OMax

The Software Improviser

©Marin Lartigues

Version 4.5.x

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

Contents

Chapter 1

Introduction

1.1 System Requirements

Hardware

OMax 4.5.3 has been developed and tested for Apple computers based on an Intel processor only. It has been extensively used on laptops of the Macbook Pro series, models going from Macbook Pro 3,2 (2007 with Core 2 Duo processor) to the most recent Unibody models.

Software

OMax 4.5.3 has been developed and used in Max/MSP 5 in its most recent (and last) version, 5.1.9. It has been tested on Max 6.0.8 and positive feedback has been given on tests on Max 6.1.x as well.

1.2 OMax Principles

1.2.1 Fundamentals

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

1.2.2 Style Modelling

"*By Style Modelling, we imply building a computational representation of the musical surface that captures important stylistic features hidden in the way patterns of rhythm, melody, harmony and polyphonic relationships are interleaved and recombined in a redundant fashion.*". In other words, style modeling is the extraction of implicit rules from a musical material (often referred as the source). Whether the system infers the implicit rules of the source or mimics the surface without trying to guess these rules depends on the representation model used.
In this field, investigations have been done (in collaboration with Shlomo Dubnov) with dictionary based models such as LZ compression algorithm (Incremental Parsing (IP) methods) or Prediction Suffix Trees. Both methods build a tree encoding patterns occurrences and allows to calculate probabilities on the continuation of a given pattern.
In the years 2001-2002 the Music Representation team in IRCAM started to use a novel representation of pattern repetition, Factor Oracle, coming from the genetic research on patterns in genome. Since then, they studied and successfully used this representation for style modelling and music generation and built OMax, a real-time improvising system using stylistic reinjection.

1.2.3 Stylistic Reinjection

The musical hypothesis behind stylistic reinjection is that an improvising performer is informed continually by several sources, some of them involved in a complex feedback loop. The performer listens to his partners. He also listens to himself while he is playing, and the instantaneous judgment he bears upon what he is doing interferes with the initial planning in a way that could change the plan itself and open up new directions.
Sound images of his present performance and of those by other performers are memorized and drift back in memory from present to the past. From the long-term memory they also act as inspiring sources of material that would eventually be recombined to form new improvised patterns. Musical patterns are supposed not to be stored in memory as literal chains, but rather as compressed models, that may develop into similar but not identical sequences: this is one of the major issues behind the balance of recurrence and innovation that makes an interesting improvisation.
The idea of stylistic reinjection is to reify, using the computer as an external memory, this process of reinjecting musical figures from the past in a recombined fashion, providing an always similar but always innovative reconstruction of the past. To that extent, OMax, as a virtual partner will look familiar, as well as challenging, to his human partner. The interesting thing is to see how the human partner reacts to his "clone" and changes his improvisation accordingly.

1.3 Quick Start

1.3.1 Audio

1. Starting point to get OMax working (after checking the system requirements) is to **load the OMax.4.5.x.maxpat patch in Max 5**. This patch is highlighted in red in the OMax Patches&Objects folder. Or you can use the alias pointing to it present in the OMax4.5.x folder. It may take one or two minutes as the patch embeds a lot of others. It should print about 35 lines in your Max window with the credits for external objects and OMax.data, OMax.oracle and OMax.buffers declarations. However, none of them should be red (error message). Next steps will make reference to the capture presented Figure 1.1.

2. If you want to use OMax on a live input, **adjust the channels** (accordingly to your hardware and Max I/O mappings) then **check the Input toggle** (frames 2, top left corner of the patch).
   NB: The first channel (named rec) is used to record the sound in a buffer (and replayed later on) while the second channel (named dtct) is fed into the detection (analysis) system. These two channel can receive the same stream if you use only one microphone.
   If you want to use OMax on a sound file, click on the open message of the Play file box and choose an aiff file in the navigator popping up. Checking and unchecking the toggle (frame 2b, top left corner of the box) will start and stop the playing of your file once the audio is turned on.

3. Provided that you had no errors in the Max window when you loaded the patch, you can now **turn on the audio** with the "speaker" button (frame 3, bottom right side of the patch).

4. You can **check and adjust the level** of the sound arriving to OMax with the horizontal vumeters (and their overlaid sliders) in and on the right of the Record box (frame 4). You can also **monitor** this stream by checking the top left toggle of the Monitor box (frame 4b) — you may need to adjust the channels of your monitoring system with the numbers in the same box.
   The Pitch and Spectral boxes **LEDs should start flashing** (frame 4) as the sound gets through. It indicates that they are detecting some activity and able to extract information from the audio. Otherwise you need to continue adjusting levels and/or detection parameters (see 2.2.3 or 2.2.4).

5. When you are ready, you can **start recording and learning** by checking the big toggle labelled Start Learning on the top of the patch (frame 5).
   If you work with a sound file, you can stop and start reading the file again from the beginning with the toggle of the Play file box (frame 2b).
   Model should start being built, Size and Hi Ctxt of the Dual_MIDI and Dual_SP boxes should start growing.

6. You can **activate the visualization** of the knowledge by checking the top toggle of the Visu box (frame 6) and looking in the Jitter window. At least an horizontal black line should appear and hopefully some white and grey arcs below and above this timeline proving that OMax is recognizing some interesting patterns.

7. OMax is now ready to improvise! **Check** the top left toggle of **one of the players and it should start playing** (frames 7). You can adjust the channel and volume of each of the players in the Output box right below.

8. Hit the Stop! button (frame 8, on the top of the patch) to **stop all the players at once**. And **reset** the Audio part of OMax **with the Reset button** underneath the Start Learning toggle (frame 8, on the top of the patch) and to be ready to start again a new improvisation.

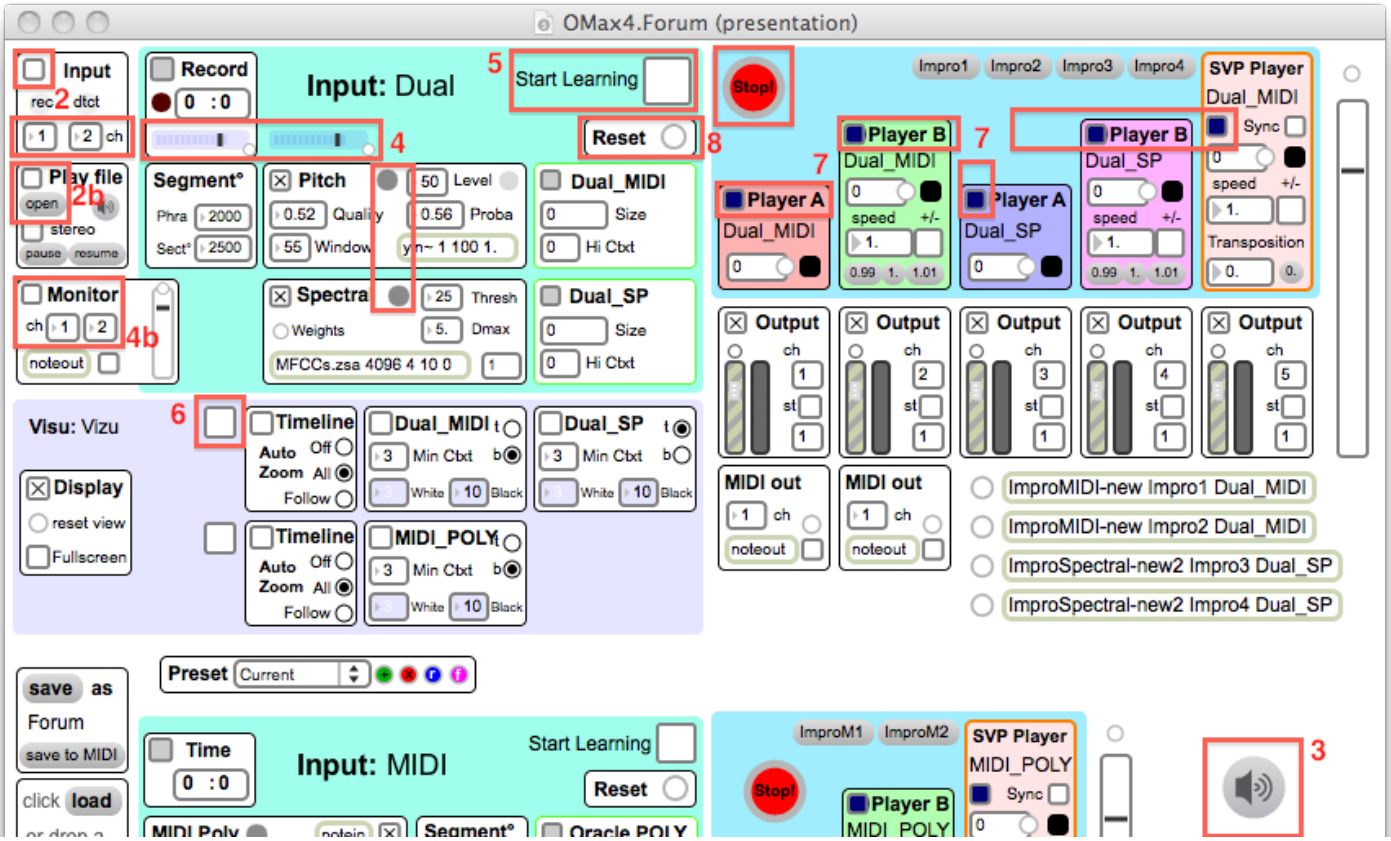


Figure 1.1: Audio Quick Start

1.3.2 MIDI

1. Starting point to get OMax working (after checking the system requirements) is to **load the OMax.Forum patch in Max 5**. This patch is highlighted in red in the OMax4.5 folder. It may take one or two minutes as the patch embeds a lot of others. It should print about 35 lines in your Max window with the credits for external objects and OMax.data, OMax.oracle and OMax.buffers declarations. However, none of them should be red (error message). Next steps will make reference to the capture presented Figure 1.2.

