameter controls of algorithms hidden behind the graphic interface (the so-called Max “presentation” mode). Users work with the modules such as they are, combining them more or less freely. The degree of freedom in module arrangement depends on the proto-architecture supporting module programming such as the package producer designed it. Clef and Jamoma module structure tends towards the construction of a work station within a Max patch, with a pre-ordered logic in signal conduct and data processing. This is not, in itself, a negative aspect since it contributes to programming linearity and clarity in sound signal conduct both before and during DSP. From the standpoint of Max expert users programming freedom, the impossibility to control every step of the ADC-DAC chain is a significant shortcoming.

	OS	Platform	Components	External code	Customizable	Modules content closed or open	Proto-Architecture
BEAP Mod.	Apple OS Windows OS	Max	Modules	No	No	Open	No
BEASTTools	Apple OS	Max	Modules	No	No	Open	Yes
CLEF	Apple OS	Max	Objects & Modules	Yes	No	Closed	Yes
Jamoma	Apple OS Windows OS	Max / Pure Data	Objects & Modules	Yes	No	Open	Yes
NMI	Apple OS	Max	Modules	No	No	Open	No

Fig. 4.12 Summary overview of collection of Max objects features

Similar comparison criteria are adapted for the three stand-alone platforms examined here, as we can see in Fig. 4.13:

	OS	Platform	Components	External code	Customizable
P-Soft	Apple OS	Max	Parameters	Yes	No
Integra Live	Apple OS	Max	Modules	Yes	Yes
Usine Hollyhock	Apple OS	Max	Modules	Yes	Yes

Fig. 4.13 Summary overview of Mixed music softwares features

P-Soft enables, through the graphic interface, control of some of the parameters of audio processing such as it is pre-programmed in the platform. Customizing possibilities are reduced to a bare minimum; however, as I have mentioned, such needs are not the priority for programmers considering the project's purpose behind software production (and the hardware the software program is dedicated to).

Integra Live and Usine Hollyhock are a DAW attempt dedicated to mixed music where users can freely connect audio processing modules building their own processing chain and governing the graphic interface through MIDI or OSC protocols in full autonomy.

Usine Hollyhock can also host third parties plug-ins. Fig. 4.14 takes into consideration more general parameters relevant to both Max module collections and stand-alone software programs:

	Ch.	SR (kHz)	MIDI OSC	Video	DMX	Hosts VST	Architecture	Architecture	Design pattern
BEAP Mod.	64	96	No	No	No	No	No	Pipelined	MVP
BEASTTools	64	96	Yes	No	No	No	No	Pipelined	MVVM
CLEF	64	96	Yes	Yes	No	Yes	No	Pipelined	MVC
Integra Live	32	192	Yes	No	No	No	Yes	Pipelined	HMVC
Jamoma	64	96	Yes	Yes	No	No	Yes	Pipelined	MVP
NMI	64	96	Yes	No	No	No	No	Pipelined	MVP
P-Soft	2	96	No	No	No	No	No	Pipelined/ client server	MVC
Usine H.	64	96	Yes	No	Yes	Yes	No	Pipelined/ client server	HMVC

Fig. 4.14 Summary of parameters for Max collections and standalone softwares

Classifying architecture types and design pattern types to build the graphic interface is, in each and every case under scrutiny, the result of empirical observation of structure and work mode. All software programs include pipelined architecture since, once input (data, audio, video) is entered, and following a cascading processing series, the result is an output depending on the chain itself.

P-Soft and Usine Hollyhock are the only software programs including a separate section from the software with a client server type architecture. Some content is downloadable from a server to make installation on the user's local computer easier. In P-Soft, audio files and presets for electronics of the pieces are “transcribed” for this platform. In Usine Hollyhock, instead, audio and video components, presets and templates are downloadable on request.

As far as the selected design patterns are concerned, Clef and P-Soft divide the section hosting the algorithms of the control section. Controls communicate with the modules, affecting graphic interface status. It is a classic, MVC design pattern. Access to modules including algorithms is not visible. In the other collections usable in Max, they therefore include an MVP design pattern. The only exception is BEASTtools, where the module control presentation space sometimes also hosts the computer's answer after data output, typical of MVVM-type design patterns.

As stand-alone applications and by attending to various needs, Integra Live and Usine Hollyhock inevitably include un multi-layer design pattern. ad hoc solutions are chosen for every section, although some of them will be different from those for other sections in the same software program.

Integra Live, for example, features a sharp graphic split for the audio control section compared to the one hosting connections between various processing modules. In this sense, it acts like an MVC interface. The timeline all modules are tied to, also responsible for activating and dis-activating all processes, is assigned a hierarchically superior priority compared to the modules, all of which are required to answer to the timeline itself as far as their on and off status is concerned. This accounts for the reason why the structure is closer to an HMVC interface. Usine Hollyhock also makes a similar approach clearly visible. While Integra Live features a dynamically changing interface following recalled sub-windows, Usine Hollyhock relies upon a structure with pop-up windows displayed once recalled though their closing can only be done manually. Usine Hollyhock pop-up window structure is less clear than Integra Live's control interfaces. Perhaps it would be more accurate to speak in terms of HMVP design pattern; classification border lines, however, often shift and do not always pertain to all aspects of a software program since, at times, choices are only locally relevant for some of the contents of the software itself.

One must also take into consideration the fact that, in the developers' mission presentation of menu possibilities, Usine Hollyhock offers more in terms of producer-user interaction, including more diversified solutions in the graphic representation of every work mode.

5. MMixte

5.1 Background

At the time I am writing this dissertation, the latest version of Max 7.3.5 is and has been, for the last thirty years, the data and audio streaming control software program most widely used in mixed music by more than 20,000 users in the whole world, a figure which could easily reach up to 100,000[55] if we were to take into account users of all versions of the software program, including those enabling Max use in Ableton.
Max applications have been extended far beyond the area of installations and video interaction through major mixed music achievements in qualitative terms. Indeed, their most knowledgeable users have developed original pedagogic techniques through various teaching positions. Universities, conservatories and fine arts academies all over the world have set up electronic music classes devoted to learning Max with specific, multi-level modules. More than 1500 institutions own official versions of the software[56] program. Personally, I was trained on Max through the Ircam computer music *Cursus* 1 and 2 in 2009 and 2010. Along with software program fundamentals, I learned ways of teaching them following a tradition of use developed at the Ircam by designers of the program's early versions, prior to its transition to Cycling74. This is the idea I subscribe to: the way audio signal is processed in a software program, at a broad level of software architecture, affects the signal quality itself at the end of the digital chain before the conversion to analog for audio diffusion in physical space. Max programmers have no control over fundamentals of digital audio streaming since the software program only enables module use for elaboration through visual programming; notwithstanding this limitation, conceptually linear programming in the composition technique of the “patcher” file and the chain of passages making up audio elaboration into objects does much to render, in physical space, only the audio signals with the processing wished for without any other artificial colorings. This is the reason why, in my short teaching experience, I have become such a keen supporter of audio Max programming techniques and I have constantly sought to convey the idea that proper programming helps achieve better audio in the analog world.

---

[55] Source: Cycling ’74 board

[56] Source: Cycling ’74 board

Since the beginning of my Ircam *Cursus* years, most of my commissioned pieces have involved interaction with digital technologies. Over the years, I have found myself writing a number of pieces requiring a number of audio and computer set-ups. I had no choice but to take a few days' work just to find the right settings at the beginning of the Max patcher, and finally get on with programming audio processing itself.

In other instances, I have had to devote additional time to computer parts of previously performed pieces, re-adapting them to new production demands -- to new spaces, that is -- requiring different input and output channels from the original set-ups. Converting the number of I/O channels, though it might seem "manual" labor or, at best, work to be assigned to such conversion algorithms as WWS [57], ambisonics and other techniques of multiplication of listening points in space, actually turns out to be quite a delicate task in a Max patch since it involves not only digital re-adapting of required I/O channels but also, and especially, control of all communication functions associated to these channels: data and audio send, data control modules, matching all functions linked to other parts of the patcher graphically connected through alternative syntax (coll, preset, pattrstorage, poly~, gen~, objects themselves, etc.) instead of cables. Indeed, work involved in conversion and control of such complex set-ups can be considerable. I have had many an occasion to see exquisitely complex patches in audio programming just as I have seen patches in total disorder from the standpoint of graphics, utterly unmanageable by people other than the composer himself -- a situation at times incomprehensible even to the authors themselves years later. When graphic order only complies with the order of data send in software threading but disregards the Gestalt of figure perception of the human brain, the use of these patchers simply ends up being deceitful.

Since the 2009 version 5 release of the software program, Max has integrated a "presentation" mode. From a structural point of view, it enables MVP or MVVM implementation of design patterns; it also enables isolation of objects matching controls without visualizing all objects contributing to the way they work. The presentation mode was a crucial novelty improving the experience of Max usage in live situations, in *gaming* quality, and in possibilities of creating true user interfaces. Though not compulsory, proper order in object arrangement in the software program's *edit* mode is likewise necessary. The *edit* mode enables *a posteriori* data and audio module creation and modification; whenever graphic order fails to exemplify the data and audio streaming

[57] World Wide Stereo

chain, the patcher's re-adaptation to other performance situations is simply much too cumbersome.

Ultimately, all of these needs lead us back to one of the main reasons why software architectures were at the center of discussions all through the Nineties: the possibility of software component *re-use* in other contexts and software *adaptability* to new operative systems. The *MMixte* project started developing in 2014. It was born with bottom-to-top dynamics as a formalization of my own practice over time. At that point, I also decided to produce middleware for composers and computer producer communities, I spent a fair ammont of time thinking about what could be attributed to a habit of personal use and therefore not important in a general context and what was a common need for all users. Formalizing components in software architecture was fundamental to keep practices of use away from structural archetypes. This project, in its realization in a specific software program, already represents, at any rate, a binding choice featuring only structural elements not subject to an “opinion” regarding their function, usefulness or programming. Such neutrality was needed to avoid conveying the arrogance of believing that an entire community might deem personal practices absolutely necessary. This, I believe, is a highly significant aspect of this project: *MMixte* is neither a technique nor a user's practice to be taught to future users: since its contents are already integrated in programmers' know-how, it is simply passed on so as to enable the community to do the same things as before significantly faster. Its independence from the software's future releases - to the extent it is indeed desirable - is another crucial aspect of this middleware.

MMixte is produced using only the Max base library; it has no *external* objects programmed in C+ by a programmer outside the company and encapsulated in the SDK prototype which Cycling released for developers. This particular aspect was thoroughly elaborated following a few ideas:

1. the Max base library is not likely to change, since deleting objects from earlier versions would imply the impossibility of using files saved in earlier versions; it is therefore more likely that the software's standard object collections will be expanding;
2. the programming quality of *external* objects makes first the object itself, and the collection containing it later, subject to opinions regarding programming quality and user functionalities. This specific aspect was carefully avoided all through the project's production;

3. the architecture behind the programming interface for Max users might change with the upcoming versions. While developers at Cycling74' are responsible for code conversion behind the Max base library objects, conversion of external objects of third party collections is likely to fall under the responsibility of authors of such collections; consequently, upgrades will be necessary for each and every novelty introduced by the mother company.

The company owning the software program has announced the release of Max's version 8 in September 2018. Setting aside legitimate doubts, It is hard to conceive that problems in compatibility might arise given the basic choice of not implementing objects that might become obsolescent with the software's future versions.
MMixte only includes elements of software architecture without any audio processing. This is a novelty in the landscape of projects developed in Max by private users and made available to the community. Most collections have more or less contributed, depending on the case, to the modification of audio streaming in its digital features, *ergo* in the analog ones when, once out of the computer, the signal is sent to hardware machines of the electro-acoustic chain. On the contrary, *MMixte* does not affect audio quality any more than the software program itself in the way sound signal is codified, this aspect remaining under the responsibility of the user. Neutrality with respect to sound signal quality is yet another strength in the *MMixte* collection whose aim it is to give users confidence as far as the collection's non-influence upon the users' programming skills in *ad hoc* audio processing.

5.2 MMixte structure

MMixte is a Max package. "*Packages* are a convenient means of bundling objects, media, patchers, and resources for distribution. A package is simply a folder adhering to a prescribed structure and placed in the 'packages' folder. Folders adhering to this structure can be accessed by Max to integrate seamlessly at launch time" [58]. The *MMixte* folder contains the following sub-folders:

- **doc**: package documentation with a text format list of the module-by-module *send, send~* and *receive* attributes, a sub-folder with ten tutorials, presentation patch, package icon and a text file with credits and acknowledgements;

[58] https://docs.cycling74.com/max7/vignettes/packages

- **example**: a *demo* patcher with a practical use of *MMixte* in a patcher concert prototype situation. For this function, text files including the computer score needed for the Max patcher to work and two audio files to be diffused in the simulation are also included in the sub-folder;
- **extras**: the mmx.overview patcher enables access to modules from one single screen display;
- **help**: includes help files for every module (the *.maxhelp* extension helps recognize them);
- **patchers**: includes the collection's patchers. There are two files for each patcher: one including the file inserted in a *bpatcher* [59] object with module controls while the other one includes the entire module (the standard structure of patchers in Max);
- **read-me**: a text file with installation and user package instructions;
- **snippets**: snippets are patchers with a dedicated extension (*.maxsnip*) flowing into a list recalled through a dedicated default button located in the right column of every patcher's frame. “A *snippet* is a patcher file that contains something used often and that does not have to constantly be re-created as the user’s work”[60]. The sub-folder includes seventeen snippets matching the collection's 17 modules;
- **templates**: the *mmx.beginning.maxpat* patcher is a small, empty table with the collection's background default color. Patchers located in the *templates* sub-files in the *package* sub-file of the *Max7* file can be directly recalled from a pop-up menu (e.g. *Extras*) when the software program is active.

5.3 The modules

Seventeen modules in the shape of snippets making up the collection feature a different set color compared to the software's standard presets [61]: the darkest one is used for the patcher's background; the second one as background for the modules. The third shade is assigned to the buttons of the modules' graphic interfaces as well as other graphic elements. The fourth color is used for objects' contour; the lightest one for texts and signal visualization. Color choice meets graphic, as opposed to, functional criteria;

[59] A *bpatcher* is a Max object enabling visualization of a patcher in another file. It is generally used for another patcher's graphic interface in a larger context. Such an interface, its own structure included, cannot be altered without having to openthe original file; it is therefore meant to be used such "as it is", in the form (algorithm and graphic interface), that is, as it was saved and designed for use.

[60] https://docs.cycling74.com/max7/vignettes/snippets

[61] Source: *noche color* da https://color.adobe.com

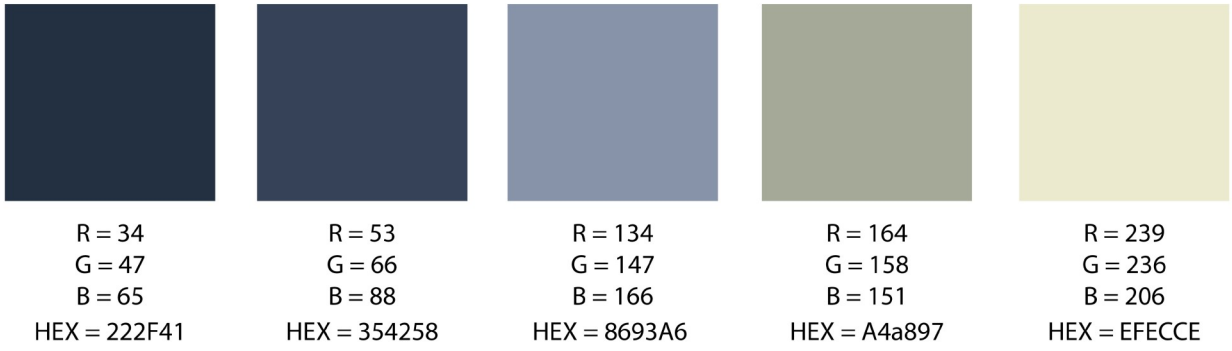


Fig. 5.1 Color patterns used in *MMixte*

when more modules are in the same patcher, the main window's dark background and its neighboring tones enable quick visualization of textual indications and of audio signal input and output: in other words, the most relevant elements in the context of live performance. The package enables personalization of all modules, though their variation may not be strictly necessary. Once a module is recalled from the *snippets* list, it may be changed in the use patcher without changing the matrix module.

All modules feature an enclosed section in a *bpatcher* and another one in a *subpatcher* [62]. *Receive* (for control-rate communication) and *receive~* (for audio signals) objects are located at the outermost level in every module. In Max syntax, a *bpatcher* is a graphic object enabling the picturing of a patcher in *edit* or *presentation* mode (depending on saved file preferences) in another window. While such a window does not enable changes in internal elements, it does enable their use. All Max objects used for module construction, in need of no visualization or changes, flow into bpatchers. *Receive* and *receive~* objects ensure communication with objects in the file loaded into the *bpatcher*.

In Max syntax, a *subpatcher* is an object containing another patcher which users can modify since it can also be edited. *Subpatchers* contain all the objects of module sections subject to user changes and must be adapted depending on specific cases.

As already stated in the previous paragraph, *MMixte* only deals with architecture, not audio processing; users are therefore responsible for all data and audio processing functions. A threefold motivation justifies the choice of splitting the software program's different possibilities for multi-layered module creation:

---

[62] A *subpatcher* is an object including a series of objects enclosed at a lower level. This kind of hierarchy enables direct altering of the patcher structure the programmer is busy operating in. More *subpatchers* may appear within the same patcher and be altered independently of each other. The choice of enclosing a section of the *patcher* at a lower level is often made following merely graphic criteria of order, suitable for the simplification of the overall algorithm structure obtained through programming for objects. From a functional point of view, nothing changes if the objects recalled in the patcher are located at the first level or at a lower one.

- **economy of objects**: every module in *edit* mode is made up of the lowest possible number of objects of the highest possible intelligibility in the least amount of space. In *presentation* mode, every module features the *bpatcher*, a single object always in default *presentation* mode.
- **saving graphic skin**: though users may change everything in MMixte, graphic skin is saved in order to provide the collection with an identity as well as ensure graphic homogeneity between modules.
- **additional programming freedom**: users can, and must, add their own patching in the collection's modules. In its own subtle way, the collection invites users to keep conceptual order in relation to the typology of functions to be implemented. This helps keep order in the general patcher. Below is the list and description of the modules making up the package.

5.3.1 mmx.audio_input

The input module enables audio signal input in the middleware. Signal input has no processing connected to the software; signal quality therefore depends on the kind of transductor eventually to be used on pre-processing performed *out of the box*. At the module's output, signal amplitude will be the only aspect of the signal to have been handled. Thus, the module works like a *pre-fader*. When Max is set in *edit* mode, the snippet contained inside the patcher looks like this:

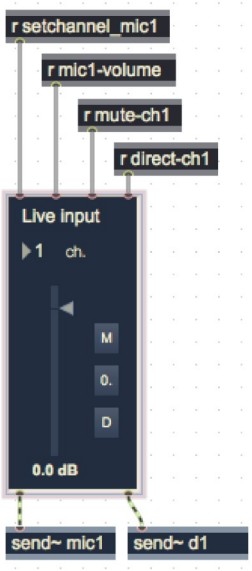


Fig. 5.2 mmx.audo_input module in edit mode

The *bpatcher* module features controls which users don't need to change:

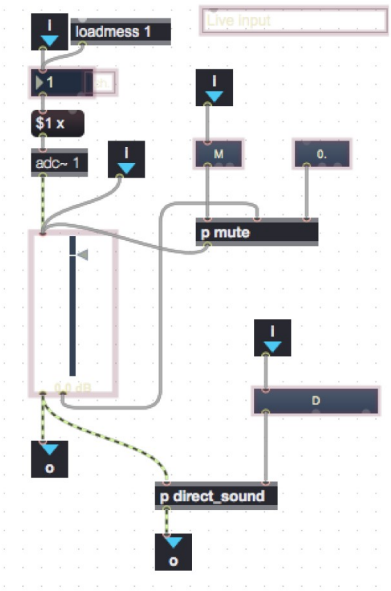


Fig. 5.3 Bpatcher including mmx.audio_input module controls

Three buttons are featured:

- *M*, setting the module[63] in mute mode;
- *0.*, bringing the signal back to 0. dB output level;
- *D*, activating direct signal send[64];

The channel number can be altered either through direct action on the control interface via scripting or through *send and receive* objects. The same applies to the signal's output volume from the module; the interface enables manual changes; these, however, can be remotely activated communicating with the object. As previously mentioned, send~ and receive~ objects are located outside the bpatcher: they can therefore be modified. The syntax of these objects' attributes enables default communication between various modules; users can change attributes for their own project according to their own preferences. Whenever more audio inputs are involved in a single project, users must change the names of the attributes by altering the progressive figure which is part of the default attribute. This process could have been automated; part of the gaming which is a significant aspect for Max use (that is to say, the craft needed for production of proper

[63] The module's gain is designed to work in *pre-* mode to keep monitoring input signal level even when the audio is not pursuing its course in the patcher up to the output.

[64] Direct signal send is not always needed in a mixed music piece.

audio patchers) lies in the very possibility of hands-on change of the way modules work. As it turns out, this hands-on operation is absolutely necessary as it reduces module complexity, making it more easily intelligible and modifiable to average skill users.
In the module's architecture, the signal is doubled along two independent audio threads called *mic1* and *d1*. The first one flows into the processing module and it is modified before entering the module for the signal's output. The second one is projected towards direct signal send to the module output for direct signal diffusion. Here is the design pattern supporting the module:

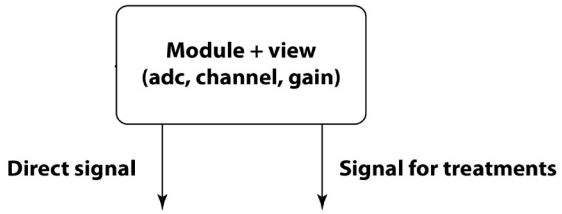


Fig. 5.4 mmx.audio_input design pattern

5.3.2 mmx.audio_output

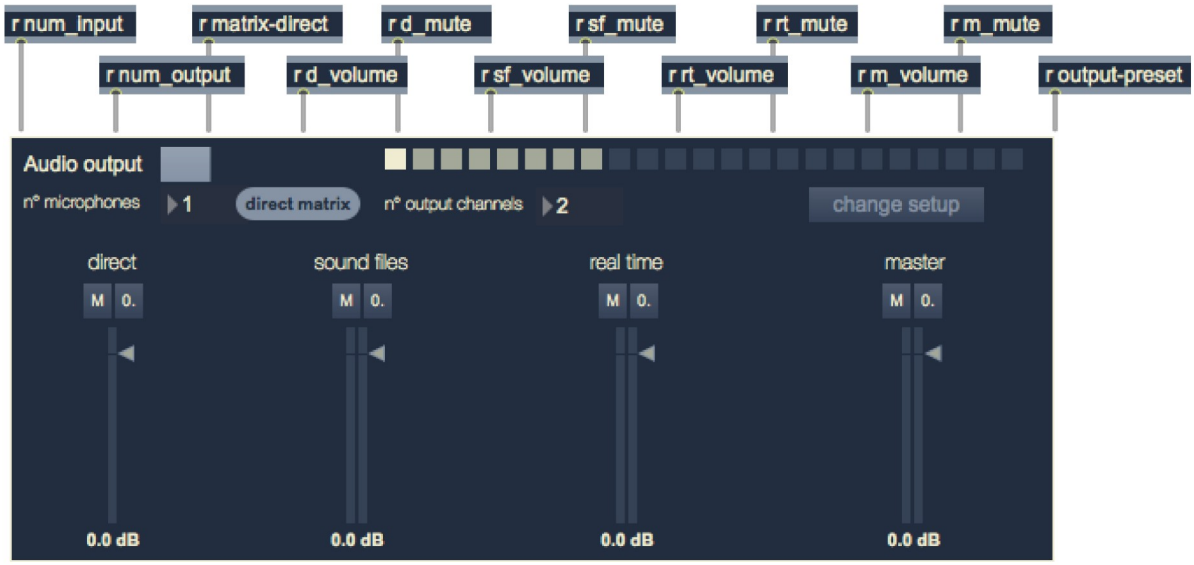


Fig. 5.5 mmx.audio_output module

All audio signals entered in the patcher to be sent to the software program section dealing with digital-to-analog conversion (DAC) travel through this module:

- direct signals as they come out of the *mmx.audio_input* module;

- the signals coming from audio files;
- analog signals processed through signal transformation algorithms and digital audio from wave generators.

Audio categories all have their own dedicated faders. All of these flow into a master fader directly connected to the software program's DAC:

The use of dedicated faders for every signal typology enables total signal amplitude modulation in the mix through master gain. In a concert situation, it helps to be able to set every audio signal typology in balance: depending on the room, on speaker positioning and typology, a different balance may be required. At best, this system compensates for the need to set sound amplitude through a mixer, since the total signal sum can be set directly *in the box*. The number of channels is programmable via interface or scripting.

