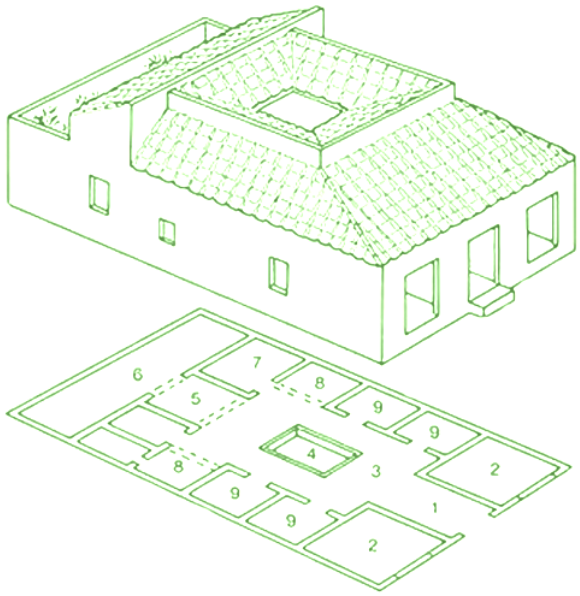
Proyecto DOMUS

IllaSynth

Mathieu Bosi, MMXI



Gràcia Territori Sonor

Centre comercial "Illa Diagonal"

Table of Contents

1 Introduction

L'Illa Diagonal (Fig. 1) is a commercial center located in Avinguda Diagonal, Barcelona, hosting several shops. While some of these shops put their own music inside of them (usually dance / pop style), there also are shared areas and avenues where the clients walk to go from a shop to the other. In these areas a large-scale sound diffusion system is being used. For this system the whole commercial center is divided into **8 zones**, each one located in a different part of the architectonic complex. This division into zones can be seen in Fig. 2.

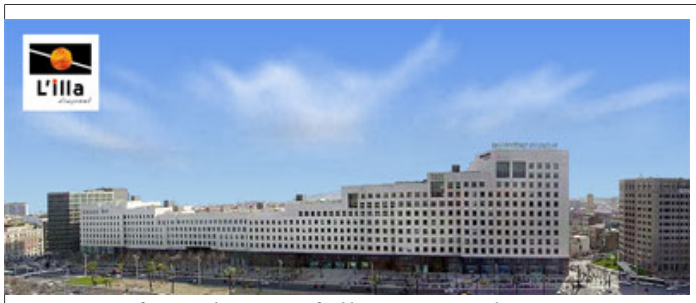


Fig. 1: A frontal view of Illa Diagonal.