2. First you need to **define a MIDI input** to receive data from by double-click on the notein object in the MIDI Poly box (frame 2). You can also adjust the channel you want to listen to with the number box right below (frame 2). A value of -1 means *all channels*.
   NB: if you want to play a MIDI file and feed it to OMax, you need to use an external software as *Sweet MIDI Player* to play and route the MIDI stream to Max/MSP.
   The **LED** of the MIDI Poly box should start flashing as soon as you receive MIDI data.

3. You are now set to **start the learning** of OMax by checking the toggle labelled Start Learning (frame 3, on the top of the MIDI part of the patch).

4. You can **activate the visualization** of the knowledge by checking the bottom toggle of the Visu box (frame 4) and looking in the Jitter window. At least an horizontal black line should appear and hopefully some white and grey arcs below and above this timeline proving that OMax is recognizing some interesting patterns.

5. Before making OMax improvise, you need to **adjust the MIDI output of the players**. Double-click on the noteout box of each of them (frame 5) to decide if you want to use the default synthesizer of the Apple Operating System, *AU DLS Synth* or if you want to send the MIDI stream to another synthesizer you may have on your computer.

6. OMax is now ready to improvise! **Check** the top left toggle of **one of the players and it should start playing** (frames 6).

7. Hit the Stop! button (frame 7, near the players) to **stop all the players at once**. And **reset** the MIDI part of OMax **with the Reset button** on the left of the Stop! button (frame 7) to be ready to start again a new improvisation.

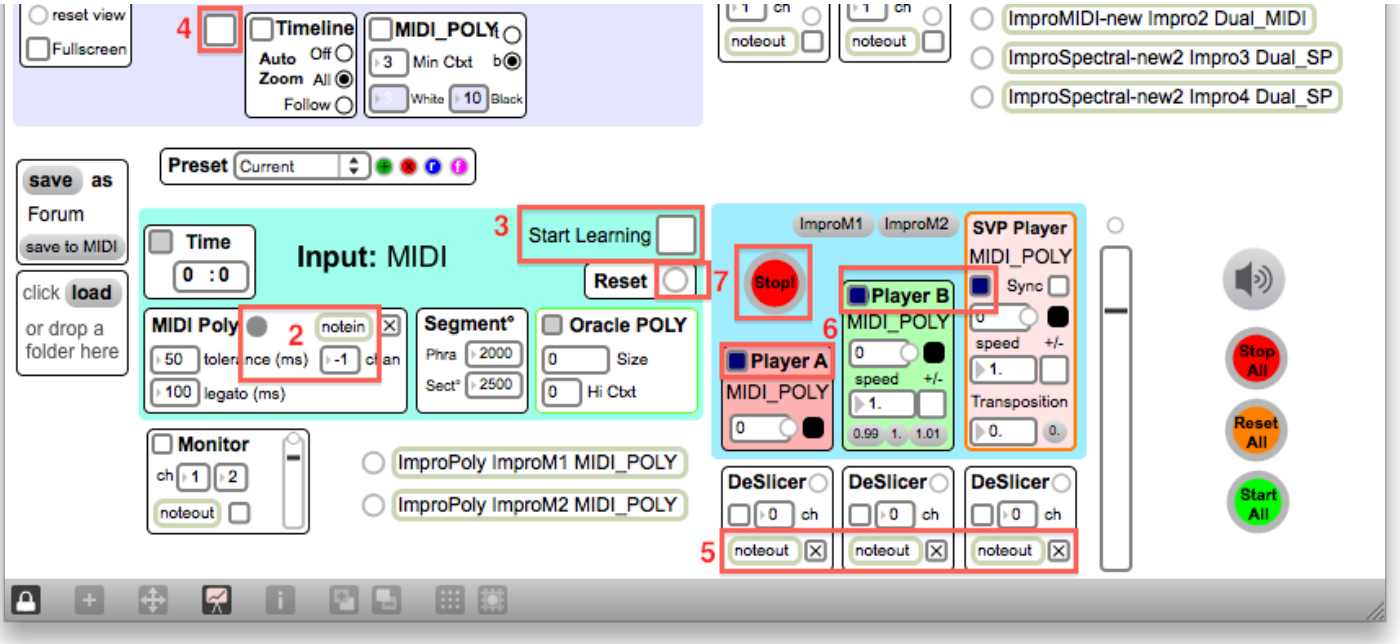


Figure 1.2: MIDI Quick Start

1.4 Architecture

1.4.1 Overall view

From a very distant point of view, OMax is built in two parts. A first ensemble is in charge of the listening and learning process and a second ensemble is the improvising part itself. The analysis is conceptually divided in three stages. A first stage extracts from the incoming stream some abstract description based on signal processing principles (which will be described later in this documentation). Once this **detection** has been done, a **segmentation** stage takes the abstract stream (of numbers mostly) from the detection to aggregate them into consistent units that have a symbolic coherence and a defined value.
For example, in the common case of pitch information, the detection stage will be an algorithm for fundamental frequency extraction. It will output a continuous stream of frequencies that need to be filtered and grouped by the segmentation to give a symbolic information of note defined by its onset, offset and pitch value (possibly given also with a velocity value).
The result of these two steps of analysis is a symbolic representation of the input stream that can be learnt in the pattern recognition model of OMax. This **modelling** is done by constructing the Factor Oracle graph previously mentioned.