The module automatically adapts all necessary connections so the number of programmed channels may be directly connected to the DAC. Through *thispatcher* object syntax, an algorithm inserted in the module's *bpatcher* enables channel number multiplication for every partial output typology (direct, audio files, real time) as well as for the *master gain*:

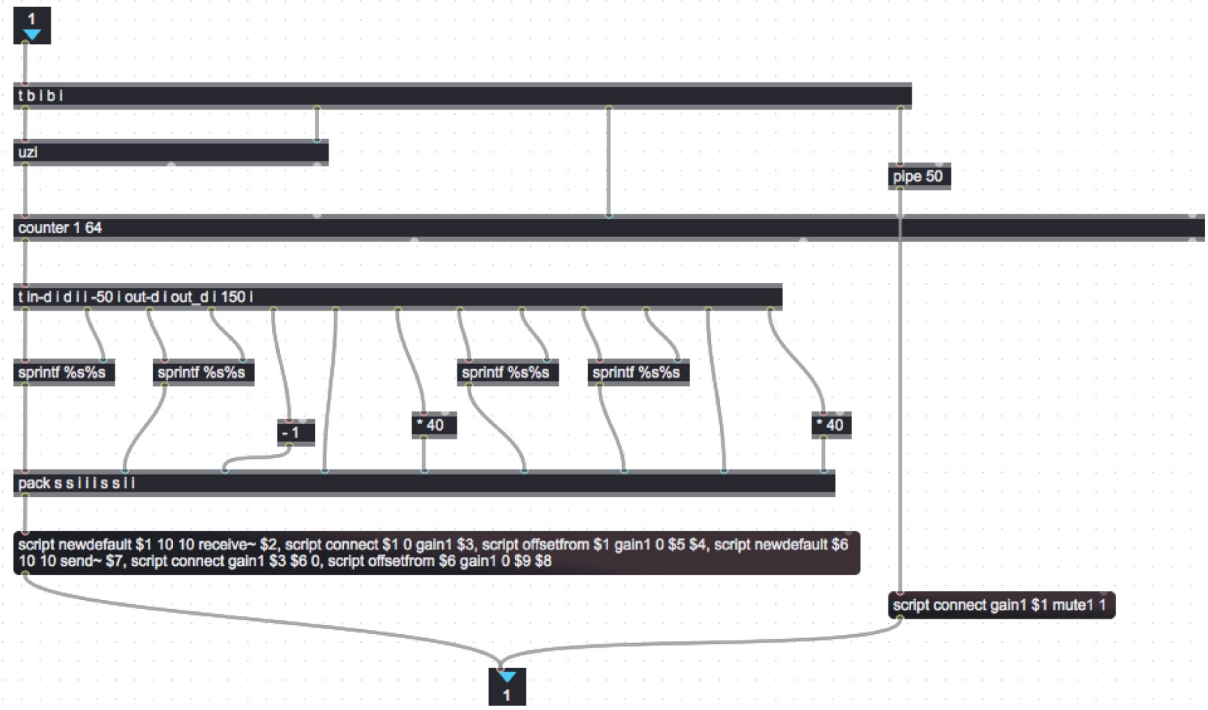


Fig. 5.6 Partial view of the algorithm generating the number of channels creating channels connected to the DAC

All of this *sub-patcher*'s objects are connected to the module's *thispatcher* object. Algorithms for automatic generation of connections can also delete them whenever the number of channels needs to be reduced as desired:

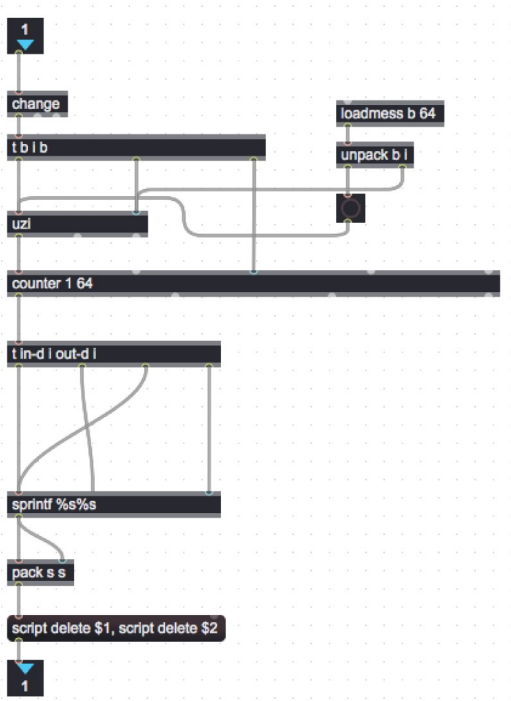


Fig. 5.7 Partial view of the algorithm generating the number of channels deleting channels connected to the DAC

The same system of automatic channel generation deals with the number of matrix inputs and outputs for direct signal routing. Channels eventually sending direct source amplification may vary according to the project; hence, the extreme difficulty in establishing *a priori* rules. Once the number of diffusion channels is set, users are free to choose the channels to which the direct signal will be sent via the dedicated graphic interface displayed like a pop-up window once the *direct matrix* button is pressed on the module's controls:

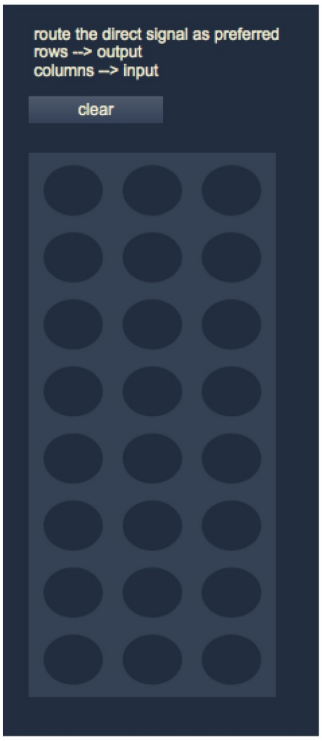


Fig. 5.8 Presentation mode view of the matrix for direct signal routing in three input signal and eight output channel set-up

A series of classic set-up combinations (1 input and 2, 4, 6, 8 channel output, 2 inputs and 2, 4, 6, 8 output channels are featured as presets. Users can save other presets and changes can be blocked to avoid accidents in the course of performance. All parameters requiring user direct control can be remotely altered through the *send* and receive objects outside the module's bpatcher. The design pattern underlying the module is the following:

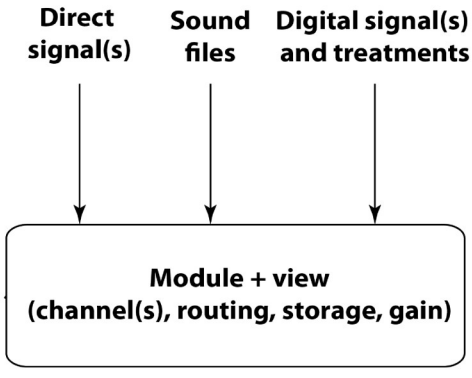


Fig. 5.9 Module design pattern

5.3.3 mmx.event_counter

An event counter is meant for visualization of cue list figures. In the structure of electronics in real time divided by events, cue list figures are index numbers of a series of events[65] with a series of controls for changes in software status.

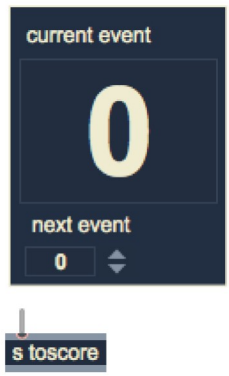


Fig. 5.10 mmx.event_counter module

The module enables visualization of the current, as well as the following, event number. The module's interface being of the MVVM kind, it works to visualize and change module status. Changes in cue list numbers can occur either through another module in the collection (*mmx.score_select* or *mmx.score.qlist*) or through the graphic interface. The “next-event” number prepares the software program for a cue list figure send which is not necessarily ordinal compared to the previous one without immediately changing software status. Instructions in the pre-established event are only sent once the control to step up to the new scene is forwarded.

5.3.4 mmx.generator

Designed to hold digital audio generators, this is a user program module. Such generators may be simple waves handled according to use or physical models:

[65] Depending on authors, programmers and composers, such events are also called “scenes”, with reference to the same set of instructions subsumed to an index number.

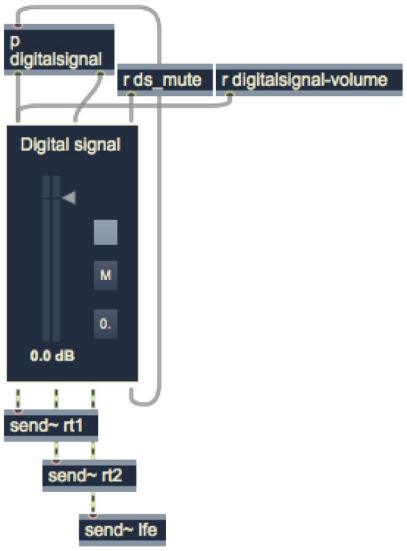


Fig. 5.11 mmx.generator module

The module is pre-arranged to send stereo signals. As a result of personal choice, attributes of the two *send~* objects matching the first two outputs of the signal flow into the module's output section for processed signals diffusion. This choice was made for reasons of mere order; thus, only signals coming from outside the computer will be flowing into the *mmx.audio_output* module section for visualization, routing and handling direct signal amplitude. A third output, called *lfe*, is programmed to send, independently of the overall output, the signal to the subwoofer; otherwise, it may be used for direct signal send as in the *mmx.audio_input* module. In this case, this channel is a signal sum set to a default -3. dB decrease in the amplitude of the resulting signal. If there's a need to send separate stereo signals across two independent channels, users can multiply *send~* objects matching the first two outputs changing *ad hoc* attributes' names. Programming a third audio thread coming from the module leads to personal uses of the module a priori not envisioned.

In and of itself, the module is empty: it can only work when completed with the programming of the user's wave generators. To enable this, the programming section is located outside the *bpatcher* with the modules controls; the module's *ex ante* diagram may, in this case, be represented as follows:

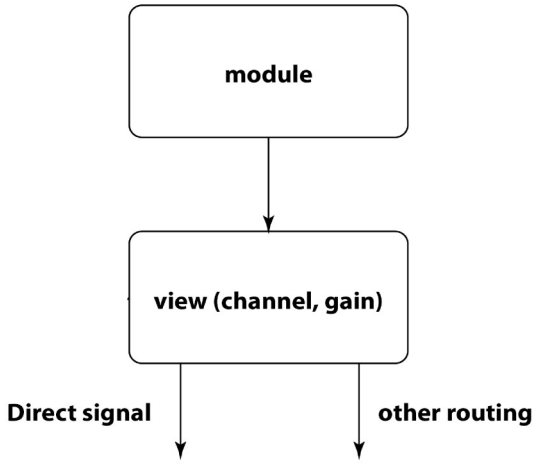


Fig. 5.12 mmx.generator module design pattern

5.3.5 mmx.initialize

The initialization module includes a series of instructions to be sent to the computer's CPU after the patcher's opening prior to middleware use. Patcher initialization operations involve:

- setting DSP parameters (I/O set-ups, sampling rate, vector size, signal vector size, etc.);
- loading audio files to be launched when using middleware;
- preparing programmed patcher modules to the status they must begin working with before instructions for the first event are given.

I personally appreciate, once the DSP is activated, that all external audio sources (therefore connected to *adc~* objects) are set to “mute” mode to avoid audio feedback that might be caused in the course of performance. The “mute” control does not in itself make up an event; it is, however, the situation preceding event n° 1; among the n°1 event instructions, some will be fulfilling the purpose of bringing input signal amplitude back from a -∞ level up to the desired one. Max software includes a number of objects (*loadbang* and *loadmess* for automatic sending strings (text commands) and impulses to the opening of the patcher). When the number of strings to be sent is considerable, a conflictual situation might arise, leading to software crash. Therefore, I prefer replacing the software's default objects with a *send* object I have chosen to name “asloadmess”, so that when the software opens and requests the CPU to perform a series of operations it

does not match recall time of the middleware's local settings. Middleware initialization ensures correct performance for all such operations.

In such a module, users can also perform other operations needed for middleware setting: loading data files for plugins (.fxp), loading scripts for physical models, loading text files for objects requiring external text files (*coll*, *pattrstorage*, ecc.) or selecting external devices to be used with middleware. Below is the module in its *presentation* mode:

Fig. 5.13 mmx.initialize module

The small number in the square shows the number key of the keyboard enabling subpatcher opening without having to use the graphic interface. A dedicated paragraph includes an explanatory guide to the shortcut series.

5.3.6 mmx.listener

This module's purpose is to alter input signal's amplitude according to a given function. Altering signal amplitude coefficients is a technique known as *envelope follower*, typically used to assign amplitude coefficients to a signal processed in ways other than the original according to a rule expressed by a pre-established function.

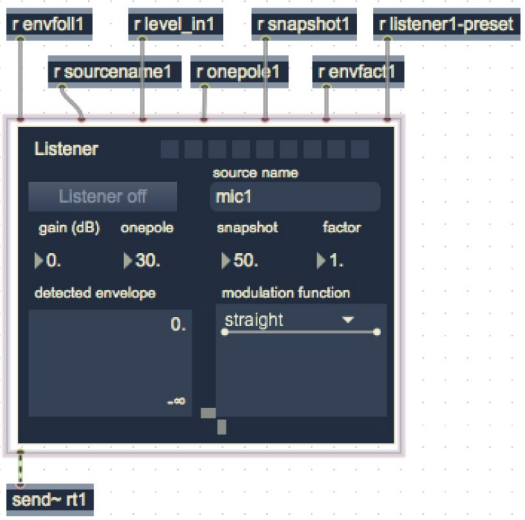


Fig. 5.14 mmx.listener module

Once the listener module is activated (listener on), the signal from the *mmx.audio_input* module is entered into the module; amplitude is then processed as follows:

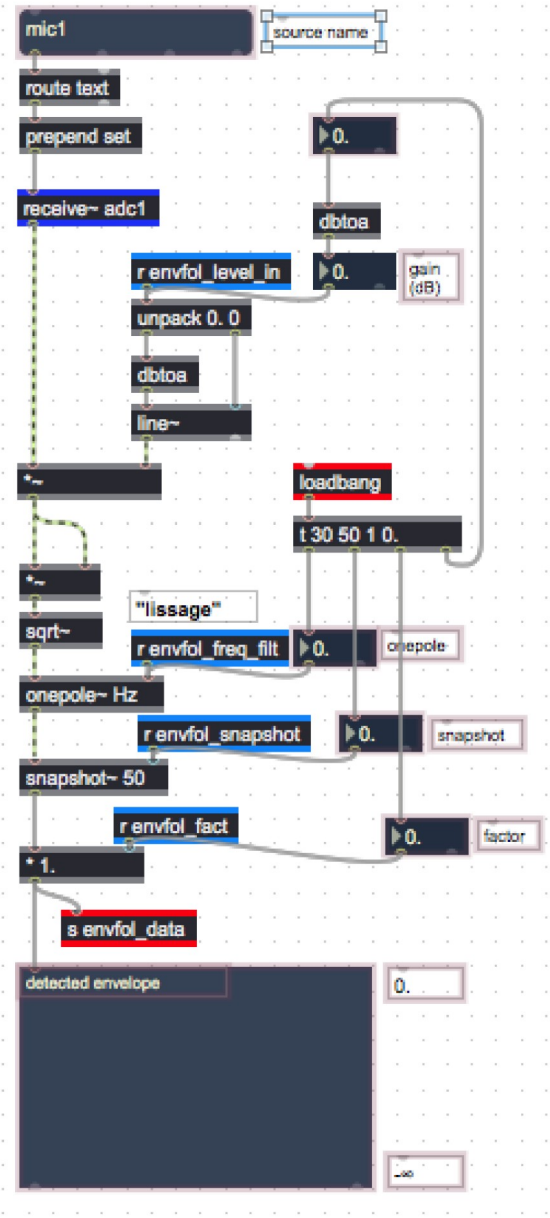


Fig. 5.15 Trasformation of the input signal amplitude coefficient

Following preliminary changes in amplitude (envfol_level_in), the square root of the signal's square[66] goes through a *onepole~* filter. The onepole~ filter implements the simplest IIR filter, providing a 6 dB/octave attenuation. This highly efficient filter, shown in the following diagram is especially useful for such operations as gently rolling harsh high frequencies as well as smoothing out control signals as shown in the following work diagram:

[66] This is done to get rid of the negative part of the signal.

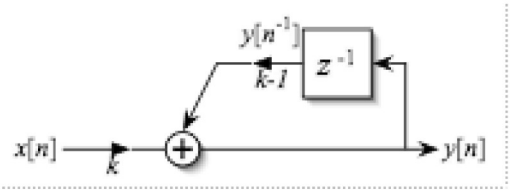


Fig. 5.16 Onepole~ filter diagram

Amplitude coefficient information obtained through this procedure is sent every *n* ms (*snapshot~)* and it may be further multiplied with another linear coefficient to magnify or attenuate the signal. The curve thus obtained is sent to a second function altering the signal according to three possibilities available in the module presets:

- no added change (*straight)* to input coefficient;
- directly proportional (*increasing)* to input coefficient;
- inversely proportional (*decreasing)* to input coefficient.

Personal non-linear functions may be saved in module presets. Coefficients thus obtained (*sfplay_level_out)* will be used for input signal amplitude to be sent to the processing module (through *send~ r1*).

All of these processing steps have *receive* objects located outside the *bpatcher* enabling data changes via scripting and graphic interface.

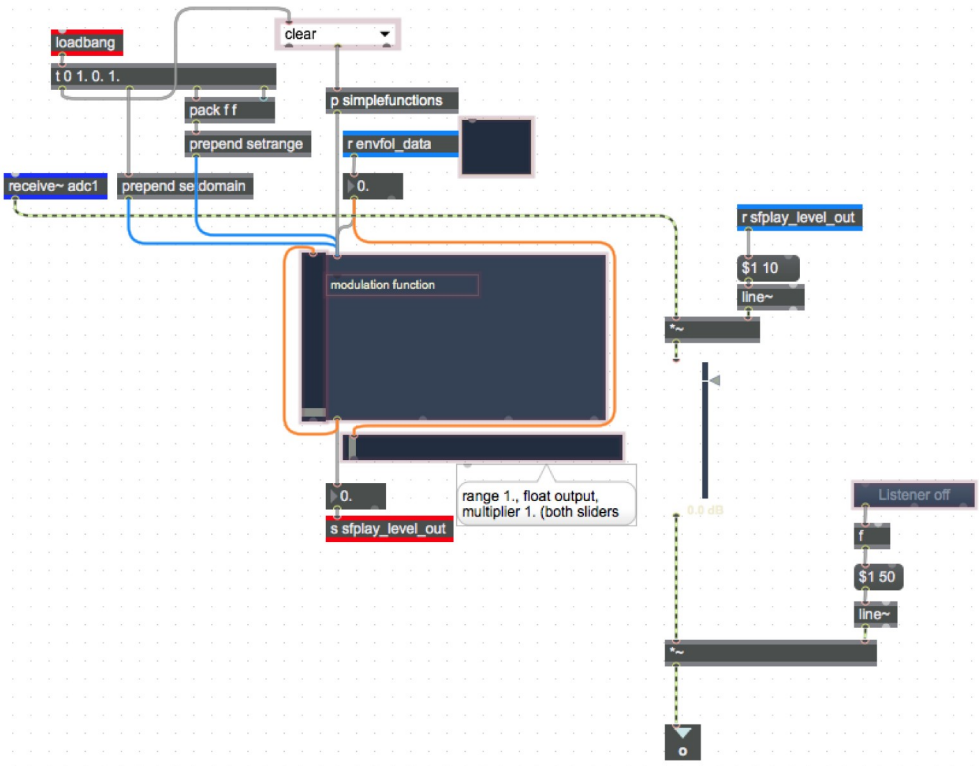


Fig. 5.17 Coefficient transformation according to a pre-ordered function

5.3.7 mmx.lock

This module exploits *thispatcher* object possibilities to prevent closing of the patcher containing it. This is a module that provides a function that does not concern the conduction of the audio signal or of the numeric data that contribute to the electronics in the strict sense; it is an accessory that completes the collection, providing a system that prevents accidental closure of the patcher during a concert performance, with clearly catastrophic results:

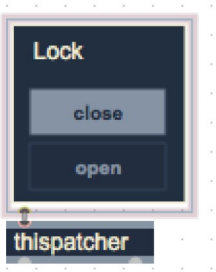


Fig. 5.18 mmx.lock module

Once recalled, the module is automatically set in *close* mode; the module's switch, however, requires manual unlocking for the patcher to be opened.

5.3.8 mmx.midi-interface

This module provides a graphic interface for midi controller use:

Fig. 5.19 mmx.midi-interface module

The figure in the square to the right inside the *bpatcher* indicates the numerical key in the computer keyboard recalling the controller's graphic interface without using the module knob. The Behringer BCF 2000 and Mini-Korg controllers have partly inspired the midi controller's graphic interface which includes:

- 9 knobs;
- 18 buttons to be set either in *button* mode or *toggle* mode;
- 9 faders;

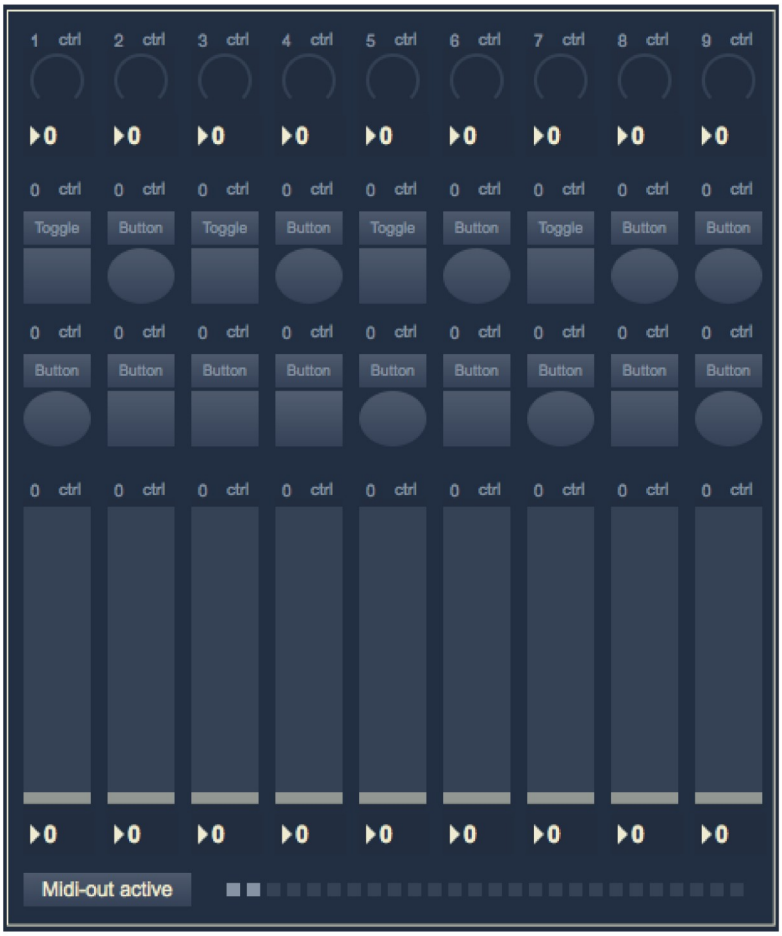


Fig. 5.20 Midi controller graphic interface

This module also enables midi-out message send, thus updating fader and knob status of a midi set up capable of elaborating input data. Assigning control numbers of the interface is required prior to module use and saved in one of the presets.

5.3.9 mmx.midi-keyboard

This module enables the use of a midi keyboard inside the middleware. A midi keyboard may serve a number of purposes: launching pitch/velocity couples matching every key for pre-loaded sample launch in a sampler, keeping in mind that every key can trigger a series of computer score instructions. In other words, midi data launched through pressure applied to midi keyboard keys can either perform a musical function of a digital musical instrument or work as a midi controller. Every project has its own peculiarities and needs: it is therefore impossible to formulate general rules applying to midi device use. The module is pre-arranged for use in both ways:

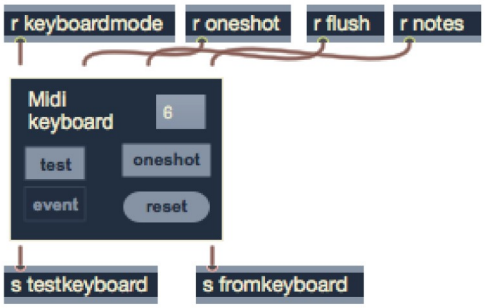


Fig. 5.21 mmx.midi-keyboard module

Access to these functions is granted via scripting or a graphic interface:

- **keyboard mode**: *test* mode checks communication between devices and computers, while *event* concerns the use of middleware midi data (for instance, event launch or playing a sampler). Practically speaking, midi information from external devices can be geared towards two different threads according to the demands of users project usefulness;
- **one shot**: this mode filters information which would have otherwise been launched on key release (that is note-off messages: velocity values of zero with the pitch number matching the key) so that a key may send a single pair of pitch/velocity data in the action pair given by key pressure and ignoring its release;
- **flush**: erases active pairs of pitch/velocity data. In the graphic interface, I have chosen the term *reset.*
- a data pair matching pitch and velocity for every note is expected by the *receive* object with *notes* attribute. Midi data input from external devices occurs through the *midiparse* object which needs no upstream specifications about the midi channel set-up sending information. The module therefore works independently of the channel set in the external midi set-up.

Keyboard visualization may be useful both for communication efficiency check between the external set-up and computer and to replace external set-ups whenever the computer operator personally wishes to send pitch/velocity pairs through the computer's interface. From the top right-hand button of the module's *bpatcher* or through the number matching the number of the numerical key to be pressed to open the pop-up interface, a complete midi keyboard can easily be visualized:

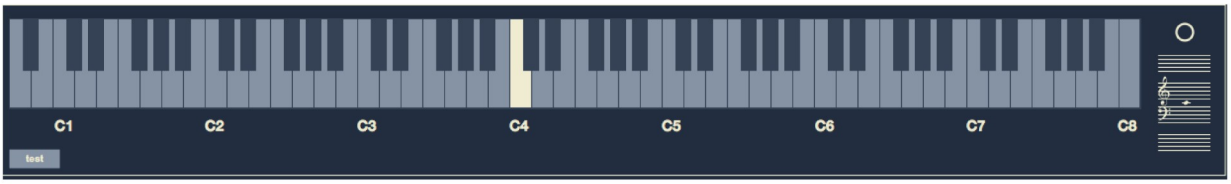


Fig. 5.22 Midi keyboard graphic interface in mmx.midi-keyboard

The right-hand section blinks whenever the computer receives information; staves display the note sent to the computer as well as those also sent from it. Every module can manage a single set-up; more modules can co-exist within the same project. The diagram supporting this module is the following:

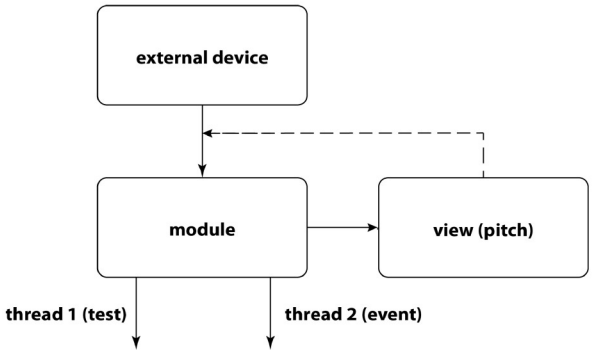


Fig. 5.23 Module design pattern

5.3.10 mmx.midi-pedal

This module is designed to use the 127 value coming from controller 64 of the GM2 protocol -- the so-called *sustain* -- to launch events. Midi pedal use for forward movement of the electronics scenes divided by events is perhaps the classic method used since the Nineties to be assigned to musicians on stage.

Since this module is mainly dedicated to an external device, not all functions are matched in the graphic interface. Sending a *bang* by a midi pedal can be substituted, for instance, by pressing the space bar on the computer keyboard; in that case there will be no need to introduce the module within the project. The module enables regulation of the following functions via scripting:

- **pedal device**: selects the name of the device receiving the information;
- **pedal direction:** most commercially available sustain pedals have one switch for data send direction change. In practice, the choice of sending number 127 when pressing or releasing the pedal is available. The module ensures compatibility with such an option;

- **pedal mode***:* as with the midi keyboard, this module works either in *test* mode or in *event* mode. Two separate data threads can be used to for the user's project. The two modes may also be selected through the graphic interface;
- **pedal gate delay**: once it has received information, the module does not accept input data for an *n* duration expressed in milliseconds. This is a safety measure meant to set limits to damage caused by accidentally repeated pressure of the pedal. This gate's timing can also be changed through the graphic interface. As a rule, such delay value preventing new data input up to the new gate opening should be slightly inferior to the minimal temporal distance set between two project events the middleware is being programmed for. This module's design pattern is similar to the one in fig. 5.20.

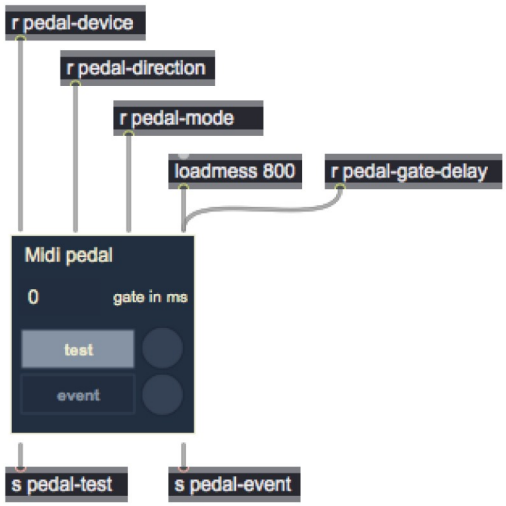


Fig. 5.24 mmx.midi-pedal module

5.3.11 mmx.midi-setup

This module is a control interface for midi instruments and controllers selection:

Fig. 5.25 mmx.midi-setup module

The choice of devices is to be made manually. The interface is pre-arranged to communicate with *mmx.midi-pedal, mmx.midi-controller,* and *mmx.midi-keyboard*

modules. If more devices were put to use for the project the middleware is being programmed for, then more *mmx.midi-set-up* modules would have to be selected; send attributes outside the object's *bpatcher* would have to be changed.