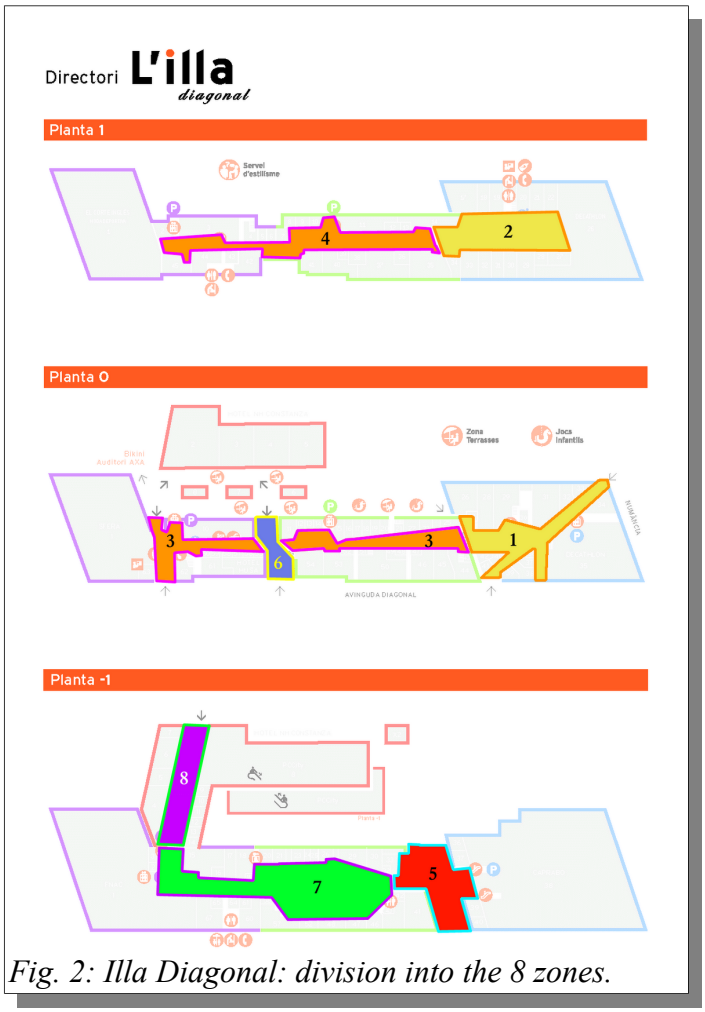


Fig. 2: Illa Diagonal: division into the 8 zones.

2 IllaSynth

The purpose of *IllaSynth* is to generates music and soundscapes for these 8 zones in real-time, offering a pleasant and non boring listening experience both to the visitors, and to the owners of the shops located in the areas and venues in each zone. For this purpose, the free and open source *Pure Data* computer music programming language has been chosen, in its extended version[1].

2.1 Technical considerations

During the various development stages, some facts were learned, principally about the constrains of a limited sound processing power and the limitations of pure generative sound synthesis. The following considerations are meant for whom may take on with the future development of the D.O.M.U.S. system.

2.1.1 About using generative synthesis

Computational resources can run out quickly when implementing pure generative sound synthesis.

Another caveat is this: for reality-like sounds, a sound synthesis model is generally not trivial to implement, and once implemented, it will always be bound to the set of parameters that define it, the very same bounds that make that sound belong to its category (e.g. a bird will not become a frog).

For imaginary sounds, the situation is a bit more free from the categories, however special attention is required in the definition of what the control parameters will be, and to enable the maximum possible variety with the minimum number of parameters. However, finding such a model and then finding the set of effective parameters can become a daunting task.

So, due to all this implications, it is good to have some purely generative sound synthesis models, however the best compromise is obtained by combining pure software synthesis with the sampled approach.

2.1.2 About using samples

While samples don't have the flexibility of a generative model, they also are much easier to obtain and use. By adequately combining all these samples, we can obtain a huge amount of combinations, for the price of just adding enough sound samples.

Sampled sounds can be custom-produced. On the other hand, a huge amount of recordings is freely available under the Creative Commons license on the www.freesound.org website. These recordings come from worldwide and can be searched by keywords / category. *IllaSynth* uses many sounds from the freesound database.

To support a big quantity / variety of sampled sounds, it would be desirable for sounds to be compressed into some way. For *IllaSynth*, the **Ogg Vorbis**[2] sound compression format has been chosen. Ogg Vorbis is an Open Source and royalty-free lossy compression format, superior to similar patented and non-free to use technologies like mp3 audio. Without using a compressed format, for example 1 hour of recorded mono audio at CD quality would occupy about 320 Mega Bytes (MB). By using the Vorbis compression format the same amount of sounds just takes 32 MB of disk space, which means a 1:10 compression ratio, this without noticeable degradation in sound quality for a non Hi-Fi system, like in our case. For comparison, in just 320 MB we can have 10 hours of sounds, in 1 GB

1 http://puredata.info/community/projects/software/pd-extended
2 http://en.wikipedia.org/wiki/Vorbis

about 31 hours, and so on.

2.1.3 About achieving independent zone-outputs

Due to limited CPU power it becomes necessary to share sound generation between the zones, at least partially. It is also necessary to implement an easily controllable and robust sound routing mechanism.

Moreover the CPU consumption should always be below the point in which audio output starts to get corrupted, or, in other words, the real-time constrains of the generative system are no longer met. For this reason an always all-on approach has been chosen in *IllaSynth*, to avoid any possible accidental breaking of the real-time constrain.