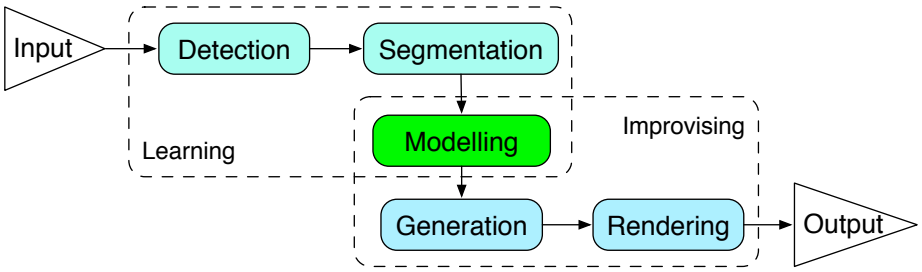


Figure 1.3: Overall architecture of OMax

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

1.4.2 Functions

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

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

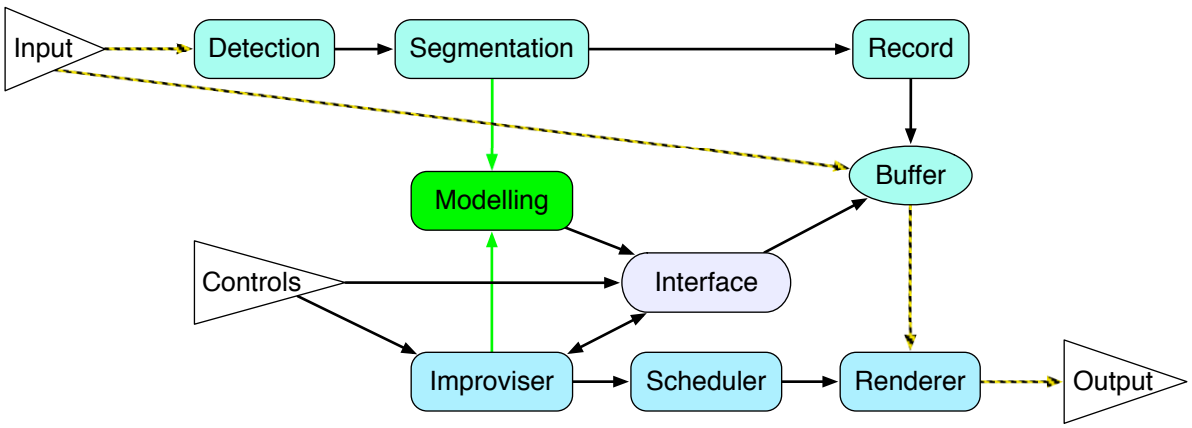


Figure 1.4: Functions in OMax

1.4.3 Externals

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

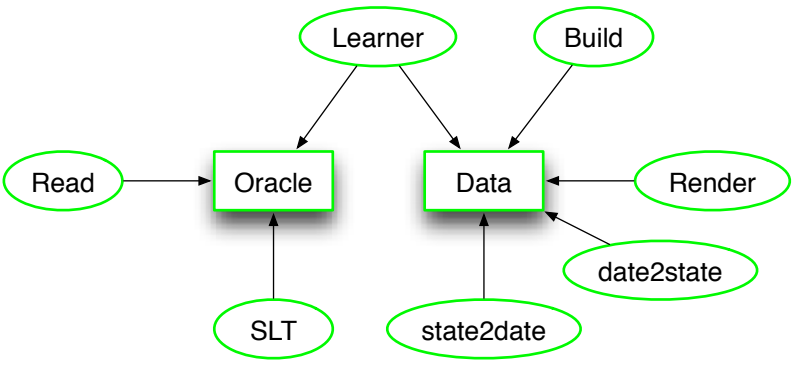


Figure 1.5: External objects for OMax 4.x

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

Chapter 2

Modules

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

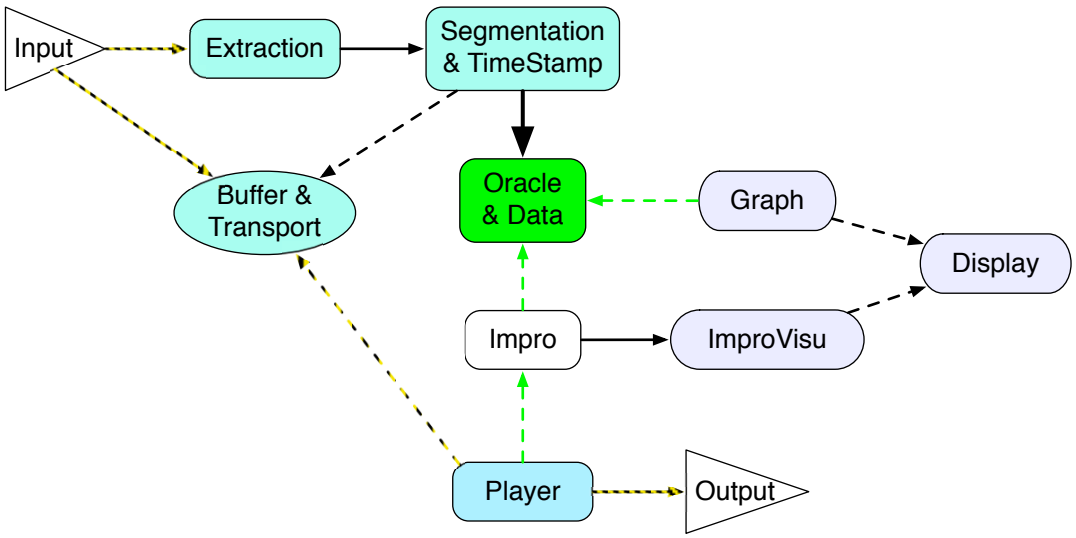


Figure 2.1: Modules of OMax

The **Extraction** module is in charge of computing the detection information on the input stream. It passes the result to the **Segmentation & TimeStamp** module which does the gathering of consistent and sequential description data to constituted the symbolic units and dates them with respect to the recording time. Timestamping is the link between the symbolic units describing the input and the actual recording of the stream made in the **Buffer & Transport** module.
Then, the symbolic information is organised and stored in the **Oracle & Data** module which is thus the core of OMax knowledge. The **Graph** module periodically reads the model and gives a visualisation of its state which will be displayed by the **Display** module thanks to Jitter.
The **Impro** module navigates in the knowledge model with respect to the parameters and constraints given through the interface and computes a new path to generate a “clone”. This path is graphed on the