5.3.12 mmx.monitor

This module enables CPU status visualization and manual selection of the DSP's work settings:

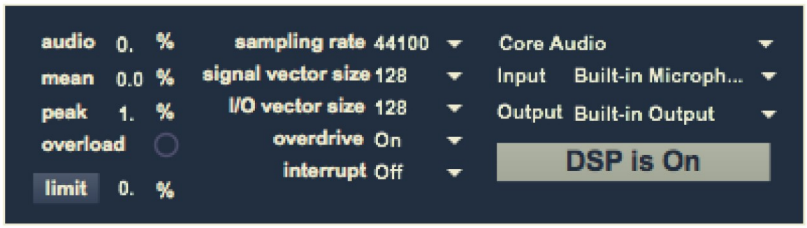


Fig. 5.26 mmx.monitor module

Available controls are borrowed from by the Max *Audio status* patch enabling the same parameters' settings through the software program's default external window.

5.3.13 mmx.player

Players read audio files saved in .wav, .aif and .aiff formats in all possible sampling rate and bit depth combinations. Default version players support both mono and stereo files. Though it may be possible, reading multi-channel files in Max is, in my own experience, somewhat risky: I have often found myself having to separate multi-channel files in a number of pairs of stereo files or in a series of mono files. If a multi-channel player is necessary, users may wish to add their own module with the logic of attributes for *send* and *receive* objects enabling communication between package modules.
The module is pre-arranged to read files loaded through a playlist, so they can be updated by recalling their own cue list number. When recalled, the number is displayed in the module's *bpatcher* and an audio file may be recalled from the same box thanks to the module's MVVM design pattern.

Fig. 5.27 mmx.player module

The button at the top left allows the numerical keypad to open the modifiable subpatcher of the module. It contains two players connected to the same cue list so that up to two files can be merged at the same time. If the project the middleware is being used for requires it, more players can be put in sequence for several audio files to be read at the same time. The module features three outputs: the first two are mono for both channels which the module is pre-arranged for; the third one is a sum-output to be eventually used for audio send to the subwoofer or any other function requiring sum-output.

5.3.14 mmx.score_qlist

The computer score is the central part of a system elaborating and diffusing electronics divided by events in a system such as the one for mixed music middleware architecture. It features a series of instructions mainly operating on the following fronts:

- modify DSP status;
- communicate with other computers through udp or tcp client protocols[67];
- modify the visualization of windows making up the middleware;

[67] While the software program integrates communication through udp protocols, as we gather by the base library objects (*udp.send* and *udp.receive),* communication via tcp client protocol is enabled through script and external objects programmed for Max by third party developers.

- work on specific object functionalities to create, delete[68] and modify objects in the patcher;
- work on objects' parameters to handle data, audio and video;
- launch audio and video files;
- change communication data with external devices through various supported protocols (midi, osc, implemented proper protocols).

The list of instructions is grouped, as already stated, in a number of events marked by an index number. Whenever the index number -- more commonly called "event number" -- is recalled, instructions are sent to the *receive* object connected with the same attributes; then the data immediately following the attributes will be altering object status for the functions listed above. The current state of the art provides a number of computer solutions. They vary according to the degree of complexity and automation marking the relationship between text and software program -- the score -- whose actions it must be synchronized with. The *score following* method came to prominence at the end of the Nineties (but became commonplace by the end of the years 2000) for software status change automation through pitch detection algorithms which, as they read a score transcription in the guise of a pitch list, are capable of launching events in performance without an operator's intervention at the computer.
As complexity of the synchronization method increases, so does the need to increase the number of rehearsals and test series: middleware must be submitted to them before the concert. This is not always possible in today's productions: thus, from a computer technology, less complex solutions requiring an operator to manually change middleware status in performance are usually preferred. In the last few years, more electronic musicians in the field of so-called art music have been increasingly concerned with the role of *performance* owing, perhaps, to the influence of electronic musicians involved in other musical *genres* requiring, as a set standard, the presence of an operator and his digital machines on stage. In such cases, manual solutions to be subsumed to launching event cue list are the safest solution from a computer technology and the one best suited to the kind of man-machine interaction shows require.

---

[68] For the sake of a comprehensive overview, I need to mention that, following a theoretical line of argument, it would also be possible to program a patcher where, through the *thispatcher* object, objects are created via scripting as needed; likewise, once they have performed their function, they may be deleted anytime through scripting. Technically, this is possible, even though, from a practical point of view, it obviously makes no sense beyond futile demonstrations of programming virtuosity.

MMixte features two types of computer score reading modules. The first one is based on the *qlist* object from the Max base library and includes syntax in the writing of instructions in the object's available helpfile. *Qlist* enables the launching of instruction sequences and programming linked to an event number whose linear latency is expressed in milliseconds:

```
----------------------- 1;
0 1 matrixrealtime-ramp ramp 100;
matrixrealtime clear;
segnale1 interp 500;
segnale1 -70.;
segnale2 interp 500.;
segnale2 -70.;
vol-direct interp 20;
vol-direct -10.;
vol-realtime interp 20;
vol-realtime 0.;
vol-files interp 20;
vol-files 0.;
vol-master interp 200.;
vol-master 0.;
distortion-onoff 0;
vol-distortion interp 20;
vol-distortion -70.;
----------------------- 2;
0 2 segnale1 interp 1500;
segnale1 0.;
volumegrancassa -50;
----------------------- 3;
0 3 volumeplayer 0.;
500 player 2;
```

Fig. 5.28 Example of a computer score written for a *qlist* object

In the above example, the final set of instructions reads "500ms" late compared to the time needed to send the first signal; both are linked to the number of event *3.* Variable " ----------------------- n " is a *dummy* with no content of its own: its sole purpose is to visually mark a separation between events in the list.

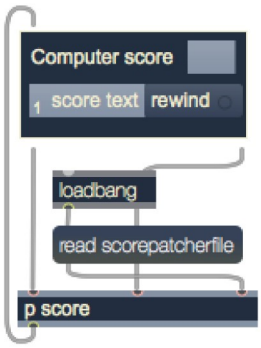


Fig. 5.29 mmx.score_qlist module

The *bpatcher* of the module illustrated above, only includes buttons enabling the opening of the computer score and the graphic interface enabling, in turn, ordinal forward movement, rewind from zero and non-ordinal leap between events. The computer score is

in a .txt text file saved outside Max's main patcher to be loaded in the *qlist* object in middleware initialization phase.
The number in the “score text” button shows the number of the numerical key used to directly recall the screen display with the computer score's text without having to use the graphic interface.

5.3.15 mmx.score_select

This module offers another reading method of a series of instructions grouped for the computer score function. It is best suited for beginners, since a higher degree of strictness reduces the need of creating scripts following compulsory syntax.
In this module, instructions are to be found in message boxes with *send* syntax[69]; they are connected to a *select* object including an “if - then” expression. When the number of the event recalled by the events counter matches the conditional expression's number, the instructions in the *message box* connected to it are sent.
The number in the right-hand button displays the number of the numerical key used to directly recall the screen display with the computer score text without having to use the graphic interface.

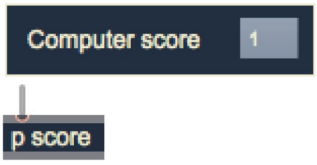


Fig. 5.30 mmx.score_select module

5.3.16 mmx.shortcuts

All numbers in the previous modules blue boxes display the number of the numerical key of the computer keyboard used to recall a module sub-interface [70]. These shortcuts work exclusively if the *mmx.shortcuts* module is in the main patcher whose “if then” instructions enable connections between window opening and key pressure through the ACSII code. Users may program their own shortcuts:

[69] The Max *message box* object performs a send function when a series of elements begins with the “ ; “ symbol.

[70] Such a pop up window may be hidden through the button located on the window frame itself or through the software program's internal shortcut (cmd+W).

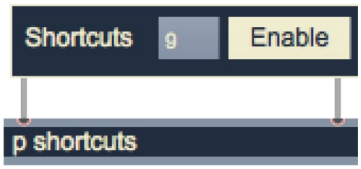


Fig. 5.31 The mmx.shortcuts module

The module's *bpatcher* features the shortcut button enabling visualization of the window with programming of all shortcuts in addition to another button enabling shortcut use or de-activation.
In *enable* mode, shortcuts are active; in *disable* mode, shortcuts are de-activated though still present in the middleware. This helps protect users from accidental module opening while using middleware in concert situations.

5.3.17 mmx.treatment

This empty module is designed to host user data and audio processing algorithms. Audio from the module, according to the logic underlying the entire middleware architecture, must flow into the *mmx.audio.output* module section dealing with signal amplitude coming from real time processing.

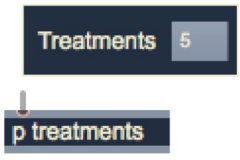


Fig. 5.32 The mmx.treatment module

The right-hand button figure points to the shortcut which may be used to recall the screen display with the computer score's text without having to use the graphic interface.

5.4 Send, send~, receive and receive~ attributes package list

The diagram below lists object attributes enabling communication with various modules through computer score programming. All module parameters are thus programmable either through graphic interfaces or pre-programmed event launch. When modules are doubled within the same project (a situation which is very likely to occur,

especially when involving modules for audio input, audio listeners, file players, digital generators and midi machines), the ordinal figure belonging to each of the attributes must be changed with the next ordinal figure. This syntactic rule follows logical meaning though other logics adapting to each project from time to time may be substituted to it:

	receive	send	send~
mmx.audio_input	setchannel_mic1 mic1_volume mute_ch1 direct_ch1		mic1 d1
mmx.audio_output	num_input num_output d_volume d_mute sf_volume sf_mute rt_volume rt_mute m_volume m_mute output-preset		
mmx.event_counter		to_score	
mmx.generator	ds_mute digitalsignal_volume		rt1 rt2 lfe
mmx.listener	Envfoll1 sourcename level_in1 onepole1 Snapshot1 envfact1 listener1-preset		rt1
mmx.lock			
mmx.midi-controller			
mmx.midi_keyboard	keyboardmode oneshot flush notes	testkeyboard fromkeyboard	
mmx.midi_pedal	pedal-device pedal-direction pedal-mode pedal-gate-delay	pedal-test new-event	
mmx.midi_setup		pedal-device	
mmx.player			sf1 sf2 lfe

Fig. 5.33 *MMixte* list of programmable parameters

5.5 Examples of design patterns

The use of package modules may be partial or total thus responding, with varying degrees of complexity, to every project's needs. Generally, the simplest diagram includes audio input and audio output modules alone:

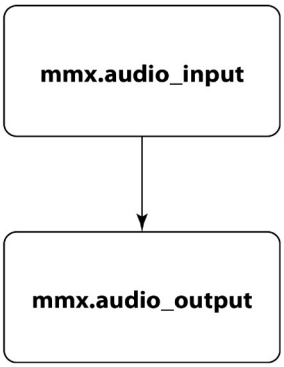


Fig. 5.34 Design pattern for *MMixte* n° 1

In the box processing of equalization, changes in dynamics, spectral processing and every other kind of processing -- not to mention reverb -- must be placed before the signal output:

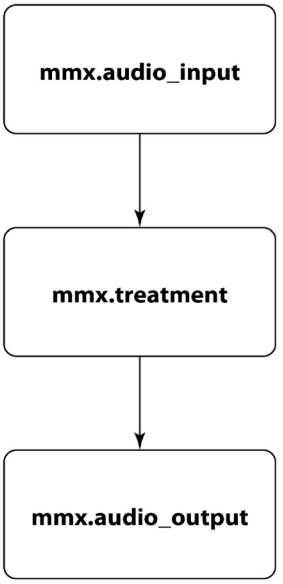


Fig. 5.35 Design pattern for *MMixte* n° 2

A similar diagram can also apply to audio files:

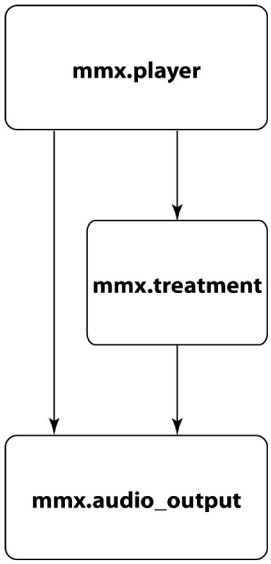


Fig. 5.36 Design pattern for *MMixte* n°3

An audio file may be diffused as it was designed or might need *ad hoc* processing prior to diffusion; hence the possibility of going through the *mmx.treatment* module as shown by the double arrow. If the live source is not captured by a microphone as in the elaboration of such virtual instruments as keyboards or midi controllers, the structure is then transformed into the following architecture:

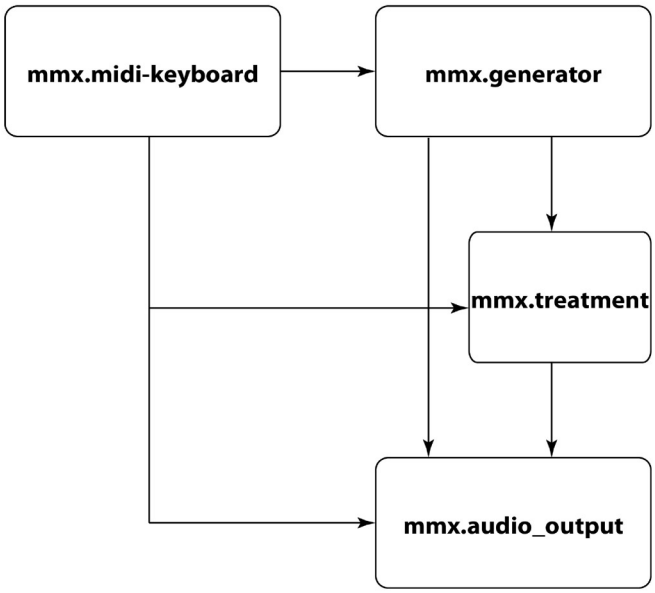


Fig. 5.37 Design pattern for *MMixte* n°4

Specifically designed for such purposes, the envelope follower requires insertion between audio input and audio processing modules. In Fig 5.38, *mmx.audio_input*, *mmx.generator*

and *mmx.player* modules are grouped in a single class since, in spite of their peculiarities, all perform the generic function of audio input:

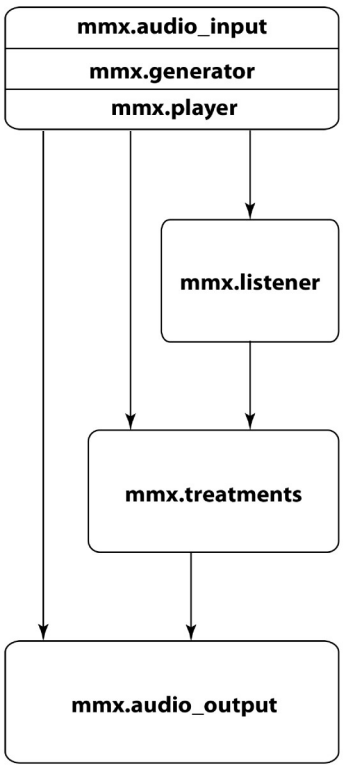


Fig. 5.38 Design pattern for *MMixte* n°5

The above diagram illustrates *MMixte* architecture for the audio block. Max's *presentation* mode implemented in the modules involves MVP design patterns working independently in management and audio streaming elaboration.
Midi controllers can be programmed to change parameters for other modules as well, aside from governing virtual instruments programmed in *mmx.generator*. A second midi set-up such as a controller is commonly arranged according to the following configuration:

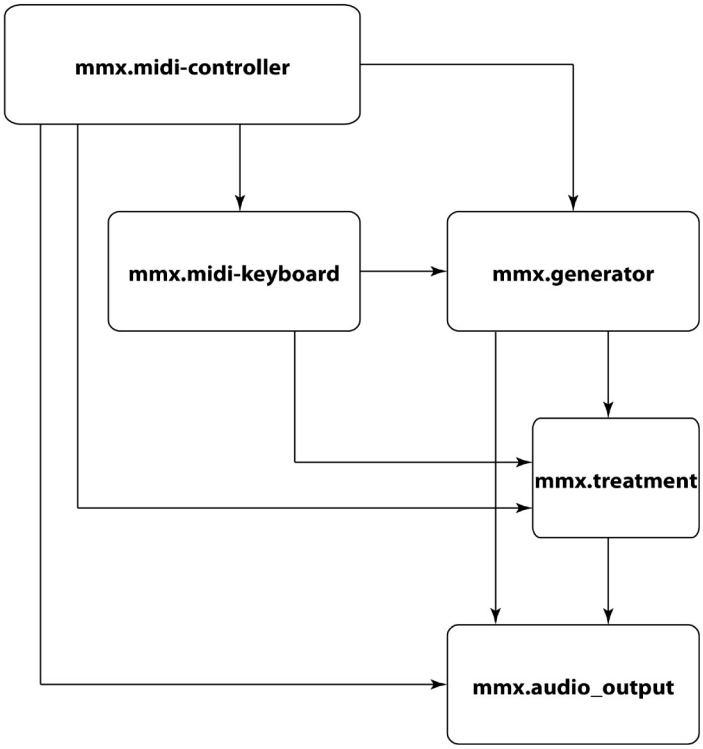


Fig. 5.39 Design pattern for *MMixte* n°6

The introduction of a computer score to split the instructions for middleware status change across several events alters the architecture in this way:

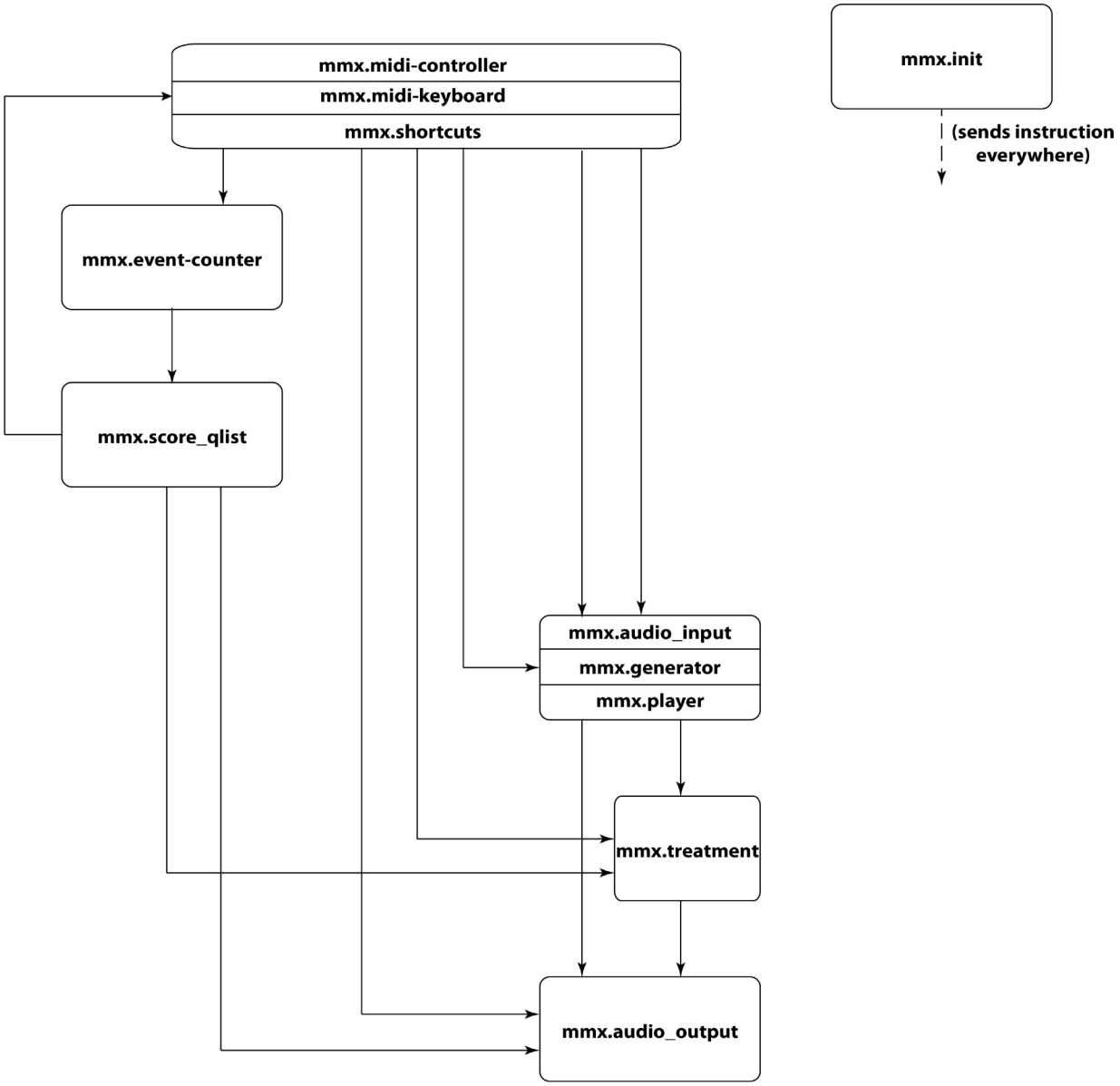


Fig. 5.40 Design pattern for *MMixte* n°7

The *mmx.score_qlist* module is indicated for the computer score; the *mmx.score_select* module performs the same function.

Fig. 5.40 illustrates only one among other architecture possibilities since various intermediate configurations are possible too. The structure of *MMixte* always leads to pipelined architecture of some kind; an external device can access event progress with instructions for middleware status changes which, in turn, can access the remaining modules. The initialization module can access all modules in middleware pre-use phase.

In figure 5.41 all modules linked to midi data reception and transmission are grouped in a single class. The audio block is completed with the *mmx.listener* module while the remaining modules complete middleware functionalities from the standpoint of usability and *free hands* control:

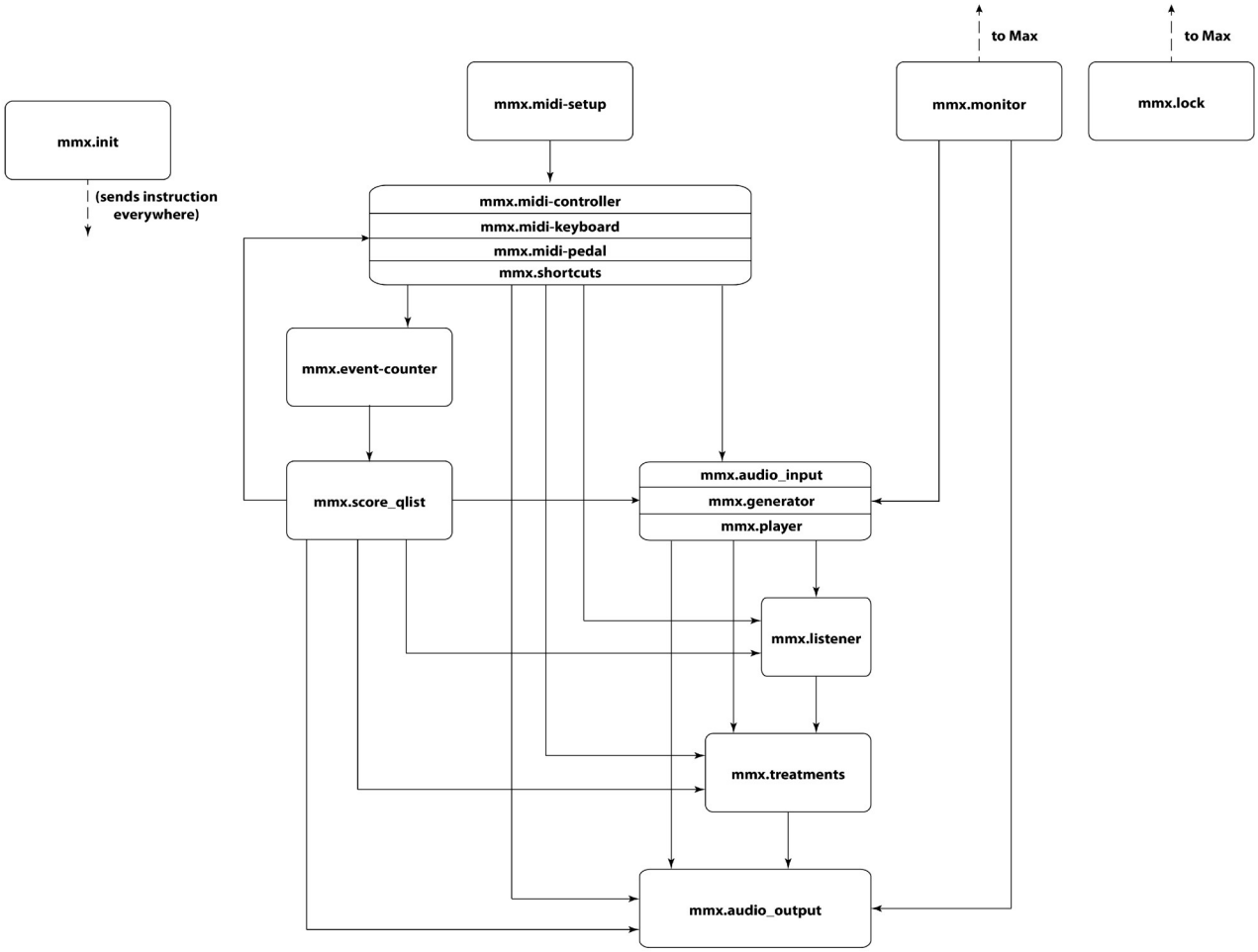


Fig. 5.41 Design pattern for *MMixte* n°8

The *mmx.lock* is disconnected from the rest of the architecture because its function does not concern the data and signal path but the Max software operation. In modern hardware architecture, DSP is part of the computations performed by the CPU. *Mmx.monitor* can handle DSP parameters and therefore constitutes an MVVM-type control in the architecture between CPU and Max. The amount of use of the CPU is, for example, a dynamic datum which can be limited by an appropriate setting; in this sense, mmx.monitor is an element of software architecture for what concerns the path of the sound signal, the settings of the audio devices, the driver etc.

At a higher level of module structure overview, all seventeen package modules can be regrouped in three categories:

- **control modules through human interfaces**: modules receiving and trasmitting digital data from protocols (such as midi) and digital information from computer status. These modules therefore enable both computer-machine and trans-software communication. Controls include a *human interface* which is also the external layer of other MVC, MVP or MVVM design patterns;
- **control modules through script**: they enable control functions without an interface for all modules described in the previous category. While, through HI, controls enable *gaming* in real time performative action, scripts require *a priori* programming: the sequence is therefore pre-ordered and deterministic;
- **audio control modules**: all modules sending and receiving audio signals from ADC up to DAC codification.