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

2.1 The two and a half worlds of OMax

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

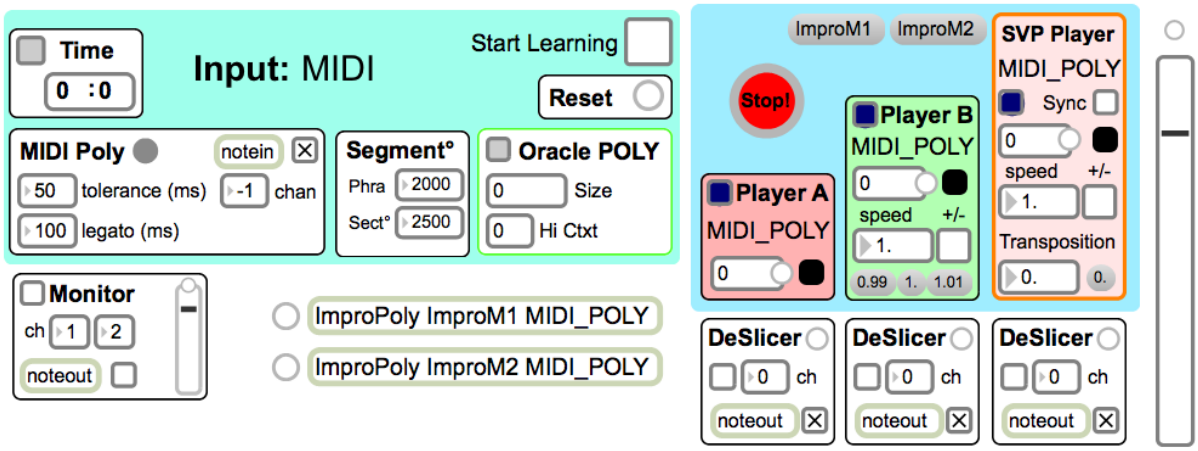


Figure 2.2: MIDI part of OMax

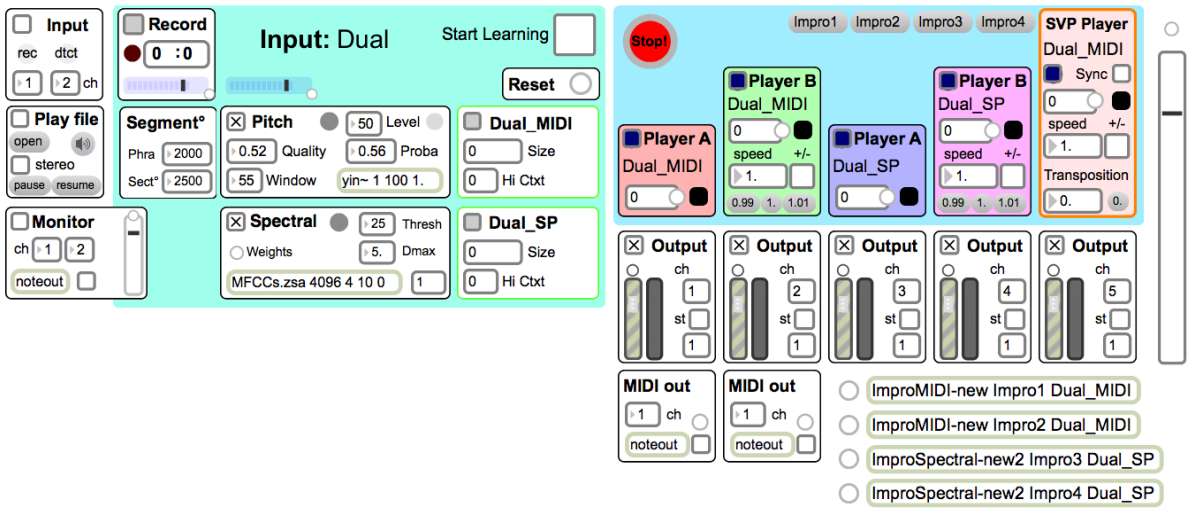


Figure 2.3: Audio part of OMax

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

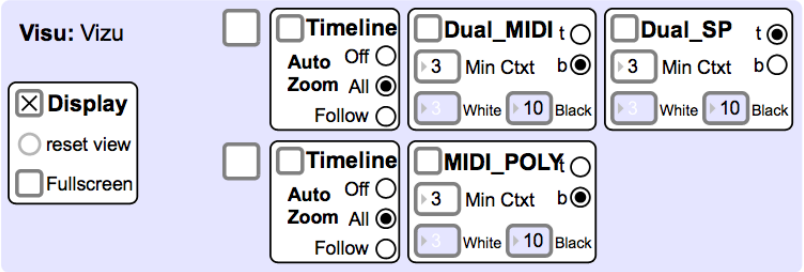


Figure 2.4: Visualization part of OMax

2.2 Input(s)

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

2.2.1 MIDI

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

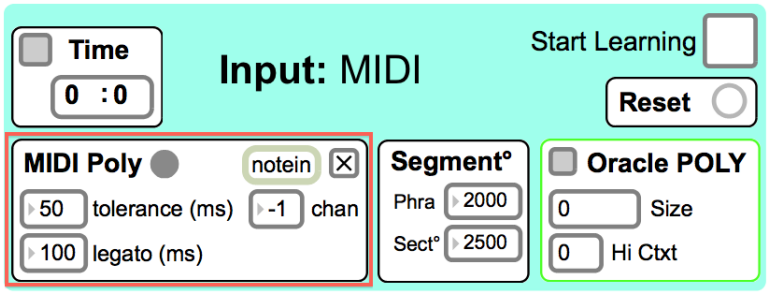


Figure 2.5: MIDI input section of OMax

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

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

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

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

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

2.2.2 Audio

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

Input: if you have one or two microphones plugged on your computer, you can activate the "live input" patch by ticking the toggle on the top left corner and use them. Once you have turned on the DSP of Max/MSP, the two flashing green LEDs named **rec** (for recording) and **dtct** (for detection) will tell you if it gets any signal. You can adjust the channel for the recording and the detection separately with the two corresponding number boxes labelled **ch** (for channel).
The detection channel goes into the different extraction and segmentation stages while the recording channel is recorded into the buffer as explained in section 1.4.2. It is sometimes better to use two different microphones for those two processes. For example a close field mic, less sensible to the environment (if there are several musicians for instance) may be more efficient for detection while a farther mic will give more bass and air for a better quality recording.

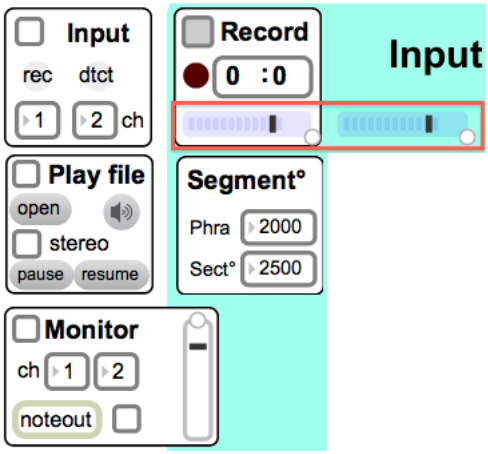


Figure 2.6: Audio Input of OMax

Play file: if you want to use a sound file as audio input for OMax, you can use the small player embedded. Start by choosing the file with the **open** button. Then, once the DSP of Max/MSP is on, checking and unchecking the **toggle** (top left corner of the **Play file** frame) starts and stops the playing. You can also **pause** and **resume** the playing with the buttons on the bottom of the frame.
The **stereo** toggle lets you choose the routing of the channels if you file is stereo. When uncheck (default), both channels of your file are summed and fed identically into both the detection and recording channels. If you check the **stereo** toggle, the first channel (left) of your file is fed into the recording while the second (right) is routed to the detection.

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

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

2.2.3 Pitch

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

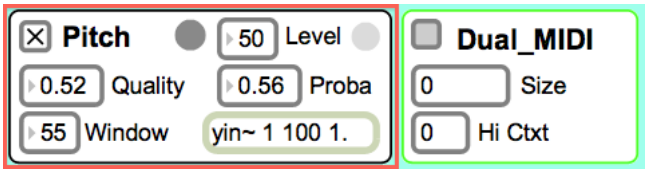


Figure 2.7: Pitch detection in OMax

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

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

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

2.2.4 Spectral

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

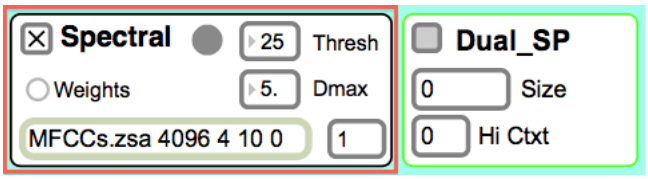


Figure 2.8: Spectral detection in OMax

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

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

2.2.5 Common interfaces

Silence segmentation

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

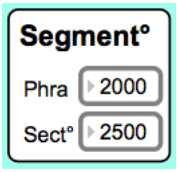


Figure 2.9: Silence segmentation

Current state of the model

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

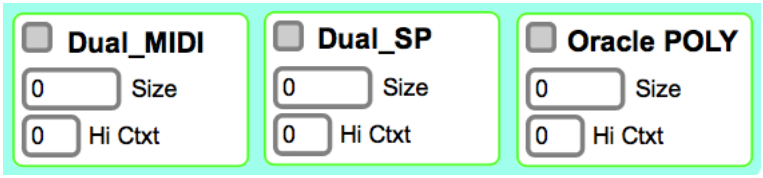


Figure 2.10: Current state of the Factor Oracle

2.3 Improvisation(s)

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

2.3.1 Core

In OMax 4.x the navigation strategy follows the functional diagram presented opposite (figure 2.11).
The original sequence is read for a few consecutive elements (ie a few notes or a short duration) that we name **continuity** then, it is time to diverge from the original sequence and the system decides to **jump**.
OMax has a small anticipation system to be able to find a good transition (variation) so it searchs and **collects** the different solutions over a short **searching window**.
It **selects** next all these solutions based on different criteria both automatic (as a **taboo** system to avoid strict loops) and user based (as **regions** which allow to choose the part of the past serving as source). And discards solutions not following these rules.
Remaining solutions are **described** musically with parameters which strongly depends on the type of analysis and each solution is **weighted** with a mixture of these parameters. So the pool of solutions is ordered according to this weight.
Finally, one of the solution is **drawn** (among the best), validated and planned as the next jump. The destination of this jump starts a new continuous sequence and buckles the cycle.

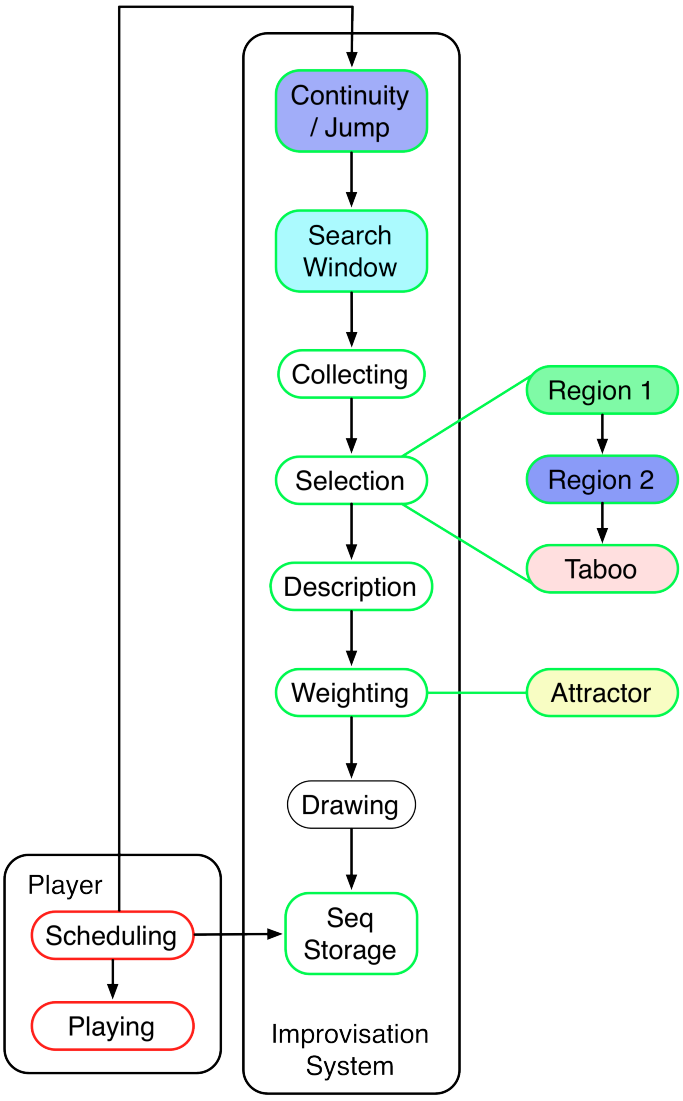
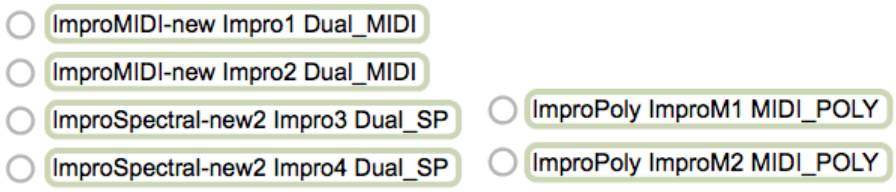