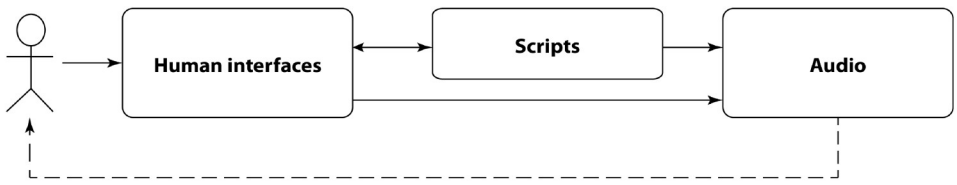


Fig. 5.42 Functional interaction between users and modules

The arrow indicating connections between audio modules and users implies that, from the standpoint of performance, diffused audio affects the way users enter data through HI, thus integrating them into the system.

Part 2 - Description of works

Introduction

From 2013 to 2018 I was fortunate enough to receive commissions for more than 15 pieces for the most disparate groups ranging from solo instruments to large ensembles, up to opera and installations. Most of these commissions are the result of personal collaborations with musicians and festivals or public institutions such as the Basel Elektronisches Studio or Karlsruhe's ZKM. Most of them are mixed music works written for one or more instruments and electronics. My publisher usually asks me to label the pieces[71] with the term “electronics” when they are performed in deferred time, involving audio files played while the piece is being performed; “live electronics” when real time audio elaboration is involved. If processing for the acoustic source is limited to amplification and reverb, the term "live electronics" is substituted by the label “amplified instrument”.

For quite some time now, I have found such a definition restrictive for a number of reasons. First of all, even in cases simply involving audio files to be diffused, I could legitimately speak of real time elaboration. Such pieces of acoustic music as my own *Jardins de fabrique* (2014), in its original version for a 4.1 diffusion system, are especially emblematic. In November 2017 it was diffused once more in the ZKM Kubus in Karlsruhe for the Giga Hertz Award days. The ZKM Kubus is a concert hall with a fixed diffusion set-up of 43.4 [72] speakers. When granted the opportunity, I decided to specifically apply to this circumstance diffusion of the four original channels to four 1.5 m high speakers and take advantage of the entire speaker set-up for reverb diffusion in addition to a few audio tracks created for the occasion. The whole system worked through an ambisonics algorithm automatically distributing four channels over the whole set-up of the half-sphere with five speaker rings.

For this piece, diffusion was substantially that of a four-channel audio file; but audio elaboration for ambisonics spatialization involved what I may call, in all fairness, "live

[71] Edizioni Suvini Zerboni, Gruppo Sugar Music, Milan (Italy).

[72] Forty-three loudspeakers organized on five rings and four subwoofers.

electronics" since it was the outcome of computer calculations performed as the audio streaming kept playing.

It is the opposite, however, for live electronics. This label usually implies real time audio transformation: audio in input is therefore perceptibly different from audio in output from one or more speakers. One piece after another, my way of using electronics has been changing. Indeed, it is now following the direction of real time audio capturing to manage machines or parameters of other instruments. For instance, when I realize that in a chamber group with heterogeneous instruments there may be acoustic imbalance in specific registers between the instruments, I rely on a personal technique one might call “auto-mix”: in practical terms, every instrument is connected to an envelope follower and the digital data resulting from such analysis is compared to other digital data of other envelope followers so that the coefficients of amplification of various instruments may auto-adapt, as it were, depending on the extent of dynamics performed by the other instrumentalists. This is what happens in *Meccanica degli avatar* (2016) where electronics perceived from the speakers are coming mainly from audio files while real time elaboration imperceptibly amplifies, whenever necessary, acoustic sources. One final section of real time electronics is diffused through a contact loudspeaker on the Timpani drum. Digital wave generators programmed with Max are diffused in mono in a dedicated channel. The result is a perceptibly acoustic sound since the electronics are filtered by the resonating body's material of the Timpani drum.

In my works, the use of contact loudspeakers on large surfaces is recurring. The first instance was *Lost in feedback* (2014), followed by *Radio Jail* (2015), *Frammenti senza cornice* (2014), *Tralci* (2015), *So loud* (2014-2017).

Two motivations have led me to the frequent use of this diffusion technique:

- with the use of contact loudspeakers, the relationship between acoustic instrument and the electro-acoustic source radically changes. In classic electro-acoustic chains, real time audio elaboration alters the physical features of acoustic sounds both in terms of dynamic (equalization, compression, amplification) and spectral parameters (spectrum elaboration). With the use of contact loudspeakers, electronics undergo changes in terms of their physical features depending on the material of the acoustic speaker since such material filters audio streaming. In other words, the acoustic instrument changes the electronics' physical, dynamic and spectral features.

- intimacy created between the electro-acoustic source and the acoustic speaker helps the production space of digital audio to coincide with the diffusion site. This principle is applied to *smart instruments* in its entirety where real time audio elaboration of the instrument captured with microphones may be diffused in the resonating body of the instrument itself. Such unity in terms of space is the solution to one of the most significant problems of mixed music, namely the unsolvable dichotomy between acoustic sources (instruments) and digital audio (speakers).

I have developed this conception of electronics while writing my pieces, programming their electronic parts and conducting personal research transcending the clichés of mixed music. The spatialization solutions I have chosen in my own pieces and the technique through which I elaborate the audio for such diffusion systems is an expression of the principles described here. *MMixte* has been a costant element in all of my mixed music works, evolving over time and yet slipping into standardization quickly enough. With a humble attitude, I admit that the genius of electronics is to be found in the way audio is processed: as far as *MMixte* is concerned, I am referring to the personal content every composer includes in the *mmx.treatment.maxsnip* module*.* At the same time I firmly believe that procedural order and “cleanliness” in programming can be determining factors in the quality of computer programming and hence, indirectly, of “audio quality”.

MMixte forces users to be orderly, schematic, and to pursue the motto "*one place for everything and everything in its place*"*.* When users are really knowledgeable in computer technology, this may translate into more aware management of information threading so as to avoid the kind of discontinuity which, in the most remote cases, is likely to spoil the results in terms of audio. It is a compositional method applied to programming and, when at its peak, it represents the way technical solutions might turn into an aesthetic paradigm. *MMixte*'*s* neutrality makes it easier to adapt to any situation regardless of musical style, of diffusion purposes, and of the complexity of the electro-acoustic chain.

In the following pages, I shall be briefly describing the pieces of my catalogue composed throughout the period devoted to this PhD wherever the application of *MMixte* is relevant.

6. Lost in feedback (2014)

6.1 Background of the work

After three years spent at Ircam studying and working on various projects, my interests shifted towards experimenting with new spatializing solutions and new strategies for the use of electronics falling outside the standards of mixed music repertory set in the last few decades. Throughout the years 2013-14, I have had the opportunity to write works of great personal significance since they have, in some ways, transfigured my own writing and especially my conception of composition. Drawing from extra-musical techniques, I have tried to conceive musical creation as an effort among different artistic synergies. This particular phase isencapsulated in *Lost in feedback* (2014), a work of about thirty-one minutes in length for electric vibraphone, percussion, stage performer and live electronics, commissioned by the Hanatsu Miroir Ensemble[73] and premiered on July 5th at Espace K in Strasbourg.

This work's strength lies in the number of contributions enhanced by real time electronics: adding sensors to the vibraphone[74] along with a few unconventional performance techniques; producing a live painting with the preparation of the canvas so as to contribute to the real time electronics ad hoc audio diffusion set-up, thus merging a personal spatializing solution with the stage set-up.

The overall result may be subsumed to the generic definition of “musical theatre” although at times, due to the actual presence of a visual artist on stage, it may be more accurate to categorize this work as “performance”.

With the production completed for the premiere, repeated with all the adapting involved in the upcoming performances (another performance of the piece featuring the same performers, was on October 5th 2017 at the Venice Biennale), along with dedicated costumes and the lighting design included, we have achieved results which, in my opinion, make *a priori* definitions much too restrictive: if music is the principal actor in this work, it also branches out towards other theatrical arts, with improvisation being added in painting for the final rendition.

---

[73] Olivier Maurel, Percussion; Yon Costes, performer; Marie-Anne Baquet, scenography and real time video; Raphaël Siefert, lighting design.

[74] Credit for changes to the vibraphone is due to Olivier Maurel.

Along with two other works of mine, *Corpo d'aria* (2013) for bass flute, shadows and live electronics[75] and *Stesso Obliquo* (2008) [76], a DVD based on the show [77] was released in 2015.

6.2 Instrument set-up

The percussionist works with a percussion set-up made up of:

- 1 Vibraphone (F3-F6);
- 1 Glockenspiel;
- 1 Spring Drum;
- 2 Bongos;
- 1 Tam Tam;
- 1 Thunder Sheet;
- 1 set of Crotales;
- 1 piece of polystyrene.

Some of the accessories are prescribed within the score for the performance of the solo section:

- 1 E-bow;
- 2 "Reibstäbe" sticks (knurled wooden sticks reminiscent of Guiro);
- 2 Double bass bows;
- 2 brushes for snare drum;
- 2 sets of springs for snare drum;
- 5 vibrating razors.

1 Timpani drum for resonance is set back stage. The percussionist does not directly perform on this instrument; a contact loudspeaker is placed on top of it, transmitting a few audio files and some of the real time processing.

[75] Edizioni Suvini Zerboni, Milano.

[76] Edizioni Arspublica, Carrara.

[77] Maurilio Cacciatore, *Lost in feedback*. DVD. MAP Editions, Milan, 2015.

6.3 Electric vibraphone

The vibraphone is set up with artisanal means including a series of pressure sensors underneath the bars. In order to reduce the number of input channels, and in light of the piece's demands, sensors flow into three audio channels corresponding to the instrument's three octaves - F3-E4, F4-E5, F5-F6.
As far as digital elaboration is concerned, two strategies are used for signals coming from the vibraphone. On the one hand, sensors enable instrumental signal amplification and its eventual “in line" processing, avoiding problems related to the electro-acoustic chain likely to arise when using a microphone (e.g. feedback). This enables the vibraphone “local” amplification by means of an electric guitar amplifier placed below the instrument.
At other times, the digital data of the signal amplitude captured by the sensor is used as a *trigger* for real time processing presets, as a *trigger* to launch audio files or as conditional of an *envelope follower*. For the latter case, the sensors' sensitivity - meaning the input signal volume in the dedicated audio card - matches the sensitivity of the electro-mechanical device; the combination of the instrumental section dynamics and the sensors' sensitivity below the bars required ad hoc work to make the software program's response appropriate.
Beyond the context of this piece, the sensors underneath every bar might eventually be connected to independent audio channels. In this case, 37 channels would be required to send the information coming from each bar over to independent audio channels.

6.4 E-bow

A number of unconventional techniques contribute to the performance of the percussion part. Here I only provide an account of what, to my knowledge, does not appear in works prior to my own. . These are, for the most part, performance techniques producing sounds that require amplification and/or computer processing. Whether for reasons having to do with sound intensity or timbre, the material I elaborate is often the outcome of research in the analog world of sounds that might be granted admission, so to speak, to the realm of “digital” sounds, by which I mean computer-generated sounds. Finding sound sources already possessing timbral complexity within them is central to my research in composition[78].

[78] In my piece *Vit_Vite_Evit* (2015), for example, I take the challenge of simulating electro-acoustic sounds with no intervention of electronics.

The use of the E-bow on instruments other than the electric guitar and the electric bass has become one of the most prominent aspects of my own writing over the last years of production. The magnet located within the device can affect fine metal objects such as piano strings; in this piece, it is used on the Spring drum (a cylinder-shaped small percussion instrument with two artificial fibre skins on the sides). A hole on the wood enables sound projection. A suspended metal spring measuring about 40cm in length is attached to the lower skin with a hook. Such springs are usually shaken up in the air or pulled with two fingers, thus transmitting a vibration to the instrument's body that may be modulated with the hand through the hole on the wooden body. In this specific case, the E-bow rubs the spring while keeping contact through its magnet. When touching the spring, the E-bow produces a weak sine wave which, once appropriately compressed, contributes, together with real time elaboration, to creating the sound I am interested in. The friction of the E-bow's plastic with the rough metal spring is really helpful in creating a sound signal that is somehow already distorted in the acoustic environment, namely before the signal's analog to digital conversion. The miniature here below is an excerpt of the score for the piece, representing some of the actions performed. Once the signal enters the computer, further distortion helps emphasize the instrumental source's acoustic traits. In other pieces I have preferred the use of neodymium magnets [79] but in this case the E-bow's action enables getting a continuous sound even in the absence of continuous friction on the strings. Such necessity precludes audio espressivity (in the sense of variation of dynamics) owing to a considerable amount of compression (with a ratio of 40:1 and relatively high amplitude gain) applied.

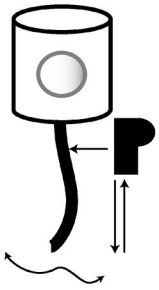


Fig.6.1 Miniature selected from the score for Lost in feedback indicating the actions to be performed with the E-bow on the Spring Drum

[79] M. Cacciatore, *I don't need to …k for music* (2016), for two electric guitars and live electronics. Edizioni Suvini Zerboni, Milan.

6.5 Double bass bows

The use of double bass bows on vibraphones and on polystyrene is well-known . Beyond the usual use of bows on the vibraphone, that is, for the production of longer continuous sounds, the choice for this instrumental technique throughout the piece is further grounded in the integration of these tools with the stage costumes. Reflective strips are attached to them as well as on the percussionist's arms. When the lighting designer decreases the lights' intensity and turns the neon lights towards the instrument set-up, the reflective strips light up, making

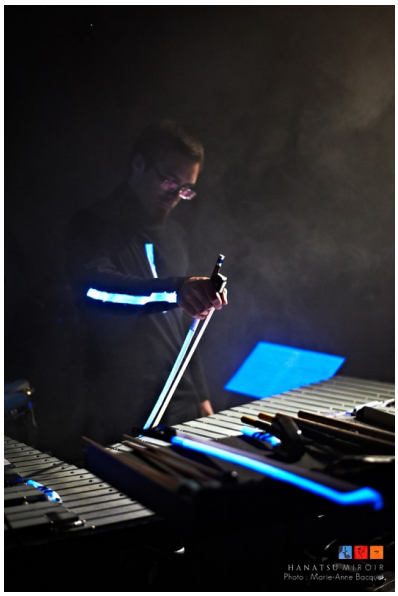


Fig.6.2 Olivier Maurel, percussionist with the Hanatsu Miroir Ensemble in the course of the creaton of *Lost in Feedback*

the arms' movement visible as the bows move up and down along the surfaces. In combination with a black stage costume, luminous white strips moving in time with the bows are the effect resulting on the percussionist.

On the polystyrene, the bows produce a sound emphasizing the entire spectrum; unlike instrumental sounds, which usually present spectra with decay of higher components, the bowed-polystyrene produces spectra where high components possess intensities directly proportional to their frequency. (ie. spectral increase of higher components).

The bow's position and the pressure exerted throughout the movement contribute, in essential ways, to creating different sounds. The sound response is enhanced when using low density polystyrene.

The sonogram below[80] illustrates the way the centroid frequency and the "weight" of sound gesture - actions performed with a double bass bow over its whole length - shift considerably towards the highest frequencies towards a region where instrumental sounds usually no longer have any components. All of the spectrum's regions[81] are emphasized.

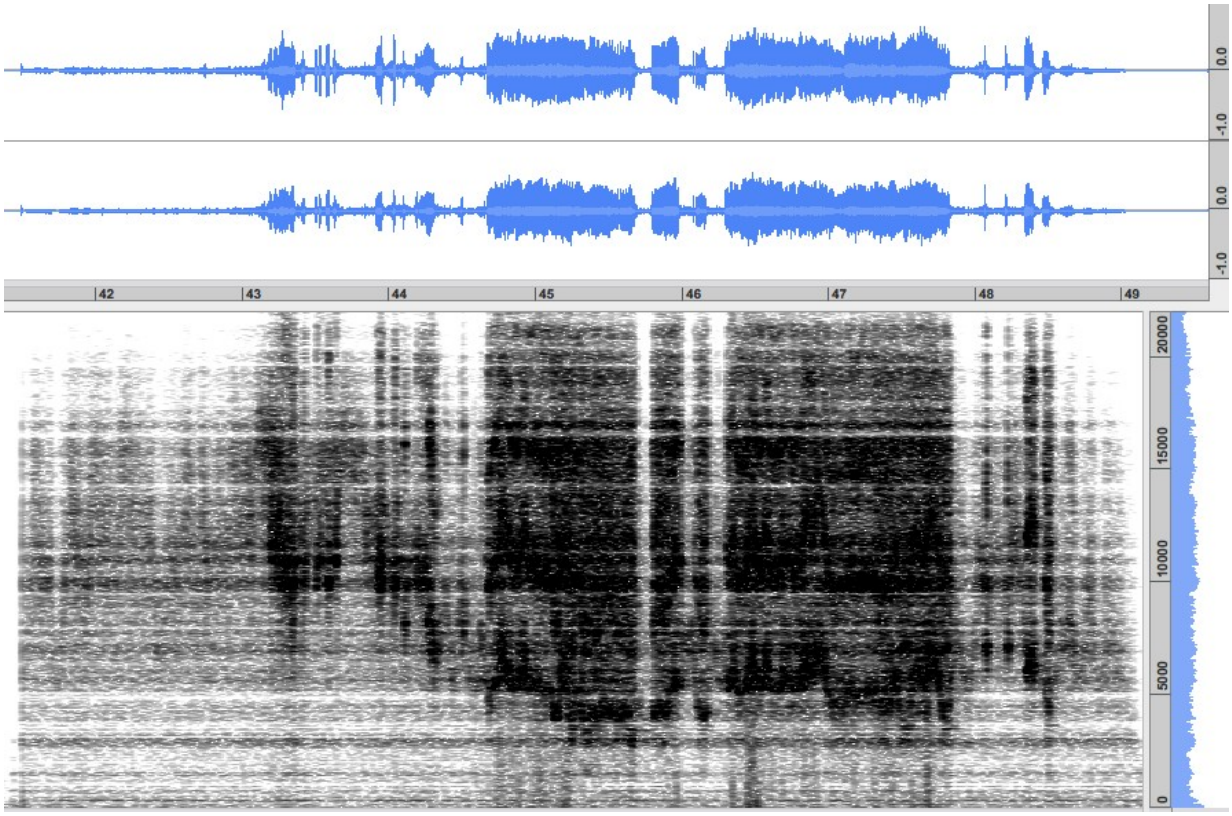


fig.6.3 Sonogram representing sounds obtained with polystyrene through rubbing a double bass bow

6.4 "Reibstäbe" Sticks

The sound these kind of sticks (fig. 4) produce on wooden surfaces is similar to a Guiro's. These wooden sticks measure about 1cm in diameter with a series of grooves all over their length; bowing with these sticks on the vibraphone bars produces an iterative sound whose repetition frequency, as well its dynamic, is directly proportional to the

[80] All the sonograms are produced with *Audiosculpt*, version 3.4.5. For the sake of space, only the left channel is illustrated even though the sounds were recorded in stereo format.

[81] The audio was recorded in studio using a Neumann KM 184 microphone.

stick's changing speed. The pitch of the sound produced depends on the vibraphone's bar being rubbed; the attack of every single sound is marked by the grooves all over its length. The result is an effect similar to a continuous granulating sound whose roughness is directly proportional to the speed of gesture. Fig. 6.4 shows the symbols to indicate Reibstäbe in the score:

Fig.6.4 Symbols of the Reibstäbe sticks for the score for Lost in feedback

6.5 Vibrating razors

The body of vibrating razors (fig. 6.5) produces a vibration transmitted on the contact surface and whose intensity, in the presence of another resonating body, increases taking on timbral characteristics resulting from the material the body is made of. In my own production since 2013[82], the use of vibrating razors is prescribed in a number of pieces; indeed, they have been used on skin percussion, plates, strings and on the piano.

Fig.6.5 Currently available Gillette vibrating razor and its stylization in the symbol used in the score for Lost in feedback

Easy to find and inexpensive, Gillette razors with a vibrating tip are highly recommended when performing these parts. Two kinds of sound are possible. The first one may be produced by applying very light pressure on the contact surface so that one side of the

[82] The first piece which included the use of razors was *Radio racconti appena accennati* (2013), written for orchestra and electronics. Edizioni Suvini Zerboni, Milan.

razor (the tip is one part of the plastic where the vibration is not adequately transmitted) may freely vibrate.

The second one is a continuous sound, mostly in the low register (depending on the contact surface and conformity with the resonating body), and it is produced through action with considerable pressure on the contact surface so as to prevent, as expected, the razor from bouncing.

From a mechanical point of view, the vibration transmitted by the razor produces micro-percussions at a steady speed rate. Here below is a body of a vibrating razor on a wooden table recorded with a XY head of an H6 ZOOM:

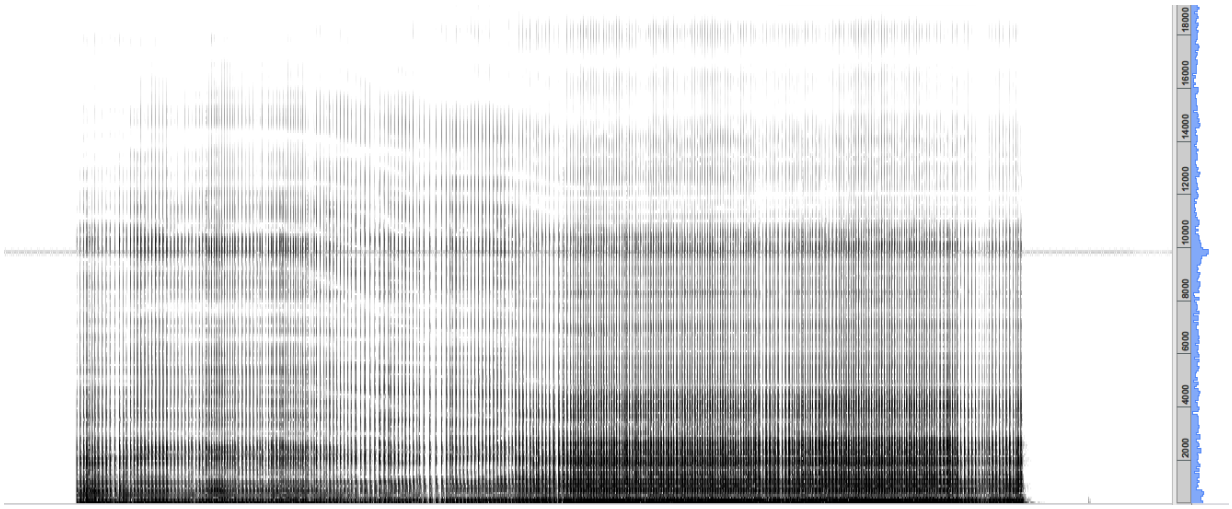


Fig. 6.6 Sound spectrum produced by the vibrating razor's plastic tip leaning on a wooden table without any pressure

The frequency bandwidth slightly below 10 kHz is set by the razor when not touching any contact surface. Strong pressure produces, instead, this kind of answer:

Fig. 6.7 Sound spectrum produced by the vibrating razor's plastic tip touching a wooden table with considerable pressure

The 10 kHz continuous vibration is still apparent. Schertler contact microphones do not capture this frequency band, and are therefore used to avoid the acontinuous presence of such (in this context) “parasite” sound. The razor's initial movement on the surface produces
a series of micro-percussions such as the ones apparent on the sonogram of Fig. 6.6: they stopp as soon as adequate pressure is exerted.
The razor's continuous vibration may be somewhat attenuated by taking away a few bits of the plastic of the razor’s top but, by and large, I have preferred avoiding such an option because of the handy work it takes, not to mention the difficulty often involved in asking for it on production sites in the absence of my own direct supervision.
In *Lost in feedback,* both performance techniques including razors are notated for various instruments. The Spring Drum, also amplified with a piezo microphone, is also used with a vibrating razor too on the upper skin of the instrument - the piezo microphone is placed on the lower skin beside the spring's hook.
In *Lost in feedback,* razors are also used on Tamtam, Vibraphone and Glockenspiel.
On the vibraphone, a number of razors are used simultaneously; some are left vibrating without being handled, thus exploiting the gap between the bars of the natural sounds and those of the altered sounds. Other razors are used to play on snare drum springs previously installed on specific areas of the vibraphone, enabling the merging of the sound of the bars with the sounds of the rattles. Below, two symbols selected from the score:

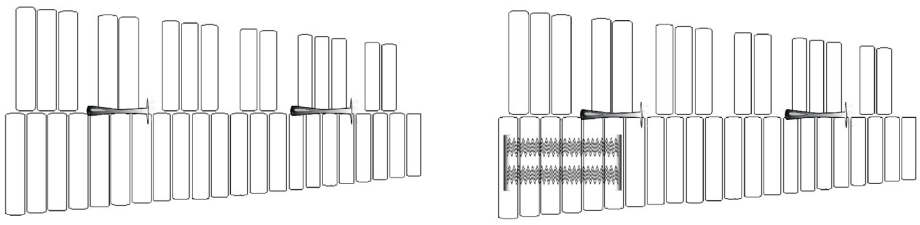


Fig. 6.8 Symbol of the vibraphone with snare drum springs and two vibrating razors

In the context of an arts residence in December 2016 with Les Percussions de Strasbourg[83], I have had many an opportunity for in-depth study of the mechanics of such modalities of sound production on a number of percussion instruments. I tried to

[83] I am especially grateful to Jean Geoffroy, artistic director of the ensemble, and to François Papirer, percussionist of the ensemble who helped me with the percussion instruments during my residency.

rationalize the production of certain sounds in particular on Timpani and Bass Drum: their response to these kinds of excitations was quite satisfactory.
The combination between skin tension and the specific point where the sound is produced (whether at the edge or at the centre) repeatedly produces different results.
The hand position on the razor also contributes to changes in pitch of the transmitted vibration.

6.6 The performance canvas

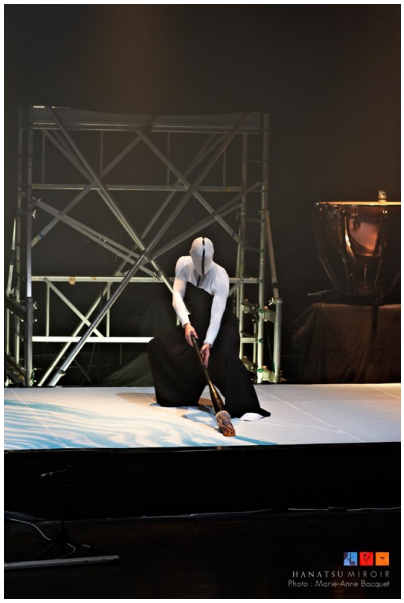


Fig. 6.9 Yon Costes performing Lost in feedback

The visual artist involved in the performance of the piece proceeds by painting with a calligraphy technique using ink and water over a canvas of 6 x 3m on top of which a few sheets of paper are placed. Underneath the series of white sheets, an equal number of blue sheets is placed so that, following the movement of water brushes, the color of the blue sheets emerges as a result of transparency.
As a performer, the visual artist is actively involved in generating live electronics while moving across the canvas, working on it at the same time. A series of piezos is set up underneath the canvas. This series of piezo microphones captures the signal every time the performer steps on them. The audio signal is not sent, due to the sound not-being interesting, but the digital data of the signal's amplitude is used to randomly launch a series of audio files contributing to the overall electronics. These audio files change throughout the performance (the overall duration is about 15', including ten minutes devoid of musical action from the percussionist) with some of the audio parameters, such

as volume or distortion amount and depending on the signal amplitude captured and the number of peaks over a threshold recorded by the software program.
The action painting is mixed with theatrical action from the artist[84], pursuing his motion on the painted surface with light steps. Below is the series of piezo microphones used underneath the benches supporting the canvas to be painted:

Fig. 6.10 Piezo microphone matrix for the transmission of information about the performer's position on the platform

The painted surface is large enough to allow the projection of such other capturing systems such as a matrix given by two still video images intersecting with a camera on the x-axis and another one on the y-axis. Wearable sensors can also be used. For upcoming performances, the paint performance's capturing systems is likely to change in relation to production schedule, available technology and possible collaborations with teams of specialists in gesture capturing.
For this work's premiere, I chose this solution for its “invisibility” from a staging point of view: the cause-to-effect relationship when using wearable sensors often suffer of being too obvious when watching the performance. To the extent that lighting design is inextricably bound with the result visible on stage, the lights variation poses additional challenges in terms of elaborating a video matrix with activation points. One possible solution is the integration of color variation as an additional parameter for the transformation of video information into audio data.