Figure 2.11: Improvisation & Player System

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



(a) Audio improvisation boxes (b) MIDI improvisation boxes

Figure 2.12: Improvisation cores boxes

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

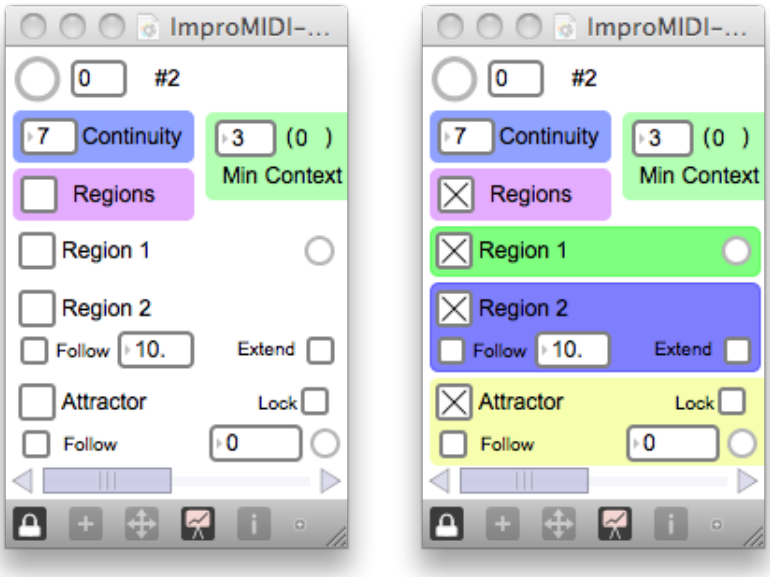


Figure 2.13: Common Improvisation Interface

Continuity : acts as a kind of *variation rate*. The higher, the longer consecutive chunks will be (so the lesser OMax will diverge from the original material).
⚠ Typical value for continuity strongly depends on the type of analysis.

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

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

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

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

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

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

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

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

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

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

2.3.2 Players

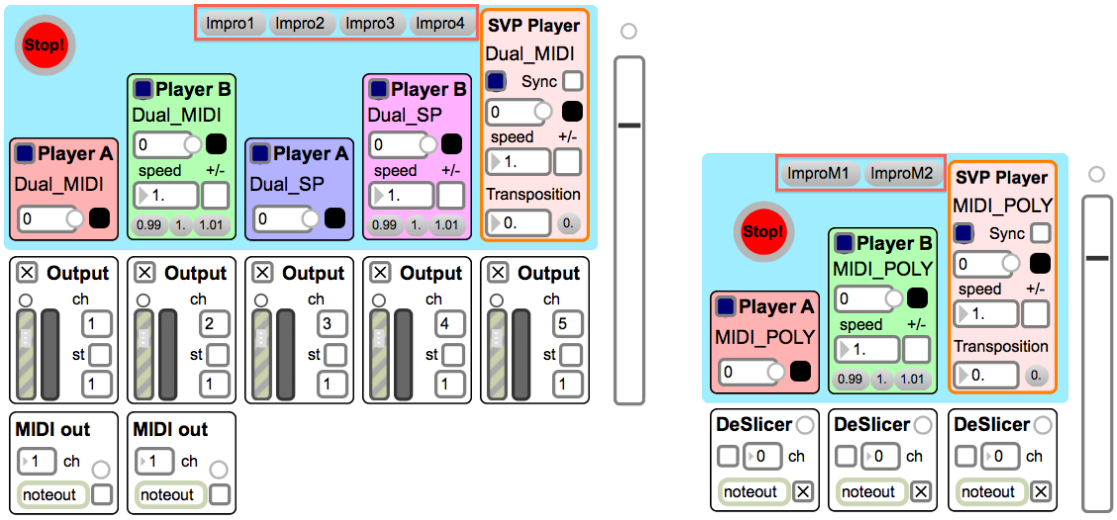
Core patches presented in the previous paragraph (2.3.1) generates improvised sequence however they do not play them. Players, located in the right half of the main OMax patch are in charge of effectively playing the "clones".

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

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

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

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



(a) Audio Players

(b) MIDI Players

Figure 2.14: Players

Player A

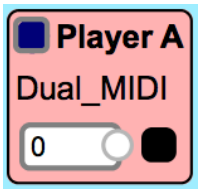


Figure 2.15: Player A

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

Player B

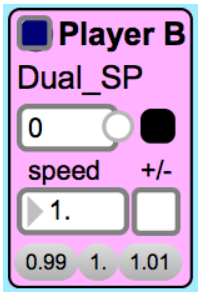


Figure 2.16: Player B

Players of type B works the same way as players of type A: start and stop **toggle** is in the top left corner on a blue background, flashing when a variation occurs. The **number** indicates the size of the sequence already played and the **black button** triggers a smooth ending.
Besides these basic functions, players of type B provides a variable speed option. The **floating-point** speed **box** is a multiplicative factor going from 5. to −5. Normal speed is 1., 2. means two times faster, 0.5 half the normal speed (ie. two times slower) and so on. Negative speeds automatically ticks the **toggle** box labelled +/- and means reading the sound backward. However these changes of speed are link with a “natural” transposition exactly as when rotating old vinyls faster or slower.

SVP Player

SVP players make use of the Super Phase Vocoder developed at IRCAM and allows on top of the regular functions of the other players to transpose and/or change the speed independently. Start/stop **toggle**, smooth ending **button** and speed **box** work exactly as in Players of type B. Another **floating-point number box** allows to transpose finely the whole clone from 18 semi-tones below to 18 semi-tones above the original tuning (a value of 1. meaning one semi-tone above).
A SVP player can also be synchronized on another player (of type A or B) and thus used to harmonize a clone with a transposition for example. You need first to choose on which of the other players you want to synchronize with the buttons labelled *Impro1...4* (or *ImproM1...2* in the case of MIDI section) framed in red on Figure 2.14. Then you can tick the *Sync* **toggle** to synchronize the SVP player. You can check that they are indeed well synchronized by watching the **number box** showing the state being read.

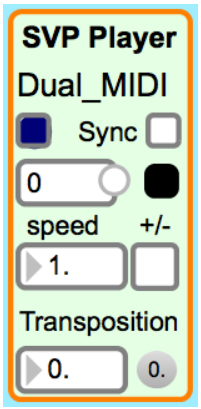
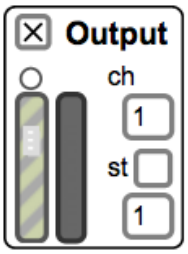
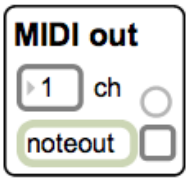


Figure 2.17: Player SVP

Output



(a) Audio Output



(b) MIDI Output

Figure 2.18

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

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

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

Figure 2.19

2.4 Visualisation

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

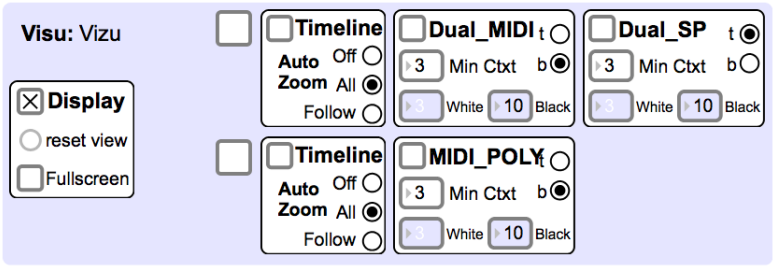


Figure 2.20: Visualization Control Interface

Display

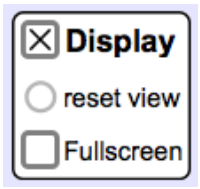


Figure 2.21

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

Timeline

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

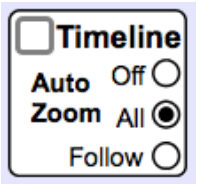


Figure 2.22

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

Links

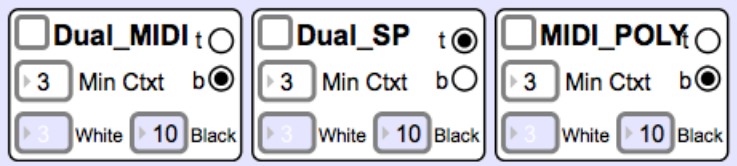
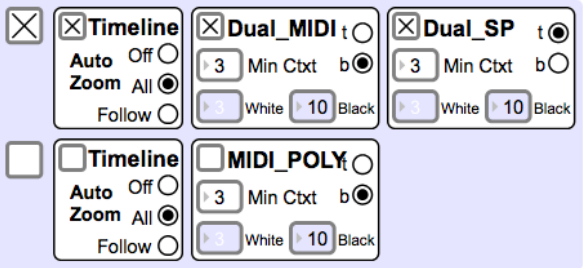
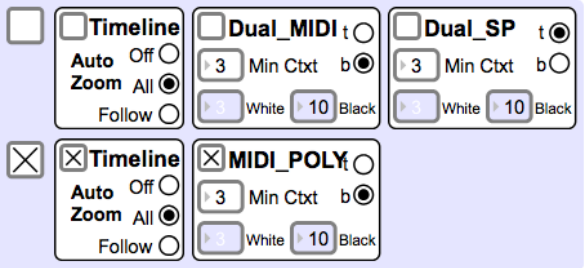


Figure 2.23: Control of the visualization of the links

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



(a) Full Audio Visualization



(b) Full MIDI Visualization

Figure 2.24: Toggle Shortcut for Visualization

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

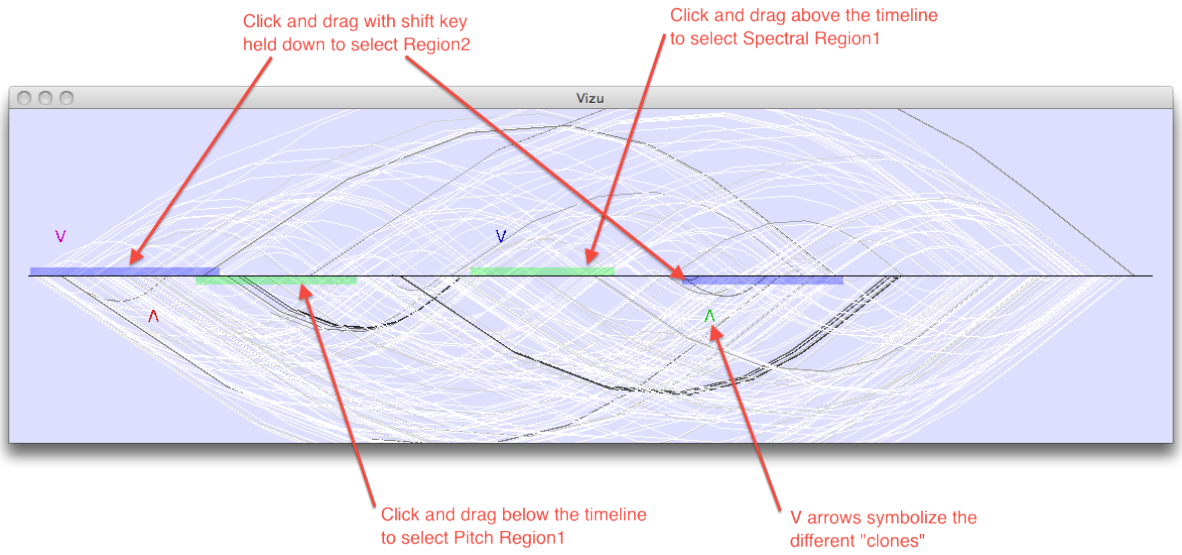


Figure 2.25: Example of the Visualization Window with Regions

Chapter 3

Advanced Usage

This section presents more advanced features of OMax 4.5.x. It assumes that you are comfortable enough with Max/MSP so that you know how to edit a patch, switch between presentation and edit mode, use the embedded help on inlets and outlets and the inspector, know the concepts of objects, abstractions or bpatchers and other basic Max/MSP functions.

3.1 Controling OMax with Messages

The inputs (inlets) and outputs (outlets) of all the modules of OMax are documented in the patch. That means that if you mouse over any inlet/outlet it will display information on the purpose of this inlet/outlet. You can use those inlets at your convenience to interact directly with any module with Max messages (and toggles etc.) instead of using the graphical interface. Most of the inlets correspond to parameters described earlier in this documentation. For example, Figure 3.1 shows the first (left-most) inlet of the Player A (see 2.15) which is used to start and stop the "clone".

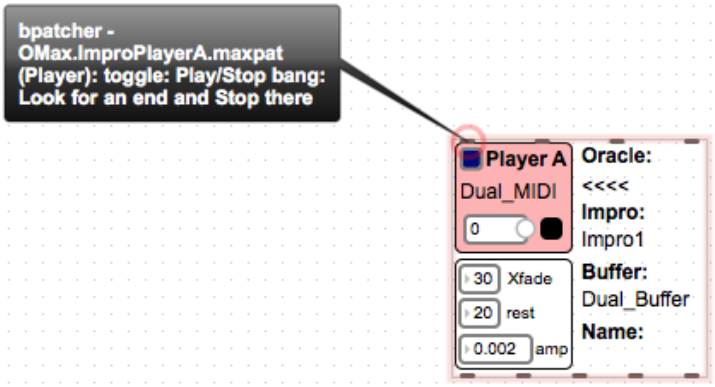
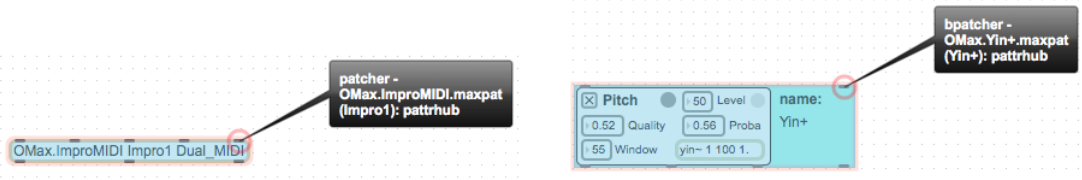


Figure 3.1: Example of an inlet and its help

Few inlets though are used for purposes that are not previously described. The embedded info should be self-explanatory or this inlet is not controlling any user-level parameter and you should not use it either. The last (right-most) inlet of every module (either loaded as an abstraction or in a bpatcher) which includes settable parameters is always named **patterhub**. It is used to access and change all the parameters of the module even if some of them are not shown in the graphical interface. Figure 3.2 shows this inlet for the Improvisation core module (see 2.3.1) which is loaded as an abstraction (Figure 3.2a) and for the Pitch extraction module (Figure 3.2b, see 2.2.3) which is loaded in a bpatcher. This inlet



(a) abstraction (b) bpatcher

Figure 3.2: pattrhub inlet (right-most)

allows to send direct messages to the module with the name of a parameter (even if the parameter is hidden in the graphical interface) and a value to assign to this parameter.

Figure 3.3

This mechanism relies on the **pattr** system of Max/MSP. Figure 3.3 shows the simple set of **pattr** objects used in every module. If you double-click on the **pattrstorage** object, a list of all the parameters accessible in the module you are exploring will pop-up. As explained in the documentation of the **pattrstorage** object, this *Client Objects* list reflects the state of all the parameters watched by the pattr system. If you change any parameter either with the graphical interface or with a message, the current value of that parameter will be reflected in this window as well. This way, it is very easy to link a parameter to the corresponding name in the pattr system and know which message to send in the **pattrhub** inlet of the module. Figure 3.4 shows an example of such control of the Improvisation core module (see 2.3.1) with a few messages and the associated list of parameters.

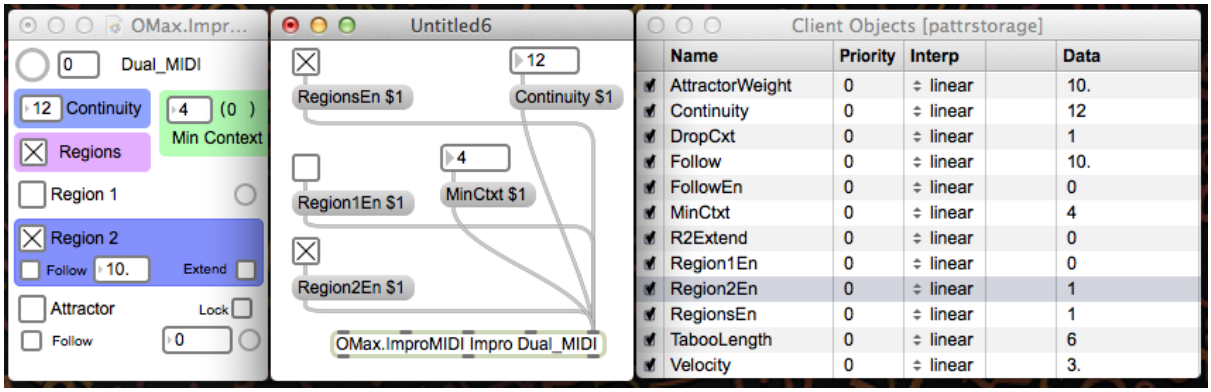


Figure 3.4: Example of an interaction through messages to the pattr system of the Improvisation core

Regions

Aside from the pattrhub system, there are messages received in the **Selections** subpatcher (Figure 3.5) to remotely set the regions used to control the location of the "clones". The boundaries of the regions should be sent using the number of the state in the model for the beginning and end of each region. This pair of integer is to be sent to **Dual_MIDI-selection1** or **Dual_MIDI-selection2** for the regions of the pitch description, **Dual_SP-selection1** or **Dual_SP-selection2** for the regions on spectral description or **MIDI_POLY-selection1** or **MIDI_POLY-selection2** for the regions of the MIDI input.