[84] To his skills as a visual artist, Yon Costes adds those acquired from the martial arts and, more recently, from contemporary dance.

6.7 The diffusion set-up

Diffusion of electronics is performed with a 4.2 system conceived as a pair of large stereo speakers, a front subwoofer (positioned backstage) and a second one to be placed under the seats taken by the audience. This choice was made so as to make the loud low frequencies, plenty of which are heard all through this piece, even more present, almost “tactile”.

The electric vibraphone is to be plugged into a dedicated amplifier (an Orange amplifier was selected for the premiere). This choice is coherent with the strategy of vibraphone processing, made up of a series of distortion pedals and other sound processors typical of the world of the electric guitar.

The fourth channel is a contact loudspeaker positioned on a Timpani drum: a DJ-Box “Thunder” model like the one in Fig. 6.11, leaning on the Timpani drum, capable of sending signals with 26 W in RMS (accepting both mono and stereo signals). The loudspeaker also features Bluetooth protocol but, for obvious stability reasons, the line connection was chosen. The audio “launched” through the piezo system underneath the painting surface is diffused from this speaker. The Timpani drum's resonating body amplifies the sound produced by this speaker, balancing the output sound pressure with the rest of the set-up.

The use of contact loudspeakers on instruments has become increasingly more frequent in recent years. Even before such practice had become widespread, I noticed that on the

Fig. 6.11 DJ-Box “Thunder” contact loudspeaker on a Timpani drum

Timpani drum results of the interaction between acoustic instrument and electronics ended up reversing functional roles in the A/D chain. Under normal circumstances, the sound of the acoustic source is transformed through capturing by transducers to be then diffused with some modifications either via software or via hardware. In this case, a digital

source (whether a sound of physical origin or a digital wave) is diffused in an acoustic instrument capable of mechanical action on audio timbre and pitch. In other words, changes in tension of the Timpani drum's skin effected with the pedal may change the sound perceived in terms of pitch and frequency richness[85].

6.8 Electronics and the title for this piece

The title of the piece comes from the challenge which live electronics poses throughout the whole piece. The speaker and microphone set-up is calibrated to produce hand-controlled feedback from the mixing board through the software program managing the electronics. This feedback, combined with audio files resulting from other feedback sounds (recorded by having a camera and a television set interact) constitute the high frequencies and complement the spectrum of other audio files with low centroid frequencies tending, from an aesthetic point of view, towards techno music.
Aside from the control of general volume levels and the launching of audio files throughout the piece, the role of the audio software operator is akin to a “juggler's” letting the feedback caused by the closeness of captors and speakers rise without overstepping overall audio volume levels. This may be a way of experiencing the mixing board and the electronics with the attention of a performer who might be deserving a place on stage but, to the extent that these computer actions are not visible by the audience, I have always preferred (as I have in other pieces as well) working from the mixing board or directing the work when not personally in charge of the electronics in a concert situation.

6.9 Applying MMixte

MMixte supports all the parts needed to manage the piece's live electronics. The number of sources necessary for creating live feedback was not stable enough until the definitive instrument set-up on the concert stage - the acoustic situation in the context of rehearsals was very different - since the impact of the objects' physical position and the distance between microphones and speakers was considerable on both the quantity and the quality of audio feedback. This is the reason why the use of *MMixte* has made the adapting of the control patch characteristics easier and enabled me to adapt the computer score directly reaching, during rehearsals, the version which I consider to be

[85] Another application on the Timpani drum is to be found in *Frammenti senza cornice* (2014); on the grand piano in *Radio Jail* (2014) and in *So loud (2014-2017).* Edizioni Suvini Zerboni, Milan.

the definitive one. Optimization of production scheduling of the software program and its graphic interface is one of the main reasons driving *MMixte* and throughout the production of this piece it was possible for me to recognize the efficiency of easy-to-operate middleware containing its own algorithms. Here is the *MMixte* architecture for *Lost in feedback*:

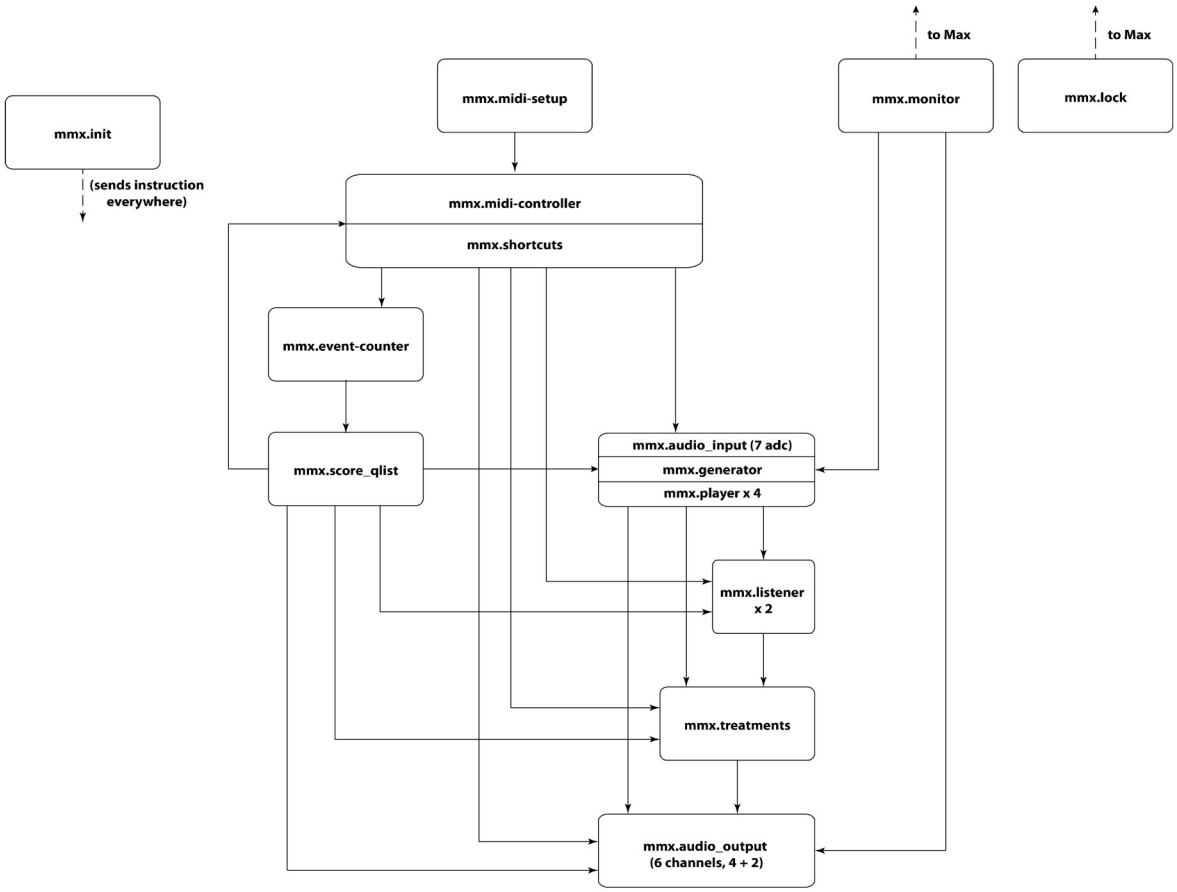


Fig. 6.12 *MMixte* architecture for *Lost in feedback*

This is the Max patcher with the *MMixte* modules for this piece in presentation mode:

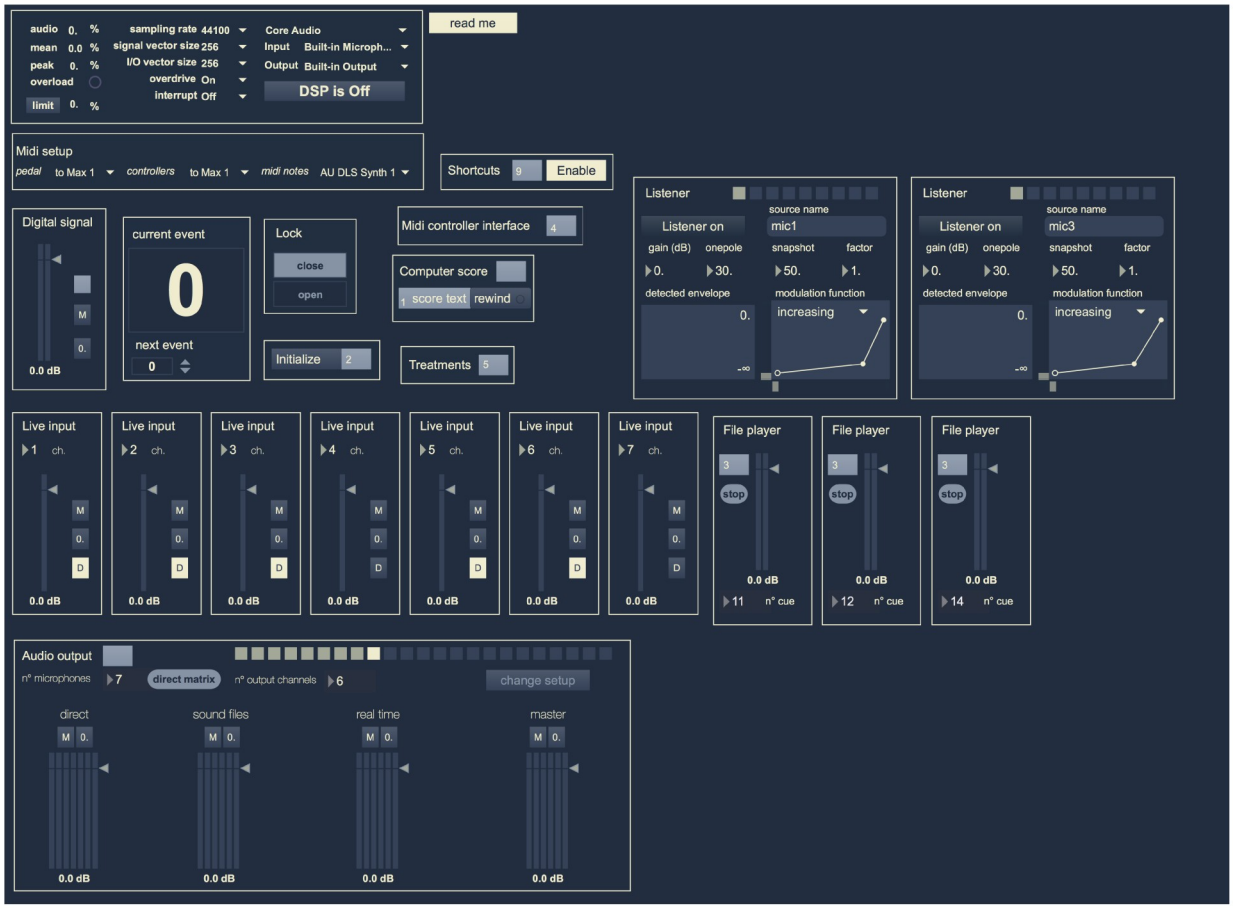


Fig. 6.13 *MMixte* modules in presentation mode in Max for *Lost in feedback*

7. I don’t need to …k for music (2016)

7.1 Background of the work

Written for two electric guitars and live electronics, *I don’t need to …k for music* was premiered on March 18th 2016 at the Luminale Festival in Frankfurt. The motivation for the title for this piece is twofold: on the one hand, my intention was to choose a title similar to one we might find for a piece of rock music or any other commercial genre, given the presence of two solo electric guitars and no other acoustic instrument; on the other hand, this piece is an ironic answer to J. Szmytka's *f* for music* (2012) for electric guitar and amplified cello[86].

The work pursues the direction set in *Lost in feedback*: against the idea of a mistake in speaker positioning and in balance between audio amplitude in input and output, audio feedback is used as musical material. To this, I have added instrumental techniques in an attempt to reduce as much as possible the connotation of instrumental writing for electric guitar in favour of more homogeneous writing with electronics played throughout the performance, whether from real time elaboration or audio files launched from the mixer.

7.2 Instrument set-up

This is the list of all the instruments and the material used in this work:

• Two electric guitars with the sixth string tuned down to D;

• two E-bows, one for each guitarist;

• a neodymium magnet, for the second guitarist;

• distortion, overdrive, and wah-wah pedals, one for each guitarist;

• two amplifiers proportionally sized to the concert hall, for each guitarist;

• two speakers to be positioned backstage, in stereo format. (fig.7.1)

[86] http://www.jagodaszmytka.com/works-f-for-music.html

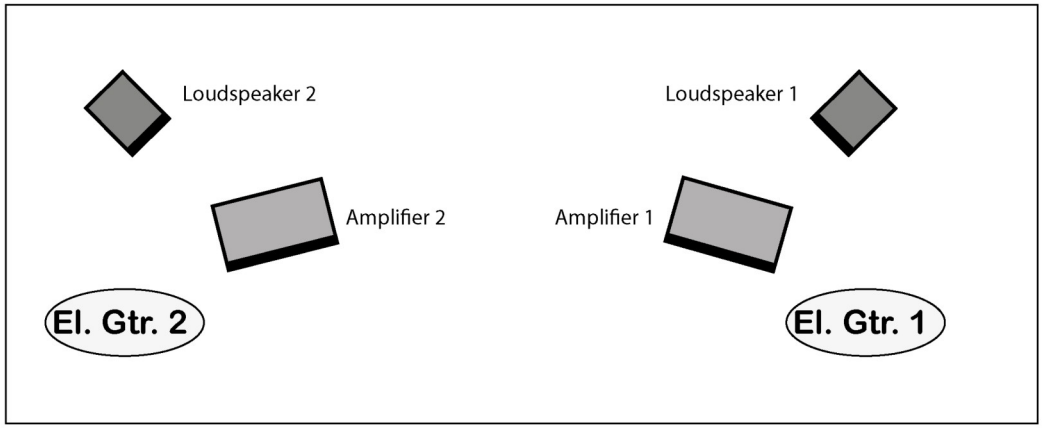


Fig. 7.1 Diagram of musicians and speakers arrangement

7.3 E-bow and magnet use

E-bows are well-known for their use on electric guitar. They are generally used whenever melodies require a stable, sustained sound without the attack typical of plucked strings. In this piece, the melodic lines' vertical motion is almost entirely absent, since the piece relies on motion along a micro-variation of the D2 unison (actual sounds) and along the changes in opening of filters and performance dynamics. Thus, attention through listening is shifted towards the timbral quality of the sounds themselves instead of their parataxis.

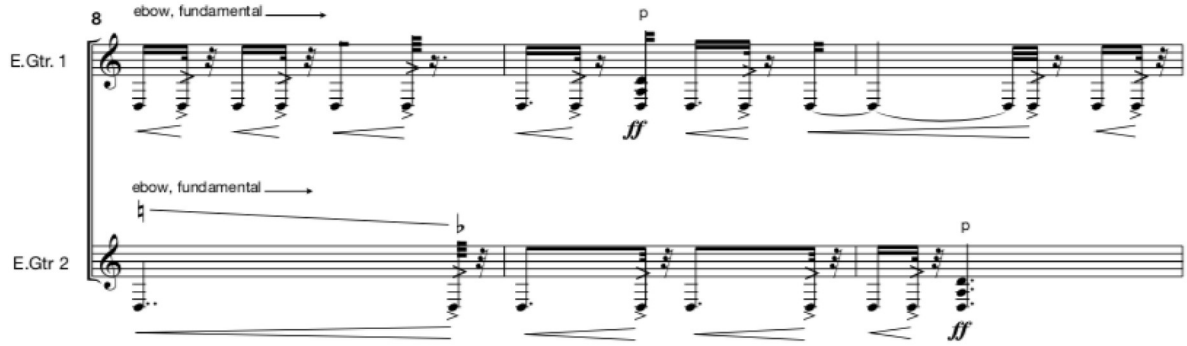


Fig. 7.2 Excerpt from the score for *I don't need to …k for music* n. 1

The sliding of the E-bow all along the string implies timbral variations much like the ones occurring in all string instruments whenever the string's vibrating point is shifted between its half - ordinary - and the bridge. The sound resulting from the friction between the plastic of the E-bow and the string contributes to the sound obtained in the end. The sound amplitude produced

through these frictions is directly proportional to the speed of the hand's shift along the string, notated as accents in the score.

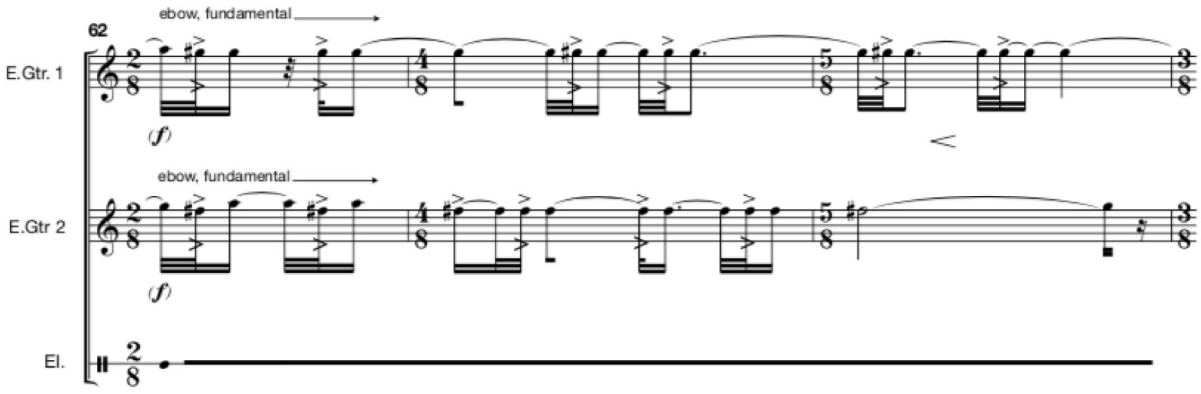


Fig. 7.3 Excerpt from the score for *I don't need to …k for music* n. 2

Timbral variation in relation to this way of using the E-bow occurs thanks to the integration of neodymium magnets in sound production. These magnets are extremely powerful despite their dimensions: they are capable of creating violent sounds when interacting with the magnets of the guitar pick-ups. The metal material of the lowest electric guitar strings are made of ensures the bare minimum of attraction guaranteeing a fair amount of stability for the magnet's longitudinal shift along the string itself. The

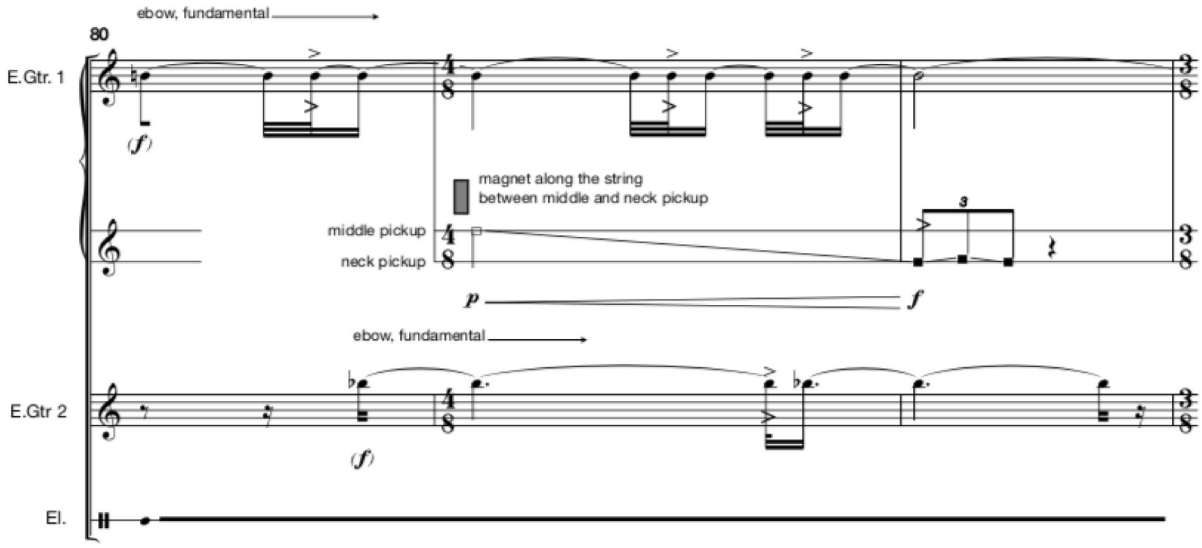


Fig. 7.4 Excerpt from the score for *I don't need to …k for music* n. 3

moment the magnet is positioned above one of the two pick-ups (or three, depending on the electric guitar model), feedback is produced, merging with the sound of the mechanical friction resulting from the magnet's rubbing action against the string grooves.

7.4 Diffusion set-up

Electric guitars are diffused in line through speakers and through microphone capturing using two microphones, one for each guitarist. In general, a large diaphragm condenser microphone is needed to capture both the sound coming from the speaker and the sound from the instrument; as I have indicated above, some of the mechanical sounds coming from the instrument contributing to the elaboration are heard in real time. The computer-generated sound sent by the microphones is then diffused through the speakers located at the back of the hall.

The speaker and microphone position makes the set-up rather unstable. A high degree of signal compression is necessary though at some points throughout the piece the eventual rise of audio feedback is welcome; the processing system of the sources is set for handling and managing these sounds just as the Max patcher graphic interface set for live electronics management enables immediate action on source and speaker volume levels.

7.5 MMixte Architecture

This is the diagram for the *MMixte* modules used for this piece:

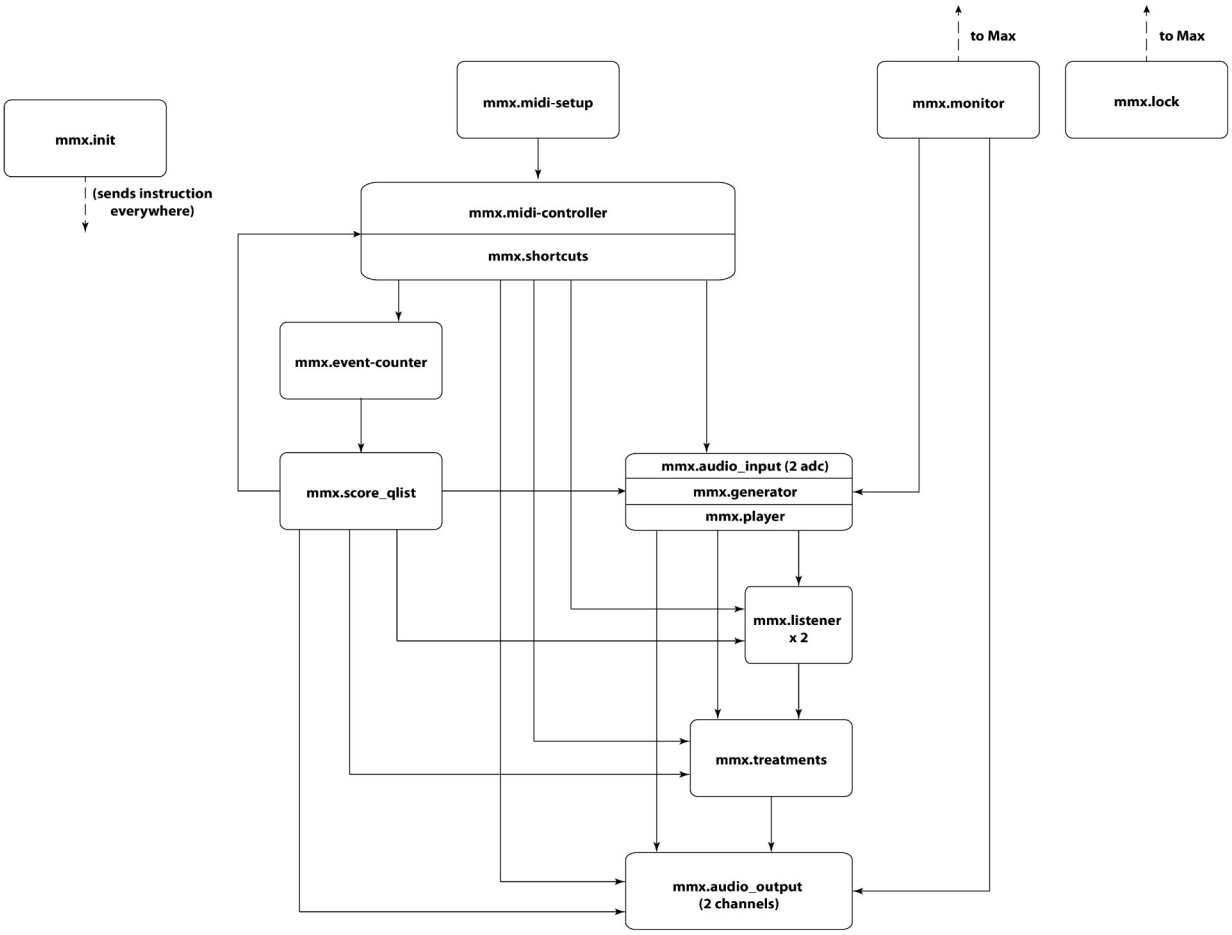


Fig. 7.5 *MMixte* architecture for I don’t need to …k for music

Since the amplifiers for electric guitar are directly connected to the instruments, without going through the composition's electro-acoustic chain needed for electronics, only two output channels are needed. Thus, the electro-acoustic chain made up of electric guitars, effect pedals and amplifiers, runs parallel to the one made up of microphones and amplifiers back stage. The patcher in presentation mode looks like this:

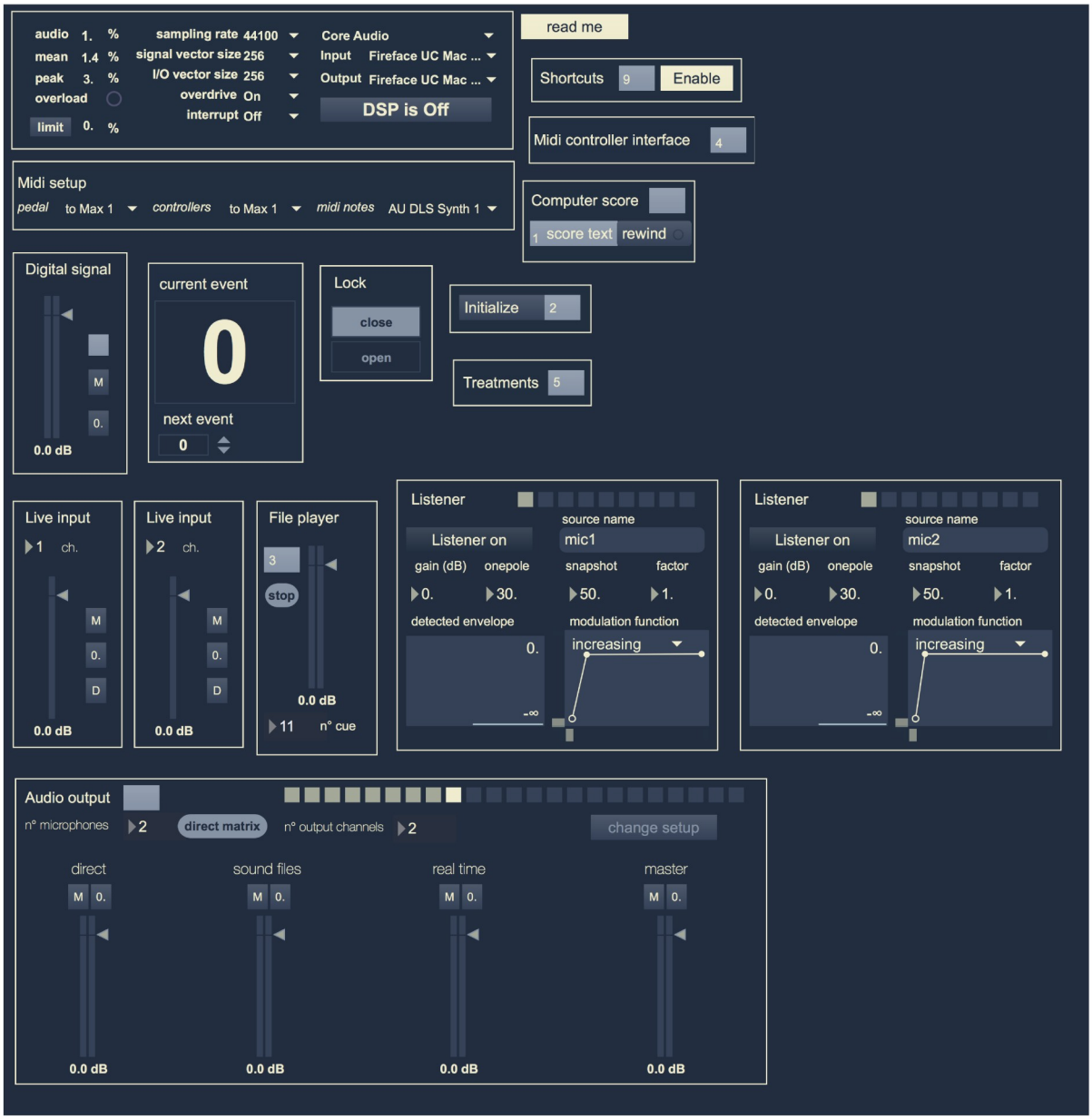


Fig. 7.6 *MMixte* modules in presentation mode for *I don't need to …k for music*

8. Tutorial 1: #mimesi (2018)

8.1 Background for the piece

This piece is the first one in a composition cycle drawing inspiration from video tutorials found on YouTube or other similar video platforms. These videos offer explanations as to how to go about specific actions including those we might need to be performing in every day life, such as repairing household equipment or making recipes and, in musical contexts, how to deal with performance techniques on an instrument or how to come to terms with a difficult passage. At the time, some of these videos were really helpful to me in terms of providing me with one or more sound options for comparison whenever doubts about writing my own works set in (people thousands of kilometers away who are willing to offer their contribution to the same topic in their own, independent ways upload quite a number of such videos).