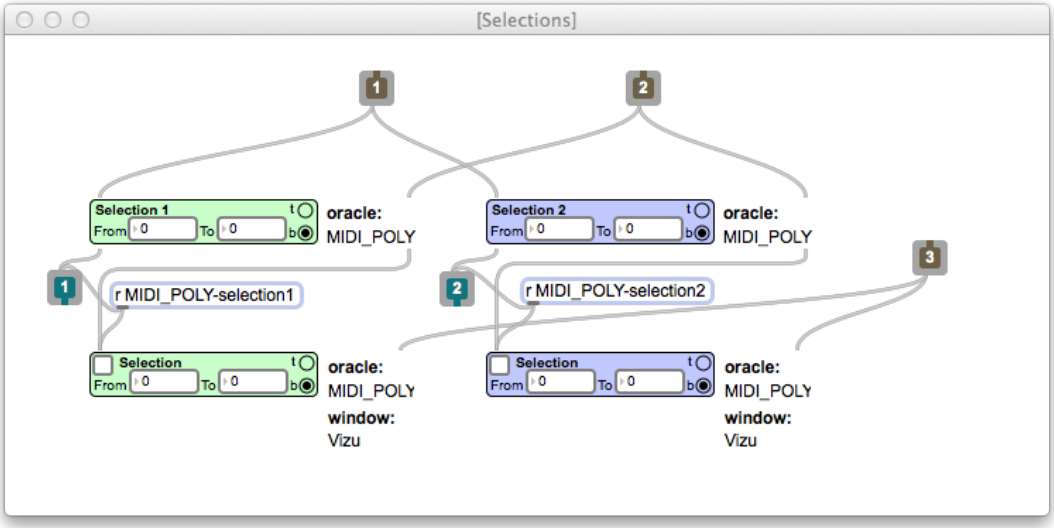


Figure 3.5: Receive objects to remotely set regions (presented here for the MIDI input)

To convert from time to state number in any of the models, you can use the **OMax.date2state** external object which takes as argument the name of the model you want to refer to. You can also use the elapsed recording time (that is the most recent date along the timeline of the learning) which is output from the right outlet of the **OMax.Buffer** module (for the audio input) or the **OMax.Time** module (for the MIDI input) as shown Figure 3.6.

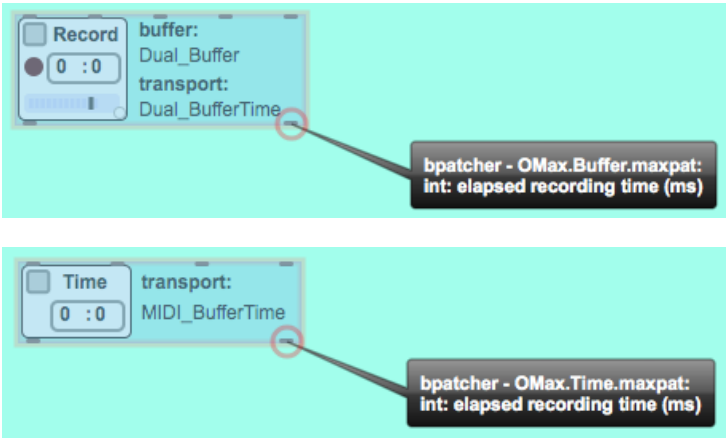


Figure 3.6: Elapsed Time Output

3.2 References Through Names

3.2.1 Names

There are in OMax 4.5.x a few absolute names which constitute univocal references to underlying data structures as the audio buffer, the internal model (see 1.4.1), the jitter window of the visualization part etc. The names are declared to Max/MSP when loading the OMax patch and can not be changed afterward without closing it. Therefore they will be conflicts if you load two times the main OMax 4.5.x for example. The architecture of the names (except for the visualization part) is illustrated Figure 3.7.

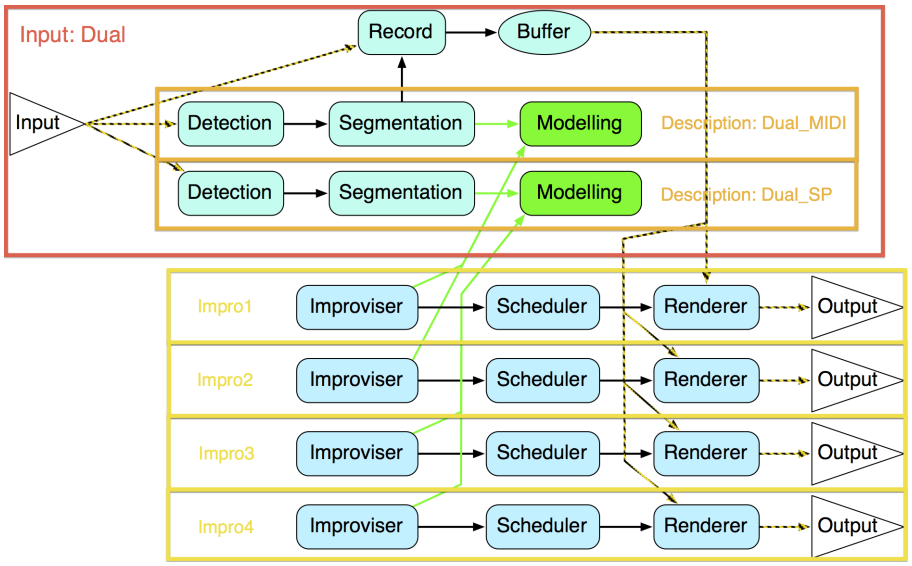


Figure 3.7: The differents names in OMax audio

There are 4 types of names: the first level is the name of the input. In the OMax 4.5.x patch, the audio input is named **Dual** and the MIDI input is named **MIDI**. Inside those input there can be different types of analysis/modeling chains which correspond to a second level of names. For example in the audio input of OMax 4.5.x, you can find two types of descriptions: pitch and spectral, the first description is referred as **Dual_MIDI** (which is ambiguous and actually do not correspond to any MIDI processing but is an heritage of the previous versions of OMax). The spectral description of the **Dual** input is named **Dual_SP**. The MIDI part of OMax 4.5.x (**MIDI** input) uses only one type of analysis and internal model named **MIDI_POLY**.

Each "clone" of OMax rely on an Improvisation core (see 2.3.1). These cores are also named, so that the different players (see 2.14) can effectively render the "clones". In the audio part of OMax 4.5.x, there are 4 improvisation cores named **Impro1**, **Impro2**, **Impro3** and **Impro4**. As explained on Figure 2.12, the first two cores use the pitch description (**Dual_MIDI**) while the last two cores use the spectral description of the input (**Dual_SP**). The MIDI improvisation cores, **ImproM1** and **ImproM2** refer to the only available description of the MIDI input: **MIDI_POLY**.

Lastly, the visualization is also referred through its name. In the OMax 4.5.x patch, the visualization name is **Vizu** which correspond to the jitter window in which it is rendered. All the modules which have to draw in the visualization have to use this name. Figure 3.8 shows the usage of names for the module in charge of rendering the pitch description (**Dual_MIDI**) in the visualization window (**Vizu**).

3.2.2 Changing Names

All the names described in the previous section are mainly used as parameter of objects, abstractions or bpatchers. In the case of objects and abstractions, they are use explicitly as in Figure 2.12. However, in the case of the bpatchers, those references are to be found in the bpatcher inspector (see Figure 3.8 for example). This makes the tracking of all the references in the OMax patch a very tedious task. However, if you would like for example to run two copies of the OMax patch, side by side in the same Max/MSP application, you will have to change every name of one of them.