The second source of inspiration were the *Tags* used in social media; the iconicity of words used for *tags* stretches their meaning towards conceptual categories. This enhances readers' understanding of the texts' meanings to such an extent that categorization simplifies the process of analyzing symbolic content. Assigning a category focuses on a topic and makes us re-assess understanding of a message on the basis of that point of view. Thirdly, and finally, this cycle draws inspiration from Fausto Romitelli's well-known *lessons*. The composer's *bad trips* cycle has brought inspiration for the title metaphor in an absolute sense: the idea of a *lesson* about something. *Tutorial* is the name of this cycle seeking perspective on some aspects of my musical writing at the end of the years ‘10. #*Mimesi* is the first number in this series: male voice and a sub-bass recorder use their common ground, merging into one another without relinquishing their own physiology, let alone their writing, their own rhetoric, their own role in general. Mimesis is a technique of sound construction I learned through my French apprenticeship years. It enables the connection between instruments and electro-acoustic sources through juxtaposable, if not common, *timbre*. This piece attempts to achieve all this, starting from such instruments as the sub-bass recorder and the male voice whose writing carries strong connotations, and taking into consideration all the difficulties which the human voice implies in writing chamber music. I approach the voice like an instrument at first, and therefore with no real words being uttered; gradually, a text emerges (only words taken from a variety of languages put together in sentences without meaning; I chose

them only for their “sonic value”. Yet, they go through some kind of “mimesis” by virtue of their foreign-ness in a language we do not know), though only for such a short time that dwelling on critical understanding simply isn't possible. Erasing the sense of time is helpful for the mix of *timbre* in the absence of clear segmentation of melodic material; therefore a very slow tempo, often reaching the limits of the musicians' breathing capabilities, accompanies the entire work.
Tutorial 1: #mimesi is a commission for the UMS’n JIP duo; its world premiere is scheduled for November 2018 in Switzerland, followed by a series of concerts.

8.2 New Font for Microphone Indications

From this piece on, I have started to use fonts of my own creation, called *Microphones* (Fig. 8.1), developed in collaboration with my student Giovanni Lando in one of my video writing classes. They enable indications for the main microphone types and polar diagrams directly on the score. The commercially available microphones I have drawn inspiration from the symbols drawings are indicated beside each character.

The use of these fonts is motivated by the idea that choosing a microphone is not only a technical issue, but it is a true compositional act:

- the choice of a microphone's position, especially in reference to small capsules to be attached to the bodies of the instruments through suitable equipment, helps raise sounds up to a level of audibility which would have otherwise been too low in the overall sound balance of the piece's sources if not adequately amplified;
- the choice of microphone type in deciding where the use of audio feedback may be integrated and where live electronics, is a common strategy that is intimately tied to dynamics as written in the score all through the piece. In my experience, I have realized that *a posteriori* choice of the microphone to be used for audio capturing often implies having to alter dynamics written in the score in order to balance the needs of live electronics and audio diffusion. When the use of a certain microphone type is officially integrated in the score, it is with sharper awareness that the composer can take into consideration the meaning and the extent of all written dynamics since it is already transposed to the dynamics resulting in concert once sources amplification and audio processing are active;

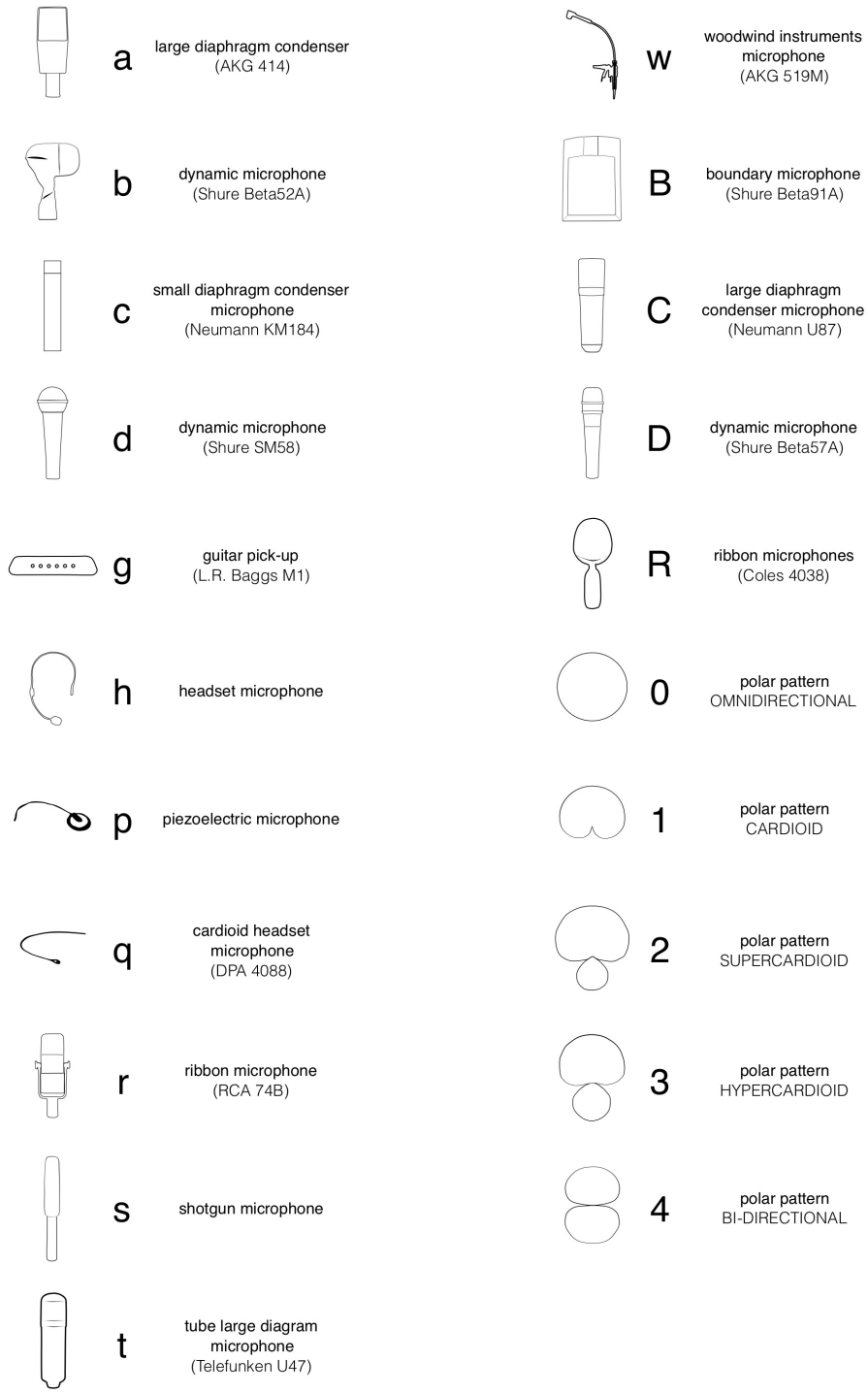


Fig. 8.1: *Microphones* font

Even when show or concert production can afford it, there is no guarantee that for a given instrument a single microphone will be enough to cover all the necessary functions for the best possible acoustic rendering of a piece. What might occur, indeed, is that classic sound design for diffusion of a source (amplification, equalization, compression, and reverb) might need a different transducer from the one needed to capture audio at crucial times throughout the piece when sounds with different physical characteristics need to be captured respecting what is written in the score. One of the emblematic cases of such assumptions is the use of piezo-electric transducers. Even when the best transducers are used, audio quality is always inferior to the one produced by an aerial microphone; conversely, accuracy of a sound's attack or any sound depending on contact through materials (or hands) on the instrument's surface will be more convincingly rendered by a piezo-electric transducer than an aerial microphone.

There is no comprehensive rule for these choices, since subscribing to a rule would mean setting limits to artistic creation - which is an oxymoron. Microphone choice, in my case, is a part of research in instrumental pre-writing, a fundamental aspect enabling obvious correspondences between the types of sounds prescribed in the score, their performance dynamics and the idea of sound result they will be producing at the end of the electro-acoustic chain responsible for real time audio processing.

The use of *Microphones* in the score refers exclusively to amplification and capturing audio for its real time elaboration. Recording a concert might require the use of other microphones and, since these are also chosen in relation to the acoustic characteristics of the hall and the cable set-up, if and when available, it would be futile to prescribe, within a score, elements likely to change in relation to the local characteristics of every performance, whether live or in a recording studio.

Indications for the microphones needed for the male voice and the sub-bass recorder are notated at the beginning of the piece:

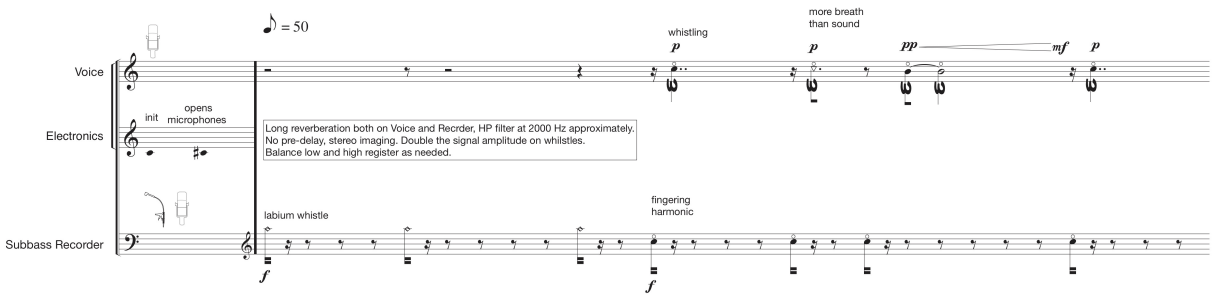


Fig. 8.2 excerpt from the score for *Tutorial 1: #mimesi*, p. 1

Distant Timbre as Challenge to Mimesis

The male voice and the sub-bass recorder have no timbral elements in common by the use of their standard ways of playing and singing. This was a major obstacle for one of the most crucial reference points in my compositional process: the synergy, that is, between various acoustic resources through a “common area” where different instruments could merge into one another and where the electronics is both in deferred and in real time, is used to increase the ambiguity between the real provenance of various acoustic materials.

At the same time, it is for this very reason that the group lends itself to the challenge of a composition lesson: finding common ground by extending conventional techniques, respecting the physiology of each instrument in action. I was therefore led to believe that in tune whistling could be an element to build a basis for this compositional process. On the one hand, tuned whistling is an extended technique specific to voice; but on the other hand, tuned whistling oriented towards the *embouchure* of the sub-bass recorder enables hearing both the whistling in itself and a low intensity sound, coming from the action on the recorder’s body.

Last but not least, the sounds obtained by appropriately covering a part of the recorder's labium share some spectral characteristics in common with the tuned whistling. In this case, the function of live electronics is to re-establish balance for both sources trying to fill the gap left by their physical distance.

Using the microphone associated to the voice enables raising up to a level of audibility out of tune sounds produced by the mouth in a number of ways. Such ways of using the voice starting from the fifties are very well-known since the fifties; in my specific case, the

use of extended techniques only makes sense in combination with acoustic amplification given that only in this case does mimesis between voice and instrument become clear, making the motivation for such sounds relevant.

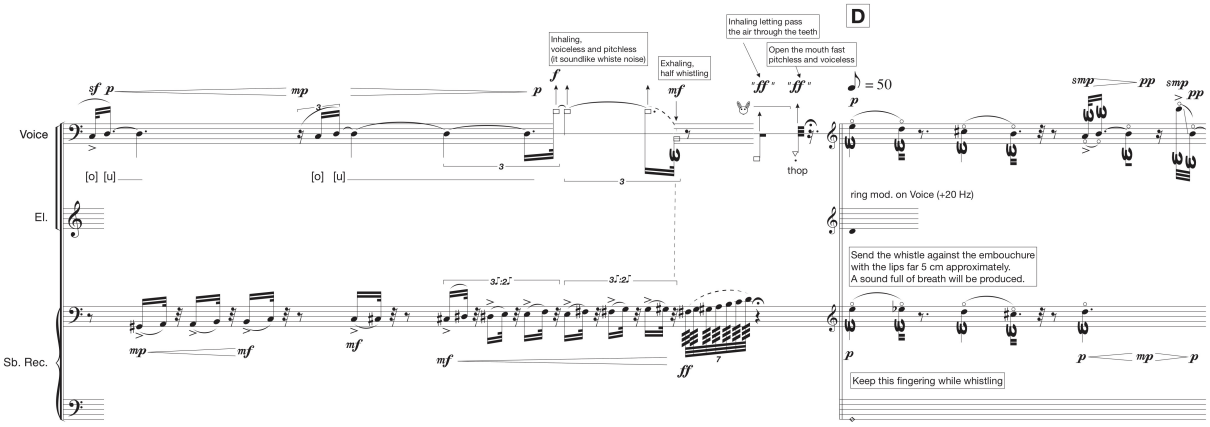


Fig. 8.3 excerpt from the score for Tutorial 1: #mimesi, p. 3

MMixte architecture

In this piece, the few events occurring in the electronics are managed directly by the voice by means of a midi keyboard. The moments when the events should be launched are written in the score as notes corresponding to the midi pitch numbers the event launch is associated to. Fig. 8.4 illustrates this piece's *MMixte* architecture; fig. 8.5 shows the Max patcher with all *MMixte* modules in presentation mode:

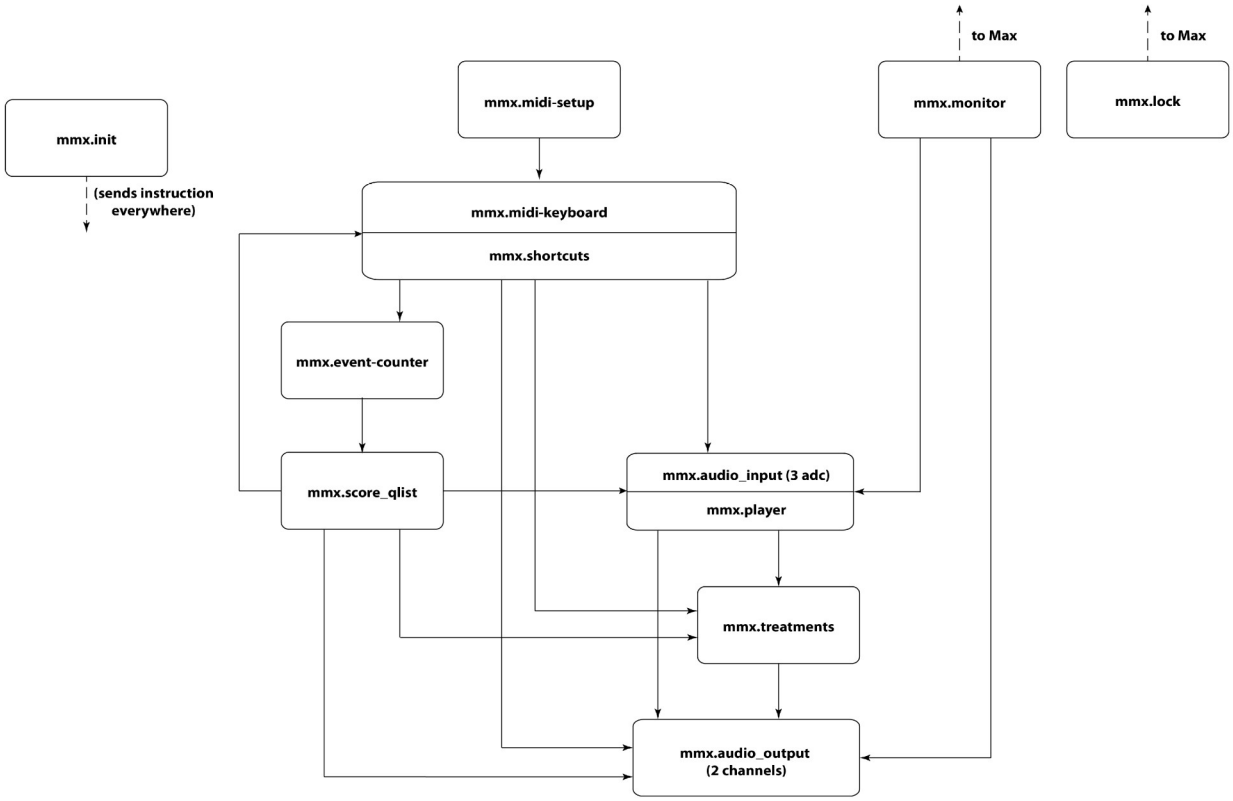


Fig. 8.4: MMixte architecture for *Tutorial 1: #mimesi*

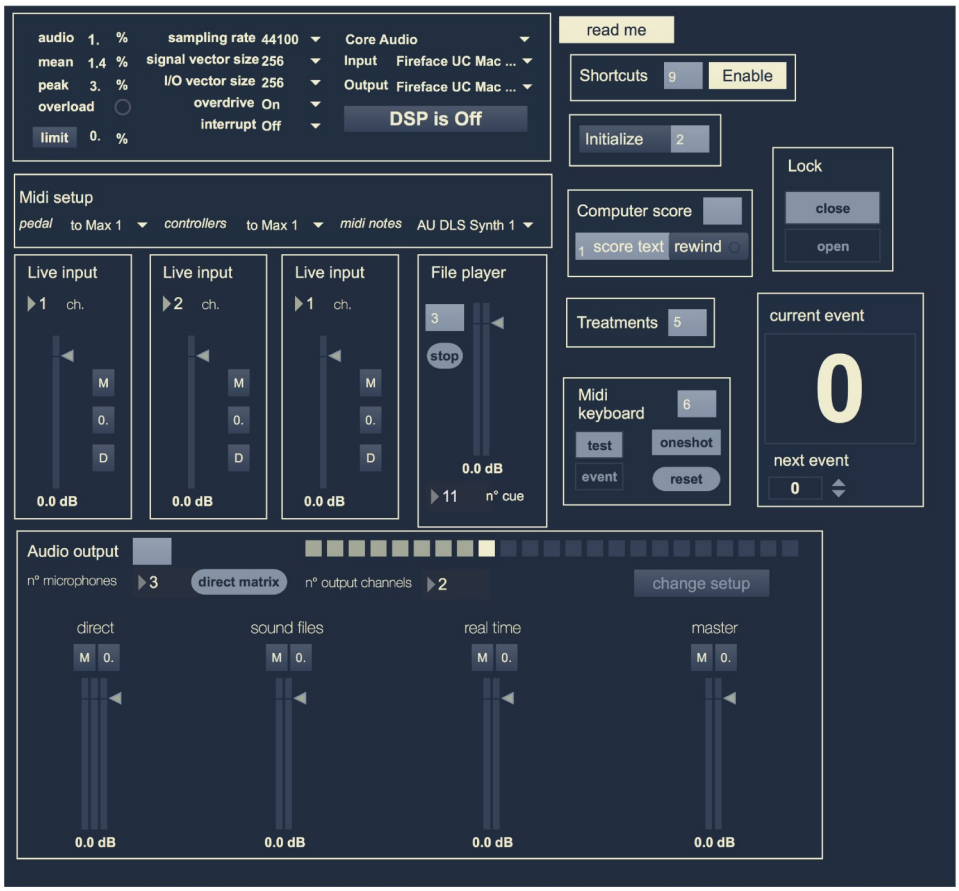


Fig. 8.5: MMixte modules in presentation mode for *Tutorial 1: #mimesi*

9. Meccanica della solitudine (2018)

9.1 Background for the work

This is the third of the pieces making up the *Meccaniche (Mechanics)* cycle. It draws inspiration from a number of situations where the condition of social distancing may be encountered. The words “solitude” and “loneliness” pinpoint, in the English language at least, two different attitudes towards what seems to be the same condition; while the first case is about a mere social circumstance the latter refers to a pathological condition. Reality shows that there are many conditions where such distancing from other individuals does occur due to roles in professional and familiar relations. The titles for the different movements in this piece, though not necessarily linked to each other, reflect the strategy through which the relationship between soloist (barytone saxophone), co-soloist (on percussion) and other members of the ensemble is articulated:

- *To be alone;*
- *To be a leader;*
- *To be invisible;*
- *To be different;*
- *To be against;*
- *To be chased;*
- *To be blind.*

Each section is based on a choice of different performance techniques and differentiated, competitive strategies ensuring cohesion in the relationship between the suite's macro-form and the musical material the sub-sections are made up of. In many cases, the percussion co-soloist is called upon to switch instruments at the beginning of each new movement. *Meccanica della solitudine*'s world premiere is scheduled for November 16th 2018 at Espace K in Strasbourg. Commissioned by the Ensemble Proxima Centauri, it is

co-produced with Strasbourg's HANATSU Miroir Ensemble thanks to financial support provided by the Ministry of Culture and Communication's program “Aides à l’Ecriture de nouvelle oeuvres originales (formerly "Commandes d’Etat”).

9.2 Instrumental set-up

- Solo barytone saxophone (with an AKG C516 ML microphone)
- Percussion co-soloist (Hurgy Toy, Polystyrene, Marimba, one 20-inch suspended “crash” cymbal, China chimes, also using a small mechanical bow);
- C bass flute;
- Contrabass flute;
- Bb Contrabass Clarinet;
- Grand Piano;
- Percussions (Vibraphone, Two 16” and 24” crash cymbals; Bass Drum, Thunder Machine. Also uses a vibrating razor);
- Sampler (through a midi keyboard);
- 1 monitor for the barytone saxophone;
- Two stereo speakers for the sampler, to be positioned close to the keyboard on stage to ensure audio file “local” diffusion;
- Two stereo speakers to be positioned by the sides of the stage for amplification and real time diffusion;
- 7 DMX lights in addition to the usual lighting stage set-up.

9.3 New instruments

Over the years, I have been working with luthiers specialized in electro-mechanical instruments and the construction of musical instrument prototypes. Collaborative work with Alsatian luthier Léo Maurel has yielded practical results for some of the pieces described here.

9.3.1 Hurgy toy

The Hurgy toy is a Chinese toy inspired electro-mechanical instrument. Initially designed for children, it has found a place in experimental compositions, in improvised music and in other contexts of contemporary creation.

It has three strings rubbed by a wooden wheel continually moving thanks to a battery-powered mechanical device. The strings touch the wheel whose motion enables sound emission whenever the strings reach a sufficiently high degree of tension at the pegs, at which point strings affect both intonation and effective sound production. Given its instability, intonation is set in relation to the context; fingers can just glide over the strings, thus working like a nut. A switch activates/deactivates the mechanism making the wheel spin. Although the basic prototype may be changed (by adding a keyboard below the strings, for instance), sound and performance techniques remain the same. The instrument may be connected to an audio card via an audio cable with a 1/4 inch TS jack.