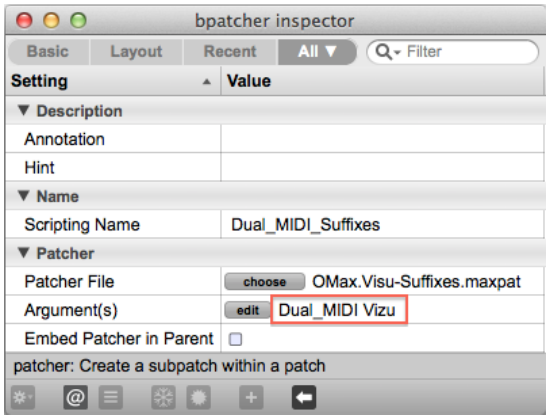


Figure 3.8

Therefore an safer and faster method is given here:

- Edit one of the copies of the patch in text mode (to benefit from search/replace functions).
- Then do the following searches and replacements in this copy:
  Caution! Doing search and replace, you need to keep extensions like _MIDI, _SP, _POLY etc.
  - Search for all the mentions of " Dual " and replace them all by " Duo " (for example). This way, you will have independent audio inputs and models.
  - Search for all the mentions of " Vizu " and replace them all by " Disp " (for example). This way, you will have 2 visualization windows.
  - Search for all the mentions of " Impro1 ", " Impro2 ", " Impro3 ", " Impro4 " and replace them by " Clone1 ", " Clone2 ", " Clone3 ", " Clone4 " (for example).
- If you are interested by the MIDI part of OMax as well:
  - Search for all the mention of " ["MIDI"] " (brackets included with their content) and replace them all by " ["14H"] " (for example)
  - Search for all the mentions of " MIDI_POLY " and replace them all by " 14H_POLY " (for example, or the same name you used for the previous step, appended with _POLY).
  - Search for all the mentions of " ImproM1 " , " ImproM2 " and replace them by " ImproH1 " , " ImproH2 " (for example)
- Save the patch (with a different name than the " original " OMax patch)
- Finally load the " orginal " (not modified) OMax 4.5.x patch
- Then load the modified copy of the patch.

You should not have any error message when loading the second copy (either saying that a structure is missing or that the name is already in use) and everything should work well. Otherwise, you probably missed some name replacements.

Bibliography

OMax

[LBA12] Benjamin Lévy, Georges Bloch, and Gérard Assayag. "OMaxist Dialectics : Capturing, Visualizing and Expanding Improvisations". In: *NIME*. Ann Arbor, USA, 2012, pp. 137–140.

[Ass09] Gérard Assayag. "Habilitation à diriger les recherches - Algorithmes, langages, modèles pour la recherche musicale : de la composition à l'interaction". Thèse de doctorat. Université Bordeaux I, 2009.

[Rob+09] Yann Robin et al. *Projet de recherche OMAX (étape I) : Composer in Research Report (2009)*. Tech. rep. Médiation Recherche et Création, 2009.

[BDA08] Georges Bloch, Shlomo Dubnov, and Gérard Assayag. "Introducing Video Features and Spectral Descriptors in The OMax Improvisation System". In: *International Computer Music Conference*. 2008.

[AB07] Gérard Assayag and Georges Bloch. "Navigating the Oracle: a Heuristic Approach". In: *International Computer Music Conference '07*. Copenhagen, Denmark, 2007, pp. 405–412.

[ABC06a] Gérard Assayag, Georges Bloch, and Marc Chemillier. "Improvisation et réinjection stylistiques". In: *Le feed-back dans la création musicale contemporaine - Rencontres musicales pluri-disciplinaires*. Lyon, France, 2006.

[ABC06b] Gérard Assayag, Georges Bloch, and Marc Chemillier. "OMax-Ofon". In: *Sound and Music Computing (SMC) 2006*. Marseille, France, 2006.

[Ass+06] Gérard Assayag et al. "Omax Brothers : a Dynamic Topology of Agents for Improvization Learning". In: *Workshop on Audio and Music Computing for Multimedia, ACM Multimedia 2006*. Santa Barbara, USA, 2006.

Improvisation, Oracle

[CDA11] Arshia Cont, Shlomo Dubnov, and Gérard Assayag. "On the Information Geometry of Audio Streams with Applications to Similarity Computing". In: *IEEE Transactions on Audio, Speech, and Language Processing* 19-1 (2011).

[Con+10] Arshia Cont et al. "Interaction with Machine Improvisation". In: *The Structure of Style*. Ed. by Shlomo Dubnov Kevin Burns Shlomo Argamon. Springer Verlag, 2010, pp. 219–246.

[Man+10] Fivos Maniatakos et al. "On architecture and formalisms for computer assisted improvisation". In: *Sound and Music Computing conference*. Barcelona, Spain, 2010.

[DA08] Shlomo Dubnov and Gérard Assayag. "Memex and Composer Duets: computer aided composition using style modeling and mixing". In: *Open Music Composers book 2*. Ed. by G. Bresson J. Agon C. Assayag. Collection Musique Sciences ircam Delatour, 2008.

[CDA07a] Arshia Cont, Shlomo Dubnov, and Gérard Assayag. “Anticipatory Model of Musical Style Imitation using Collaborative and Competitive Reinforcement Learning”. In: *Anticipatory Behavior in Adaptive Learning Systems*. Berlin, Germany: Martin Butz, Olivier Sigaud, and Gianluca Baldassarre, 2007, pp. 285–306.

[CDA07b] Arshia Cont, Shlomo Dubnov, and Gérard Assayag. “GUIDAGE: A Fast Audio Query Guided Assemblage”. In: *International Computer Music Conference*. Copenhagen, Denmark, 2007.

[DCA07] Shlomo Dubnov, Arshia Cont, and Gérard Assayag. “Audio Oracle: A New Algorithm for Fast Learning of Audio Structures”. In: *International Computer Music Conference*. Copenhague, Denmark, 2007.

[CDA06] Arshia Cont, Shlomo Dubnov, and Gérard Assayag. “A framework for Anticipatory Machine Improvisation and Style Imitation”. In: *Anticipatory Behavior in Adaptive Learning Systems (ABiALS)*. Rome, Italy, 2006.

[RAD06] Camilo Rueda, Gérard Assayag, and Shlomo Dubnov. “A Concurrent Constraints Factor Oracle Model for Music Improvisation”. In: *XXXII Conferencia Latinoamericana de Informática CLEI 2006*. Santiago, Chile, 2006.

[AD05] Gérard Assayag and Shlomo Dubnov. “Improvisation Planning and Jam Session Design using concepts of Sequence Variation and Flow Experience”. In: *Sound and Music Computing 2005*. Salerno, Italie, 2005.

[Aya+05] Mondher Ayari et al. *De la théorie musicale à l’art de l’improvisation : Analyse des performances et modélisation musicale*. Ed. by AYARI Mondher. Paris: DELATOUR-France, 2005.

[AD04] Gérard Assayag and Shlomo Dubnov. “Using Factor Oracles for machine Improvisation”. In: *Soft Computing* 8-9 (2004). Ed. by M. Chemillier G. Assayag V; Cafagna, pp. 604–610.

Style Modeling

[Dub+03] Shlomo Dubnov et al. “Using Machine-Learning Methods for Musical Style Modeling”. In: *IEEE Computer* 10-38 (2003). Ed. by Doris L. Carver, pp. 73–80.

[DA02] Shlomo Dubnov and Gérard Assayag. “Universal Prediction Applied to Stylistic Music Generation”. In: *Mathematics and Music: A Diderot Mathematical Forum*. Ed. by Feichtinger H.G. Rodrigues J.F. Assayag G. Berlin, Allemagne: Springer, 2002, pp. 147–158.

[Ass+01] Gérard Assayag et al. “Automatic modeling of musical style”. In: *8èmes Journées d’Informatique Musicale*. Bourges, France, 2001, pp. 113–119.

[Lar+01] Olivier Lartillot et al. “Automatic Modeling of Musical Style”. In: *International Computer Music Conference*. La Havane, Cuba, 2001, pp. 447–454.

[ADD99] Gérard Assayag, Shlomo Dubnov, and Olivier Delerue. “Guessing the Composer’s Mind : Applying Universal Prediction to Musical Style”. In: *ICMC: International Computer Music Conference*. Beijing, China, 1999.

[Dub98] Shlomo Dubnov. “Universal Classification Applied to Musical Sequences”. In: *ICMC: International Computer Music Conference*. Ann Arbor Michigan, USA, 1998.

Student works

[Lév09] Benjamin Lévy. “Visualising OMax”. Master II ATIAM. UPMC-IRCAM, 2009.

[God04] Jean-Brice Godet. *Grammaires de substitution harmonique dans un improvisateur automatique*. DEA ATIAM. Paris, 2004.

[Lau04] Cyril Laurier. *Attributs multiples dans un improvisateur automatique*. DEA ATIAM. Paris, 2004.

[Sel04] Carl Seleborg. *Interaction temps-réel/temps différé*. Mémoire de stage [DEA ATIAM]. 2004.

[Dur03] Nicolas Durand. “Apprentissage du style musical et interaction sur deux échelles temporelles”. DEA ATIAM. UPMC-IRCAM, 2003.

[Poi02] Emilie Poirson. “Simulation d’improvisations à l’aide d’un automate de facteurs et validation experimentale”. DEA ATIAM. UPMC-IRCAM, 2002.

[Lar00] Olivier Lartillot. *Modélisation du style musical par apprentissage statique : Une application de la théorie de l’information à la musique*. DEA Atiam. Paris, 2000.