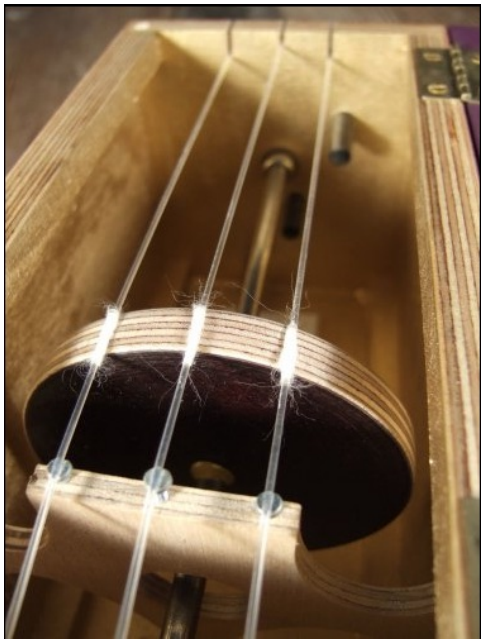


Fig. 9.1 Hurgy toy

Fig. 9.2 Hurgy toy

The "Boîte à bourdon" (drone box) and the "Babasse" are variations in size of the Hurgy toy; if we assume the Hurgy toy to be a soprano instrument, then these are its tenor and bass counterparts. Beyond register variations, sound emission, performance modes and the possibility of connecting the instrument to an electro-acoustic chain remain unchanged. In Fig. 9.3 I show how the Hurgy Toy is notated in the score:

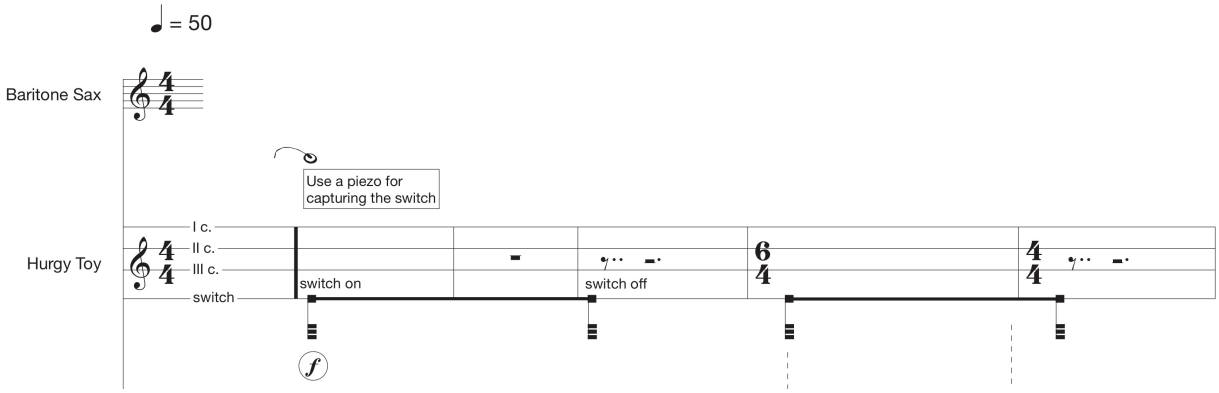


Fig. 9.3 Excerpt from the score for *Meccanica della solitudine* n° 1

9.3.2 Continuous bow

This is an electro-mechanical device made up of two steel wheels of unequal size; the larger wheel is connected to an external engine setting it in motion. A belt made of leather, or any other high friction coefficient material, is set between the two wheels; the use of liquid pitch helps in increasing the friction of the belt when leaning against a surface. A midi volume pedal ensures proper management of the speed of the main wheel's movement. This “bow” lends itself to use on string instruments but also on such percussion instruments as marimba, vibraphone, and plates.

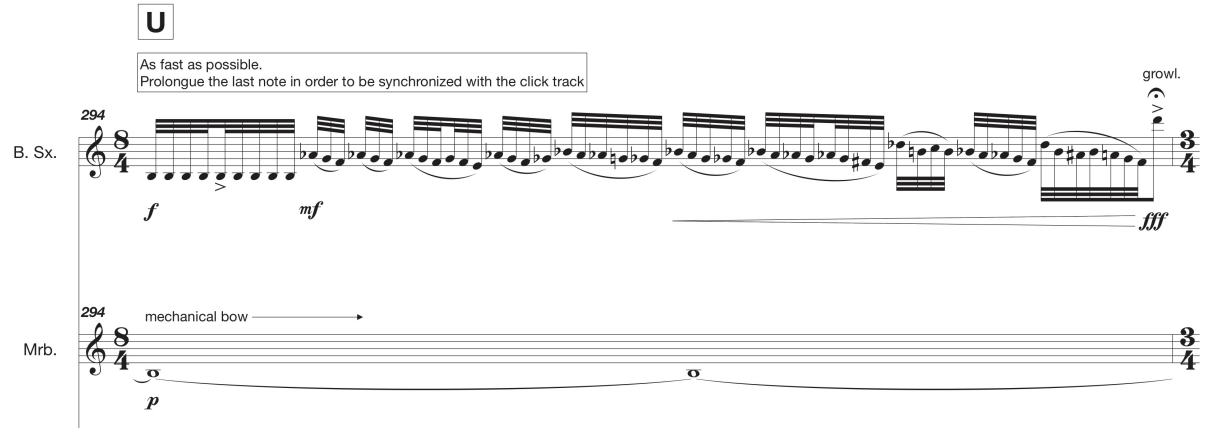


Fig. 9.4 Excerpt from the score for *Meccanica della solitudine* n° 2

Application on percussion turns out to be the most interesting one as it enables long sounds without physical limits in terms of duration, with a timbre much like those of double bass bows. Overcoming time limits of a continuous sound due to bow length is a highly relevant aspect from a compositional standpoint.

The continuous bow device features three components connected through cables:

- bow (one hand is enough to handle it);
- engine box;
- midi “volume” pedal.

The rotation speed of the bow's wheels can be set through volume pedal inclination; the engine receives the information from the pedal and sends it to the bow.

Fig. 9.5 Continuous bow and engine

9.4 DMX protocol and MMixte application

For some time now, I have been trying to integrate lighting design to music in many of my pieces. The first one was *Corpo d’aria*, for bass flute, shadows and live electronics: a flutist is hidden behind a series of translucid panels and a light set-up creates a Chinese shadow play accompanying the whole performance. Such indications concerning the absence of lighting on stage as the main fade-ins and-fade outs are notated in the score so that technicians in charge of lighting can easily adjust on this basis for every production.

The integration of lighting in the score seeks to counter any and all chances that someone other than the composer might express meta-meanings, as it were, other than those originally intended by the work. Conversely, excessively meticulous notation for lighting would be of no use given the extreme variability of existing set-ups which, indeed, depend on types of stage, size of stage and the technical possibilities afforded by the site

hosting performances. Each representation therefore yields different results from those of other productions: this necessarily imposes variable technical solutions ensuring the best possible result each and every time.

As fig. 9.6 shows, in *Meccanica della solitudine*, aside from the indications concerning the absence of lighting on stage, fade-ins and fade-outs, a few simple rhythms are notated to enable technicians in charge of lighting design to correctly interpret the lighting indications by means of a number of available solutions. Such synchronization is performed directly from the stage through midi messages sent from the sampler to a software program for DMX Protocol management.

In the *MMixte* conception, I questioned for a long time whether or not to integrate the DMX protocol within the project package. The problem I was faced with was to limit the package to the Max base library only or let external objects be integrated for the module collection to work. In the end, in order to live up to my initial claims including, in this case, the willingness not to have the survival of *MMixte* depend on programming upgrades by external sources every time a new Max version is released (which does happen rather frequently, in the market, to many external objects), I decided to limit the collection to Midi protocol. Besides, *MMixte* takes on programming tasks of a midi controller enabling the visualization of its graphic interface; the midi message destination which, in this case, is for LED lights management, is third compared to the collection and needs no visualizing while the piece is being performed.

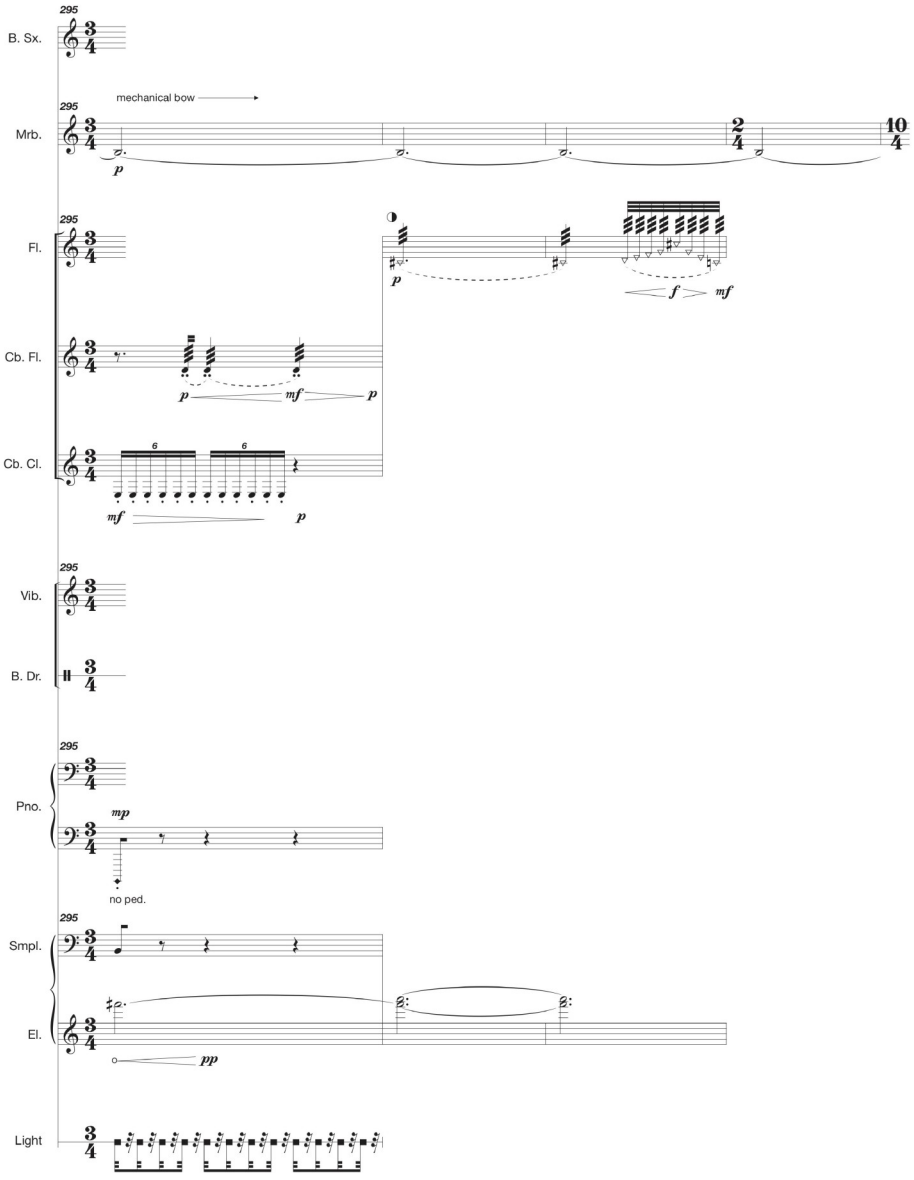


Fig. 9.6 Excerpt from *Meccanica della solitudine* n. 3

The diagram for the *MMixte* modules in this piece is therefore the following:

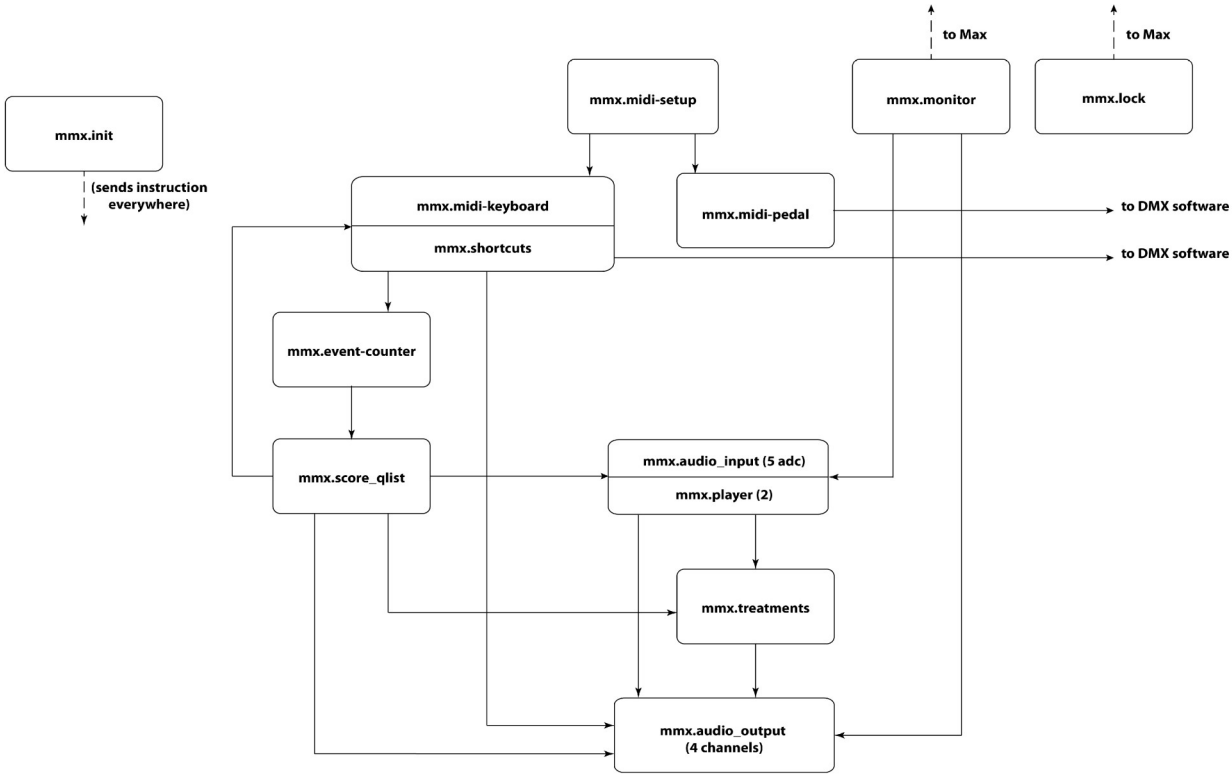


Fig. 9.7 Diagram of *MMixte* for *Meccanica della solitudine*

Here the midi keyboard launches a series of audio files connected to each programmed key. Some of these live electronics actions, occurring with the launching of audio files, are associated to the musician's action on stage. Therefore, some of the events generated by the electronics are launched from the midi keyboard sending, if prescribed, a signal to the event counter to trigger its forward movement.

The rest of the instructions for the live electronics is launched from the mixing board through the computer keyboard.

The midi pedal for the co-soloist enables sending instructions to the DMX software program external to Max.

The software program's interface, in this case, looks like this:

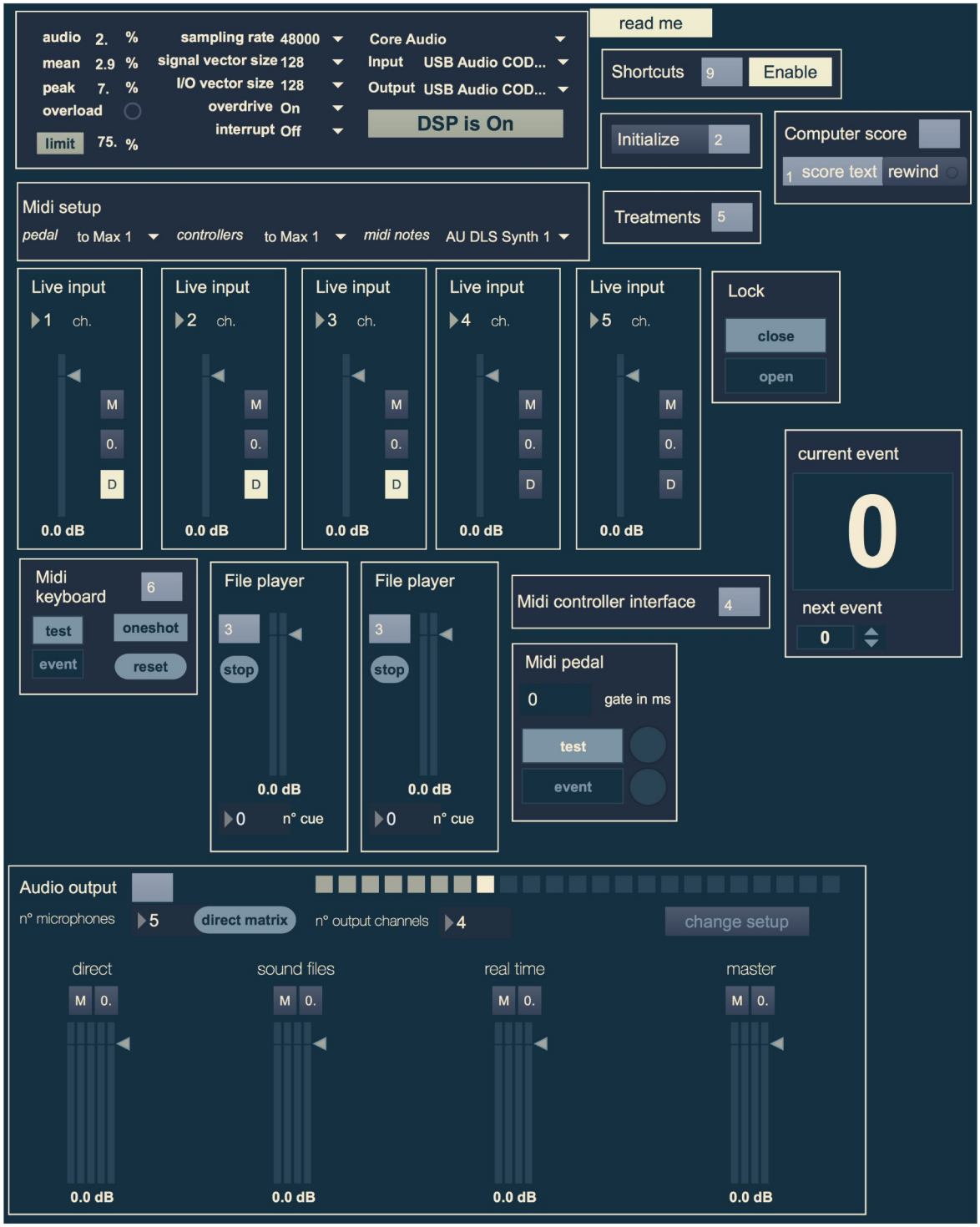


Fig. 9.8 *MMixte* modules for *Meccanica della solitudine*

Conclusion

MMixte is a middleware offering an open and versatile solution to programming software in Max to manage live electronics in addition to diffusion of audio files series in concert, with controls on the DSP and controllers eventually connected in a single window. It offers hosting space for one's own audio processing algorithms, spatialization models and a rational approach to audio threading to be managed without giving up programming habits. The use of *MMixte* e can optimize programming time for new mixed music works.

The pedagogic issues involved in the trasmission of knowledge needed to program a computer environment suitable for mixed music are hardly secondary. Content explanation for each *MMixte* module can be the focus for lessons in a live electronics course in Conservatories and University Arts Departments. At the Conservatory I teach in, the very content of this dissertation unfolds in a 30-hour, third year course in performance environments and control for live electronics for the Bachelor in Electronic Music program.

The project's modular nature, with a basic graphic interface kept simple and a pre-constituted syntax for the overall functioning of the modules themselves, is also geared towards non-professional programmers already familiar with live electronics modular programming. Users working not exclusively with audio (especially those working in theatre, interactive art installations, and so on) arguably add up to a market segment: indeed, Cycling has been steadily introducing the easily accessible modular collections *Beap* for audio and *Vizzle* for video within the software program. More programming companies have been increasingly launching middleware solutions, at no cost or low fees, enabling pre-constituted elements to be combined according to the needs of each project.

This dissertation seeks to demonstrate the extent to which the structure of a patcher concert and its architecture is the outcome of habits - thirty years, and possibly more, of age - in mixed music production. At the same time, it seems to me that the market is still lacking a rational explanation for what a patcher concert is, let alone the theory accounting for necessary, or even compulsory, ways of turning such a patcher into an effective software architecture for live electronics.

In the future, I believe we will have to face the problem of audio processing module sub-architecture, deliberately left aside in this project since digital audio processing is not the main focus under scrutiny here. Depending on adopted techniques and technologies, a number of architecture solutions can be implemented; but while this issue was already addressed in other computer science areas, pipelined architecture remains, to this day, the most widely adopted model in the world of audio processing.

Interaction and robotics are becoming ever so present in every day life - and we may presume they are likely to become all-pervasive in musical performance as well, sooner or later - what will surely force us to re-appraise the paradigms of software architecture enabling robots and delocalized user-networks to take part in programming, in the functioning and use of software programs to be created.

MMixte, while still an architecture solution for the rational management of audio signal, is likely to evolve depending on the user-base, the software programs to be developed and Max's very own evolution, which it will continue hosting for the time being.

The choice of building all modules only with the Max base library prevents, on the one hand, the need to upgrade the collection with the next software releases while, on the other hand, setting limits on some of the functions. According with users feedback, I will decide how to turn the project in the next future.

Live Ableton's purchase of Cycling ’74, owner of Max's distribution licence, will no doubt have considerable impact on users/programmers' habits and the software program's orientation in the years to come. In all likelihood, the *MMixte* version for Max for Live (a Max extension for use within Live providing one of the clues that the Max and Live bond is only going to get tighter) will be this project's next step.

For Max users, the shift from 32-bit to 64-bit architecture within Apple operative systems was a major turning point. Entire object collections can no longer be used and some of these haven't been updated in years simply because the projects they are the outcome of have now reached completion: the new OS has made them obsolete. Awareness that obsolescence is one of the worst enemies of computer science is at the heart of the choice in programming *MMixte* for its current version. Years will go by but there will always be a need for live electronics software programming and the demands for architecture will basically remain unchanged for a long time until new artistic paradigms emerge; the real challenge of innovation lies within audio elaboration and it will always find room in *MMixte* architecture for its own functioning; it will find a functional place, that is, within structures of audio signal behaviour and in live performance.

Depending on the choices Cycling ’74 will be making now that it has been purchased, communities will reach an understanding of what the new programming language behind Max software will be and, with this, which programming languages it will be able to communicate with.

One must not rule out the possibility that Pure Data[87] will find space to improve the software program's graphic interface in some computer company, thereby turning it into Max's true competitor. Pure Data's background architecture is superior to Max's: when Miller Puckette designed it in the Nineties, his experience creating Max has been crucial in conceiving this software. Despite of this experience, the absence of a company's economic investment slows down the implementation of further improvements. The software program's graphic interface is at the moment practically inexistent, and this feeble point makes its own market segment smaller than Cycling ’74's software program. Cycling ’74's choices made for Max in the last few years have been set towards increasing as much as possible the number of users, embracing everyone from beginners to advanced. Nowadays, however, the market goes through sudden changes and today's commercial balance can easily be questioned in very little time once new forces are called upon to intervene on the market.

The theoretical argument in this dissertation can be applied regardless of the software programs chosen for future use since theoretical abstraction from the model of software architecture needed for mixed music is likely to be found in every new environment where it will definitely materialize.

[87] Pure Data is a software designed by Miller Puckette. https://puredata.info/

Appendix 1 - Vibrating mallet

In the previous chapters I shown how I have made extensive use of vibrating razors in many of my pieces since 2013. Empirically, at first - as it often happens when exploring unconventional performance techniques. Then, I began compiling possible sounds from previously used instruments and moved on to audio analyses of these recorded files.
This phase of scientific understanding of the use of vibrating bodies on musical instruments was followed by a long phase designing a tool with the potential of becoming a dedicated instrument, not just any simple tool borrowed from everyday life for a musical performance.
This project's goal was to come up with a tool similar, in shape and length, to any other percussion mallet, one just as easy to handle, and with similar weight distribution, leading me back to the context of musical tools proper without stepping into the unconventional use of common, everyday life objects. Interest in unconventional objects was never justified in terms of aesthetic reasons: it was always a search for sound as such. The conception of a prototype of a vibrating mallet was therefore needed to avoid that visual connotations of these instrumental actions might take on more meaning than the sound produced.
Three different types of mallets were designed before achieving concrete results.
The first prototype of a designed vibrating mallet included a rotating engine in an aluminum, hollow cylinder-shaped body whose speed depended on a slider retrieved from a crack inside the body:

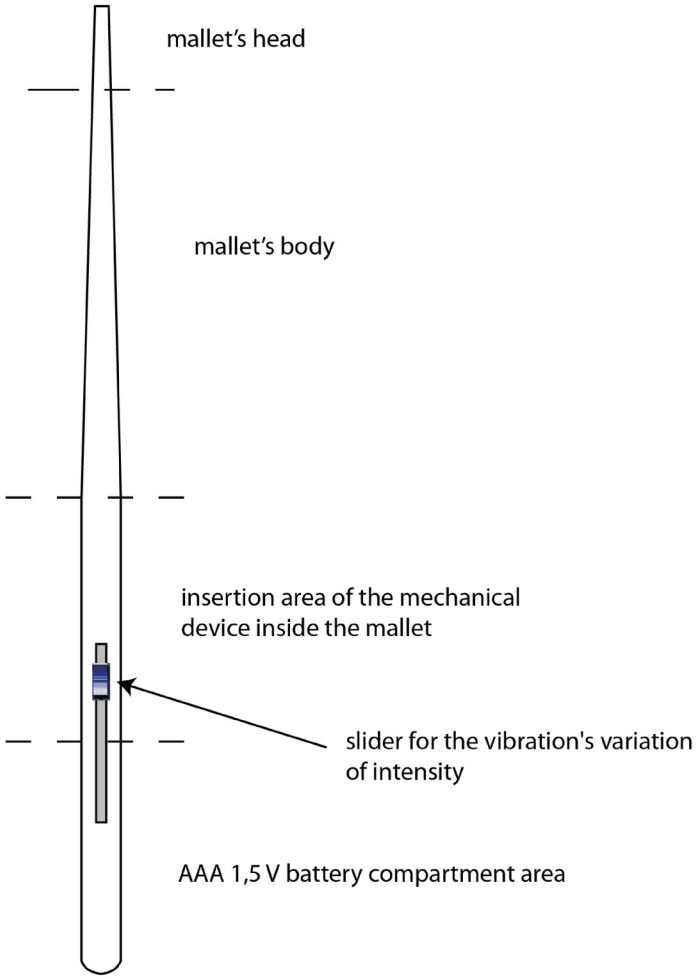


Fig. A.1: Prototype of vibrating mallet n. 1

A battery inserted at the base of the cylinder-shaped body ensured power and counterweight to the stick's head:

Fig. A.2: Computer simulation of the battery compartment

At the beginning, what I had in mind was constructing moveable heads to be screwed to the body ending with a thread. Dedicated shape heads made with different materials would have been built to get different sound results on the contact surfaces.
Completion fase of the sticks' heads, however, was never reached for this model: a number of problems was left unsolved in the completion of the vibrating body.
Even though the idea of a slider on the stick's body intuitively seemed to be the best one from a practical point of view, the following problems arose:

- where energy was transmitted to the engine, the variator and the (plastic) slider coming out of the slit on the copper body were also vibrating, causing parasite noise becoming perfectly audible at full speed;
- the mallet's diameter was much larger than any other conventional percussion stick. Holding it was safe and comfortable, except that handling the slider would force the performer to always keep his fist half-open, instead of closing it;
- power generated by the two AAA batteries was not enough to let the skin of as large an instrument as the Timpani drum or the Bass Drum resonate;
- the lack of thumb precision in the shift of the slider caused the stick's head to lose firm touch with the contact surface when changing cursor position.

Marco Bosisio of Tesla srl in Monza (Italy) was successful in elaborating a second prototype, thus solving this particular series of problems.
A new type of engine was implemented to get the most out of battery power. An internal switchable resistor was placed close to the battery compartment to change the frequency of the produced vibration.
The mallet's head was designed like that of a snare drum, slightly larger, and easy to attach to the body through a thread.
A system of weights firmly placed along the cylinder-shaped body absorbs vibrations coming from the engine and ensures stable handling.
The stick's diameter was still larger compared to a standard stick and depended on the size of the battery compartment. The user's sensation was that of holding a "tool" more than a "mallet" but this model's ease of handling was significantly improved compared to the previous one.

Fig. A.3: Prototype of a vibrating mallet n° 2. From left to right: the lower threaded cork, the engine with internal switchable resistor, the stick's body and a threaded head

The pitch resulting by leaning the mallet against a surface depends on the battery's voltage. A resistor is therefore necessary to play more pitches. Actually, demands for a resistor depend on the sound result we wish to hear when the stick is being used; it is for this reason that an attempt was made to implement an internal resistor with four switches set in three positions each (low, medium, high) so the stick could vibrate at twelve different speeds:

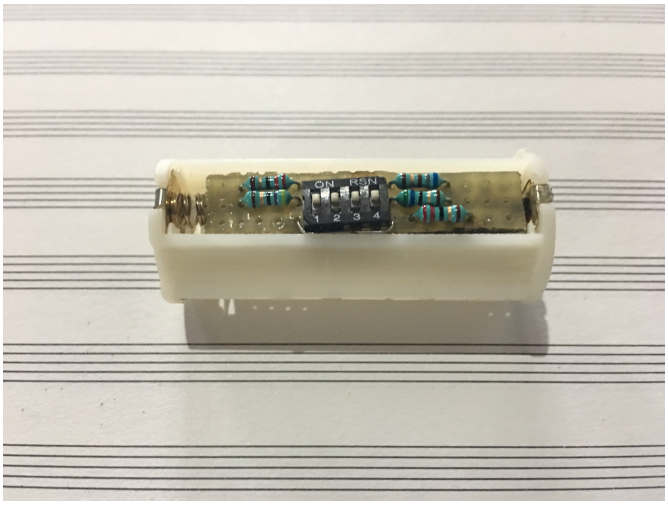
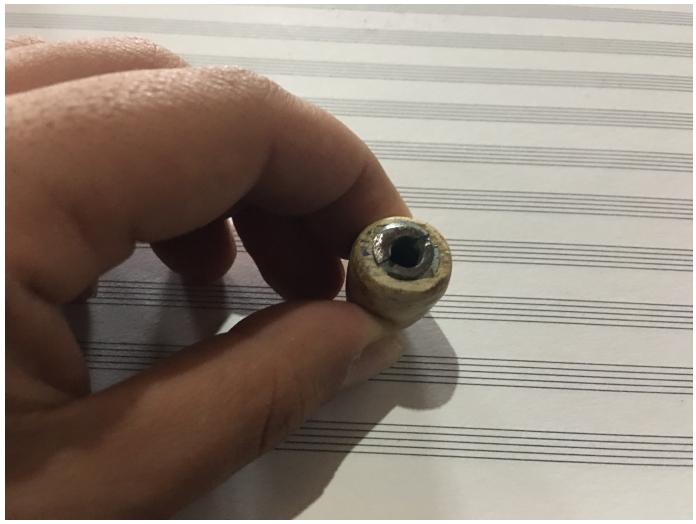


Fig. A.4: The internal engine. The component at the center is the resistor with four switches in combination and a total of twelve positions

The shape of the stick's heads can follow both an imitation of the traditional sticks' heads and their artisanal modification, by which we mean their separation from the body of the original stick and the insertion of “M6” threads. This way, all the sticks' heads large enough to be able to host this kind of thread can be adapted to be used with the vibrating



mallet. Here below, two artisanally built wooden heads:

Fig. A.5: M6 thread on a head similar to that of a Snare drum stick.

Fig. A.6: M6 thread on a head similar to that of a Timpani drum stick.

It wasn't possible to push any further the power generated by the engine since the body and its mechanical stimuli would have ended up shifting the components inside, breaking

them or disconnecting them from their enclosure. The problem of handling was yet to be solved; so was the possibility of varying the frequency of the vibration.

The third and last vibrating stick model includes a modified midi “volume” pedal connected via a TRS 1/4” jack to a stick whose diameter and length are similar to those of a vibraphone stick:

Fig. A.7: Third prototype of vibrating stick, with an external resistor inserted in a modified midi “volume” pedal.

The stick's diameter is, in this case, the one needed to host the jack's input connection. This model will be used in my instrumental music works starting from 2019. It represents the first attempt at ”silent” electronics: the use of computer and live electronics does not produce digital sounds; it only works according to data gained from real time performance (which the engine's rotation speed may be connected to according to pre-set coefficients) with tools working within acoustic and instrumental performances. This is a new paradigm of mixed music I wish to develop in my next compositions.

Appendix 2 - Public presentations and articles

MMixte was presented in public:

- on June 20th 2017 at the University of Genova, computer engineering department (coordinated by Prof. Antonio Camurri);

- on October 25th 2017 for the Korea Electro-Acoustic Music Society's Annual Conference (KEAMSAC) 2017 at the Asia Culture Center (ACC) with Miller Puckette;

- on March 8th, 2018 at the IRCAM Forum;

- on July 3rd, 2018 at the University of San Martin in Buenos Aires, Mauricio Kagel Arts Institute.

Two articles about *MMixte* have been published so far:

- "*Lost in feedback* (2014): soluzioni personali di musica mista tra il teatro musicale e la performance visuale". *Musica/Tecnologia*, 11/12, Florence University Press, 2018, pp. 29-44.

- "*MMixte*: a new Max package for live electronics with acoustic instruments". Proceedings of Korean Electro-Acoustic Music Society's 2017 Annual Conference (KEAMSAC2017), Seoul, Korea, October 24th-25th, 2017.

Appendix 3 - Scores

Between 2013 - 2018, in the context of my PhD program and with the opportunity to work under my tutor Prof. Erik Oña's direction, I have written twenty-four works for orchestra, chamber music, and musical theatre; one installation and one opera for cartoon videos. Most of these works include electronics:

	Meccanica della solitudine
Year	2018
Instrumentation	Baritone Saxophone soloist, Percussionist co-soloist, Flute, Contrabass Flute, Clarinet, Piano, Percussion, Sampler, live electronics and stage set-up
Electronics	Yes
Duration	20’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	November 16th, 2018, Strasbourg, Espace K
other	Commission of the Ensemble Proxima Centauri; piece awarded by the program of the French Ministry of Culture and Communication “Aides à l’écriture de nouvelles œuvres originales - ex Commandes d’Etat -“

	Pulse
Year	2018
Instrumentation	Alto sax and Sampo
Electronics	Yes
Duration	10’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	June 26th, 2018, Bourges, Festival Art & Science
other	Commission of the Musinfo Association

	Speedcore aptitude
Year	2017
Instrumentation	Flute, Alto sax, Synthesizer, live electronics
Electronics	Yes

Speedcore aptitude

Duration	8’20”
Publisher	Edizioni Suvini Zerboni
World premier	December 8th, 2017, Köln, Alte Feuerwerke
other	Commission of the Ensemble Inverspace
Audio / Video	http://www.mauriliocacciatore.com/the-ways-you-cry.html

My track for FM

Year	2017
Instrumentation	Tape
Electronics	Yes
Duration	10’20”
Publisher	Edizioni Suvini Zerboni
World premier	December 4th, 2017, Bari, Festival UrtiCanti
other	Tape for Jazz improviser
Audio / Video	CD soon released

Breve

Year	2017
Instrumentation	Violin and Piano
Electronics	Yes
Duration	3’30’’
Publisher	Edizioni Suvini Zerboni
World premier	June 11th, 2017, Milan, Theatre Elfo Puccini
other	Commission of the Ensemble Sentieri Selvaggi
Audio / Video	http://www.mauriliocacciatore.com/the-ways-you-cry.html

The ways you cry

Year	2017
Instrumentation	String quartet
Electronics	Yes

The ways you cry

Duration	13’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	March 16th, 2017, Bruxelles, Klarafestival
other	Commission of the Mittelfest (Cividale del Friuli, Italy), in collaboration with MUSMA - Music Masters on Air -
Audio / Video	http://www.mauriliocacciatore.com/the-ways-you-cry.html

So loud

Year	2014 - 2017
Instrumentation	Bass Saxophone and Piano
Electronics	Live electronics
Duration	20’
Publisher	Edizioni Suvini Zerboni
World premier	March 5th, 2017, Karlsruhe, ZKM
other	Piece produced by the ZKM in collaboration with IEMA - International Ensemble Modern Academy -

My child is dreaming

Year	2016
Instrumentation	Tape for Video
Electronics	Yes
Duration	2’30”
Publisher	Edizioni Suvini Zerboni
World premier	17° Festival del Cinema Europeo (Lecce, Italy, 2016)
other	Piece produced by the ZKM in collaboration with IEMA - International Ensemble Modern Academy -
Audio / Video	https://vimeo.com/195441055

Meccanica degli avatar

Year	2016
Instrumentation	Fl (C, Sliding Flute), Cl., 2 Vln., Vla., Vc., Acc., Perc.

Meccanica degli avatar

Electronics	Yes
Duration	18’30’’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	December 21st, 2016, Florence, Sala Vanni
other	Vittorio Ceccanti, conductor. Commission of the Contempoartensemble
Audio / Video	https://www.youtube.com/watch?v=7QlV4dMMylo&t=8s

IV studio

Year	2016
Instrumentation	Piano
Electronics	No
Duration	4’30’’
Publisher	Edizioni Suvini Zerboni
World premier	September 20th, 2016, Strasbourg, Institut de Culture italienne
Audio / Video	http://www.mauriliocacciatore.com/studies-for-piano.html

La Vallée des Merveilles

Year	2016
Instrumentation	Fl. (piccolo, C, Bass), Cl. (Bb, Bass, Cb.), Percussions
Electronics	Live electronics
Duration	60’00’’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	June 18th, 2016, Strasbourg, Espace K
other	Commission of the ensemble Hanatsu miroir, supported by the Fondation Salabert
Audio / Video	https://drive.google.com/open?id=0Bxn75GrY9HKQbTZRU0R2THQwTGs

Clic clac on pattern

Year	2016
Instrumentation	Flute and Bass clarinet

Clic clac on pattern

Electronics	Live electronics
Duration	10’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	April 21st, 2016, Strasbourg, Espace K
other	Commission of the ensemble Hanatsu miroir

I don’t need to …k for music

Year	2016
Instrumentation	2 electric guitars
Electronics	Live electronics
Duration	8’30” ca.
Publisher	Edizioni Suvini Zerboni
World premier	March 18th, 2016, Frankfurt, Festival Luminale 2016, Nebbiensches Gartenhaus
other	
Score	https://drive.google.com/open?id=0Bxn75GrY9HKQZ25RdTFRSUxXNkk
Audio / Video	http://www.mauriliocacciatore.com/i-dont-need-to-k-for-music.html

Three studies about the weight of drops

Year	2016
Instrumentation	Fl., Vln., Vc., Pf.
Electronics	No
Duration	7’30” ca.
Publisher	Edizioni Suvini Zerboni
World premier	January 10th, 2016, Paris, Théâtre de l’Aquarium. Ensemble Aleph
other	Commission of the Ensemble Aleph
Score	https://drive.google.com/open?id=0Bxn75GrY9HKQS2pVRlFCZENtZm8
Audio / Video	http://www.mauriliocacciatore.com/three-studies-about-the-weight-of-drops.html

Space / Time

Year	2015
Electronics	Yes
Set-up	64 loudspeakers
Duration	27’49"
Publisher	Edizioni Suvini Zerboni
World premier	September 10th-13th, 2015, Basel, Festival ZeitRäume, City Parkhaus, within the Klanginstallation “Tunnel Spiral”
other	production Elektronisches Studio Basel

Tralci

Subtitle	for 14 players and live electronics
Year	2015
Instrumentation	Fl. Cl./Cl.b. Gf. Sax A./Bar. Trb. Trbn. Tb. Perc. Pno. El.Guit. Vln. Vla. Vc. Cb.
Electronics	Yes
Set-up	6 channels on stage (4 + 2)
Duration	12’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	June 27th, 2015, Basel, Musik Akademie, Grosser Saal
other	production Elektronisches Studio Basel

Vit_Vite_Evit

Subtitle	for 6 players
Year	2015
Instrumentation	Fl. Cl./Cl.b Perc. Pno. Vln. Vc.
Electronics	No
Duration	14’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	May 6th, 2015, Milan, Teatro Elfo Puccini
other	Commision of the Ensemble Sentieri Selvaggi

III studio - Suoni saturi

Subtitle	III study for Piano
Year	2015
Instrumentation	Pno.
Electronics	No.
Duration	7’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	April 29th, 2015, Tashkent, Music Conservatory

IV anfibio/b

Subtitle	for amplified Flute and live electronics
Year	2014
Instrumentation	C Flute
Electronics	Yes
Set-up	1 mic on stand, 1 head-set, 4 channels on stage (2 front-stage, 2 end-stage)
Duration	13’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	November 11th, 2014, Seoul, Seoul Art Center, Yaju Theatre. Seoul International Computer Music Festival. Byung Chul Oh, Flute.

Frammenti senza cornice

Subtitle	for three instrumental groups and live electronics
Year	2014
Instrumentation	Fl. Cl. Ob. Fg. Cn. Trb. Perc. Vln. Vla. Vc. Cb.
Electronics	Yes
Set-up	6 channels on stage (4 + 2)
Duration	16’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	June 27th, 2015, Basel, Musik Akademie, Grosser Saal
other	Commission of the Kammerorchester Basel

Radio Jail

Instrumentation	Alto Sax., Perc. Pno.
Year	2014
Electronics	Yes
Set-up	1 channel on stage
Duration	5’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	March 30th, 2015, Athenes, French Institute of Greece
other	Commission of the Artéfact Ensemble, Athens. Piece included in the CD “Thive Km 102”. Puzzlemusic recordings, 2015.

Lost in feedback

Subtitle	for electronic vibraphone, percussions, 1 stage performer and live electronics
Year	2014
Instrumentation	1 percussionist and 1 stage performer
Electronics	Yes
Set-up	3.2 channel (2 + 1)
Duration	30’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	July 5th, 2014, Strasbourg, Hall de Chars
other	Commission of the Ensemble HANATSU miroir. Piece included in the DVD “Lost in feedback”, M.A.P. Editions, 2015.

Solo in eco

Subtitle	for solo Bass Clarinet and four instruments
Year	2009-2014
Instrumentation	B.Cl. solo - Perc., Vla. Vc. Cb.
Electronics	Np
Duration	10’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	May 27th, 2014, Paris, Théâtre Dunois, Ensemble Aleph. Dominique Clément, Clarinette basse.
other	piece selected for the VII Forum of the young creation of the Ensemble Aleph

Radio racconti appena accennati

Subtitle	for orchestra and electronics
Year	2013
Instrumentation	2.2.2.2. 2.2.2.0. Tp. Perc. Ar. Pno. 12.10.8.6.4.
Electronics	Yes
Set-up	1 channel on stage
Duration	10’ ca.
Publisher	Edizioni Suvini Zerboni
World premier	Recorded on the 20 November 2013 in Paris for the Radio cycle of France Musique “Alla breve”. Orchestre Philharmonique de Radio France. Pierre-André Valade, conductor
other	Commission of the Radio France, Orchestra

Appendix 4 - USB stick contents

A USB stick is included at the end of this dissertation containing the pdf file for the text of the dissertation, Curriculum Vitae, pdf scores for the pieces I have written all through this PhD program and their audio/video recordings.

Bibliography

Allen, Andrew Stuart. *Ruratae: A physics-based audio engine.* A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy, University of S. Diego, California, 2014

Bachmann, Felix; Bass, Len; Clements, Paul C.; Garlan, David; Ivers, James; Little, Reed; Merson, Paulo; Nord, Robert; Stafford, Judith A. *Documenting Software Architectures: Views and Beyond,* Second Edition. Boston, Addison-Wesley Professional, 2010, ISBN: 0321552687

Bahn, Christy. *Recommended Practice for Architectural Description of Software-Intensive Systems.* ANSI/IEEE Standard 1471-2000.

Bass, Len; Clements, Paul C.; Kazman, Rick; *Software Architecture in Practice.* Second Edition. Boston, Addison-Wesley Professional. ISBN: 0321154959

Bennett, Gerald. *Research at Ircam in 1978.* Rapports IRCAM 1/78, Centre Georges Pompidou, Parigi, 1978.

Bergland, G. D.. *A Guided Tour of Program Design Methodologies.* Washington, IEEE Computer, Vol. 14, No. 10, Oct. 1981, pp 13-37.

Booch, Grady; Rumbaugh, James; Jacobson, Ivar. *Unified Modelling Language User Guide*, Boston, Addison-Wesley 1999.

Cacciatore, Maurilio. *Lost in feedback (2014): soluzioni personali di musica mista tra il teatro musicale e la performance visuale.* Musica/Tecnologia, 11/12, pp. 29-44. Firenze University Press, 2018.

Cacciatore, Maurilio. *MMixte: a new Max package for live electronics with acoustic instruments.* Proceedings of Korean Electro-Acoustic Music Society's 2017 Annual Conference (KEAMSAC2017). Seoul, Korea, 24-25 October 2017.

Camurri, Antonio; Frixione, Marcello; Innocenti, Carlo. *A Cognitive Model and a Knowledge Representation System for Music and Multimedia,* Journal of New Music Research, vol. 23, pp. 317-347. Taylor & Francis Group, Routledge, 1994.

Camurri, Antonio; Innocenti, Carlo; Massucco, Claudio; Zaccaria, Renato. *A software architecture for sound and music processing.* Microprocessing and Microprogramming, Volume 35, Issues 1–5, pp. 625-632. Elsevier, September 1992.

Chikofsky, Elliot J. *Software Development - Computer-aided Software Engineering, Technology Series,* Washington, IEEE Computer Society Press, 1988.

Cipriani, Alessandro. Giri, Maurizio. *Electronic Music and Sound Design Theory and Practice with Max/MSP Volume 1.* Contemponet, Rome, 2010.

Cipriani, Alessandro. Giri, Maurizio. *Electronic Music and Sound Design Theory and Practice with Max/MSP Volume 2.* Contemponet, Rome, 2010.

Coutaz, Joelle. *PAC, an Object Oriented Model for Dialog Design,* pp 431-436. In Rullinger, H. I. and Shackel, R. (eds), *Human-Computer Interaction.* INTERACT '87. Elsevier Science Publishers, 1987.

Dannenberg, Roger; Poepel, Cornelius. *Audio Signals Driven Sound Synthesis.* ICMC 2005 - CDROM Proceedings, 2005

Favreau, E., Fingerhut, M., Koechlin, O., Potacsek, P., Puckette, M., and Rowe, R. *Software Developments for the 4X real-time System*. Proceedings, International Computer Music Conference. San Francisco: International Computer Music Association, pp. 43-46,1986.

Fileding, Roy. *Architectural Styles and the Design of Network-based Software Architectures.*

Dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Phylosophy in Information and Computer Science. University of California, Irvine, 2000.

Freeman, Peter; Wasserman, Anthony I.. *Tutorial on software design techniques*, IEEE Computer Society Press. Washington, 1976.

Garlan, David; Shaw, Mary. An introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering, Volume I, edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey, 1993

Gamma, Eric; Helm, Richard; Johnson, Ralph; Vlissides, John. *Design Patterns: Elements of Reusable Object-Oriented-Software.* Boston, Addison-Wesley, 1995.

Gorton, Ian. *Essential Software Architecture*, pp. 1-43. Springer-Verlag. Berlin, Heidelberg, 2006

Harirforoush, Maryam; Seyyedi, Mirali; Mirzaee, Nooraldeen. *The Evaluation of Reliability Based on the Software Architecture in Neural Networks*, in Software Engineering Advances, 2008. ICSEA 2008.

Hudak, Paul. *The Haskell School of Music - from Signals to Symphonies* - . Yale University Department of Computer Science New Haven, CT, USA Version 2.6, 2014

Kai, Quian. *Interaction-oriented Software Architectures*. *Software Architecture and Design Illuminated, p. 200.* Sudbury, Massachussets, Jones and Bartlett Illuminated., 2009.

Kitchen, Paul. *Architectural Blueprints-The "4+1" View Model of Software Architecture,* IEEE Software, 12(6). Washington, Nov. 1995.

Krasner, Glenn E.; Pope, Stephen T.; *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*, Journal of Object-Oriented Programming, pp. 26-49. Vol. 1 Issue 3, Aug./Sept. 1988.

Ko, Ming-Yung; Shen, Chung-Ching; Bhattacharyya, Shuvra S.. *Memory-constrained Block Processing Optimization for Synthesis of DSP Software.* Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece, July 2006.

Jacobsen, I.; Christerson, M.; Jonsson, P.; Overgaard, G.; *Object Oriented Software Engineering,* pp. 15-199. Boston, Addison-Wesley, 1992.

Lochard, Jean. *Najo Max Interface documentation*, Paris, IRCAM, Forum Ircam, 2018.

Malt, Mikhaïl. *Introduction à la norme Midi.* Essay for the Cursus 1 in Music informatics of the 2009/2010. Paris, IRCAM, 2010.

Manoury, Philippe. *Jupiter* 1987, Mac OS 9 2002. Document written by Serge Lemouton. Paris, Ircam press, 2002.

Montangero, Carlo; Semini, Laura. Architetture software e progettazione di dettaglio. Note per il corso di ingegneria del software, Corso di Laurea in Ingegneria Informatica, Università di Pisa, 2014

Oquendo, Flávio; Leite, Jair; Batista, Thais. *Software architecture in action. Designing and Executing Architectural Models with SysADL Grounded on the OMG SysML Standard,* pp. 1-25. *Switzerland,* Springer International Publishing, 2016.

Perry, Dewayne E.; Wolf, Alexander L. *Foundation for the Study of Software Architecture*. Software engeneering Notes vol. 17 n° 4, pp. 40-53, 1992

Potel, Mike; *MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java.*Taligent Inc., 1996

Puckette, Miller. *Combining Event and Signal Processing in the Max Graphical Programming Environment.* MIT Computer Music Journal 15(3), pp. 68-77. Cambridge, The MIT Press, 1991.

Puckette, Miller. *Etude de cas sur les logiciels pour artistes: Max/MSP et Pure Data.* Reprinted from David-Olivier Lartigaud, Ed., Art++, Editions Hyx, 2009.

Puckette, Miller. *EXPLODE: a user interface for sequencing and score following.* ICMC Proceeding. Michigan Publishing, 1990

Puckette, Miller. *Is there life after Midi?* ICMC Proceedings, pp. 1-2. Michigan Publishing, 1994.

Puckette, Miller. *Max at Seventeen.* Computer Music Journal Winter, Vol. 26 N°4, pp. 31-43. Douglas Keislar Editor, 2002.

Puckette, Miller. *Pure Data: another integrated computer music environment.* Proceedings, International Computer Music Conference. San Francisco: International Computer Music Association, pp. 224-227.

Puckette, Miller. *Proceedings*, Third Intercollege Computer Music Festival, Tokyo, Japan, pp. 1-4. 1997.

Puckette, Miller. *The patcher.* ICMC Proceedings, pp. 420-429. Michigan Publishing, 1988

Puckette, Miller. *The Theory and Technique of Electronic Musi* . World Scientific Press, Singapore, 2007.

Puckette, Miller. *Who owns our software? - a first-person case study.* Proceedings, ISEA, pp. 200-202. 2004.

Roads, Curtis. *The computer music tutorial.* Cambridge, The MIT Press, 1996.

Rolnick, Neil. *A Composer's Notes on the Development and Implementation of Software for a Digital Synthesizer.* Foundations of Computer Music, Edited by Curtis Roads e John Strawn, Cambridge, The MIT Press, 1988.

Solomos, Makis. *De la musique au son. L'émergence du son dans la musique des XXe-XXIe siècles.* Rennes, Presses Universitaires de Rennes, 2013

Tsingos, Nicolas . *A versatile software architecture for virtual audio simulation.* Proceedings of the 2001 International Conference on Auditory Display, Espoo, Finland, July 29-August 1, 2001

Zattra, Laura. *Intervista con Agostino Di Scipio*. 2014

Zhiqing, Xiao; Si, Wen; Heqi, Yu; Zhenyu, Wu; Hao, Chen; Chunhong, Zhang; Yang, Ji. *A new architecture of web applications — The Widget/Server architecture.* Network Infrastructure and Digital Content, 2nd IEEE International Conference, 2010

Scores

Cacciatore, Maurilio. *Concerto per Tastiera midi, ensemble e live electronics*. Milan, Edizioni Suvini Zerboni, S. 13842 Z., 2011.

Cacciatore, Maurilio. *Corpo d'aria. Quasi una pantomima pe flauto basso, ombre e live electronics.* Milan, Edizioni Suvini Zerboni, S. 14421 Z., 2018.

Cacciatore, Maurilio. *La Vallée des Merveilles, opera da camera per voce narrante, flauto, clarinetto, percussioni, video, live electronics e due danzatori*. Milan, Edizioni Suvini Zerboni, S. 15556 Z., 2017

Cacciatore, Maurilio. *Pulse(s), for alto saxophone and sampo.* Milan, Edizioni Suvini Zerboni, S. 15667 Z., 2018.

Cacciatore, Maurilio. *So loud, per sassofono basso, pianoforte e live electronics.* Milan, Edizioni Suvini Zerboni, S. 15386 Z., 2017.

Cacciatore, Maurilio. *Tralci, per ensemble e live electronics*. Milan, Edizioni Suvini Zerboni, S. 14879 Z., 2015.

Cacciatore, Maurilio. *Tutorial 1: #mimesi, for male voice, sub bass recorder, and live electronics.* Milan, Edizioni Suvini Zerboni, S. 15721 Z., 2018.

Cacciatore, Maurilio. *Lost in feedback, for a percussionist, stage performer and live electronics.* Milan, Edizioni Suvini Zerboni, S. 14788 Z., 2014.

Cacciatore, Maurilio. *Meccanica degli avatar, da un madrigale di Paolo Tenorista, per ensemble ed elettronica*. Milan, Edizioni Suvini Zerboni, S. 15068 Z., 2016.

Cacciatore, Maurilio. *Meccanica della solitudine, per sassofono baritono solista, percussionista co-solista, ensemble, live electronics e dispositivo di scena*. Milan, Edizioni Suvini Zerboni, S. 15667 Z., 2018.

Cacciatore, Maurilio. *Tamonontamo, per quartetto vocale amplificato, coro di 24 voci e live electronics*. Milan, Edizioni Suvini Zerboni, S. 14116 Z., 2018.

Cage, John, *Imaginary landscape n. 1*, 1939, London, Editions Peters, EP6716.

Diez Fischer, Santiago, *Loop's definition,* 2010.

Manoury, Philippe. *Jupiter.* Paris, Editions Durand, DA 505, 1987

Manoury, Philippe. *Pluton.* Paris, Editions Durand, DF 14653, 1989

Manoury, Philippe. *Partita I.* Paris, Editions Durand, DF 15790, 2006

Webography

https://www.audinate.com

https://www.arduino.cc

https://www.ableton.com/en/blog/beap-powerful-modules-max-live/

http://www.birmingham.ac.uk/facilities/aestudios/research/beasttools.aspx

http://brahms.ircam.fr/works/work/10482

http://brahms.ircam.fr/works/work/10493

http://www.clef.sourceforge.net

http://www.essentialdecibels.com/blog/articles/live-sound-explained-3-the-pa-system

http://www.forumnet.ircam.fr

http://www.jagodaszmytka.com/works-f-for-music.html

http://www.lua.org

http://www.musinfo.fr/fr/creation/concours/concours-2018

http://www.musinfo.fr/index.php/en/2014-contest/software-and-manual#Software

http://www.northstar-www.dartmouth.edu/doc/idl/html_6.2
Creating_Widget_Applications.html

https://puredata.info/

http://www.sei.cmu.edu/architecture/start/glossary/moderndefs.cfm

https://www.steinberg.net/

http://www.swift.org

http://violonaroue.fr

https://web.archive.org/web/20050205080537/http://www.javaworld.com/javaworld/
jw-09-2000/jw-0908-letters.html

http://www.wwisa.org