2.1.4 About the reliability of PureData objects and externals

While *PureData* is a wonderful and efficient tool to develop real-time audio applications, the following problems were encountered:

- the [switch~] object, useful for selectively allocating DSP computing resources, apparently tended to produce serious sound glitches (when switched off, a strong constant sound was sometimes caused). You won't find this object in *IllaSynth*.
- The possibility to use relative paths with [oggread~] possible only under Windows OS (partially resolved by using the [operating_system] object)
- OS dependent wildcards in [folder_list] (resolved using the [operating_system] object)
- A dangerous bug was found in the [freeverb~] reverberation object. This bug is insidious because it only happens under OSX or Linux. A workaround was implemented in the [rev~] wrapper-abstraction (see the abstraction for details).
- Dynamic signal routing was unusable due to bugs both in [send~] [receive~] and in the [send13~] [receive13~] externals.

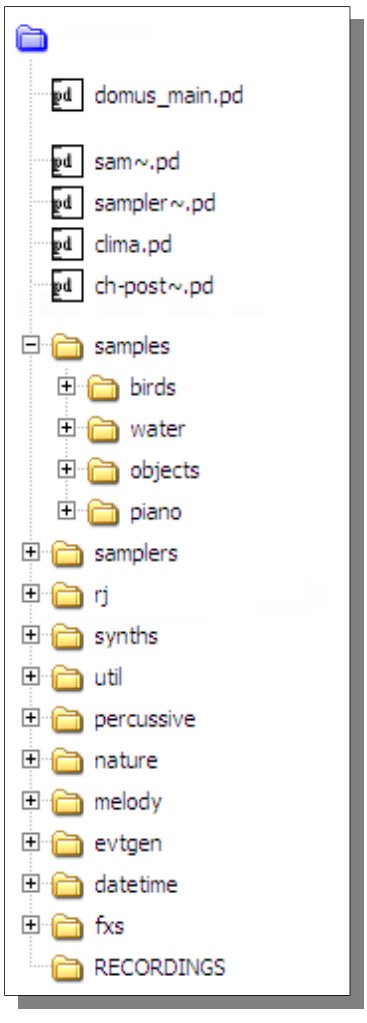
3 Software Overview

To better organize complexity, *IllaSynth* strongly follows *Pure Data* patches encapsulation principles by using a quantity of *sub-patches* and *abstractions*, *and* often using the graph-on-parent (GOP) functionality. In general, patches have been thoroughly commented and a clear layout and coding-style has been adopted to ease reading and understanding of the code. So, while the best documentation of the program is probably the code itself, an initial overview of the software will be now given, to facilitate the initial understanding of the implemented system architecture.

3.1 Directories layout

To better manage the various patches and resources, all files have been organized in folders.
At the **root level** we find the main patch `domus_main.pd` and some other abstractions that have been put at root level to access sound resources. These abstractions are:

- `clima` : reads the special files `temperaturaOK.tsv` and `humedadOK.tsv` provided by the underlying machine
- `sam~` sample player, accessing the samples/* folders
- `sampler~` used for reproducing sample sets stored in the `samplers` folder.



The folders are:

Sound resources folders

- **`samples`** various sound samples that have been divided into categories.
- **`samplers`** sets of sampled instruments organized by instrument name and MIDI note number
- **`RECORDINGS`** used to store the excerpts recorded from the main patch

PureData Abstractions folders

- **`rj`** rj library PureData abstractions
- **`synths`** various synthesizers abstractions
- **`util`** various utilities abstractions
- **`percussive`** percussive synthesizer abstractions
- **`nature`** nature sounds generators
- **`melody`** abstractions to generate melodies
- **`evtgen`** generation of events
- **`datetime`** utilities to handle date and time
- **`fxs`** sound processing effects

3.2 Adopted zone-routing mechanisms

To allow some differentiation between the zones while mantaining low CPU usage and a good control on the quantity of sounds, a lightweight combinatory approach has been chosen, after various tentatives.

The used routing system is based on:

- the [pan8~] abstraction, which sends a signal to a single zone with automatic cross-fades when the zone is changed. [pan8~] is normally used in combination with random counters and modulo offsets (see for instance Fig. 13 at page 19)
- zone audio buses: *zs1*, *zs2*, ..., *zs8* used to send signals to individual zones
- *zs_all,* used to send a signal simultaneously to all zones

3.3 Main Patch

The main patch is called `domus_main.pd` and is automatically loaded at system startup. The software can be explored starting from this patch which encloses all the needed sub-modules, as it can be seen in Fig. 3.

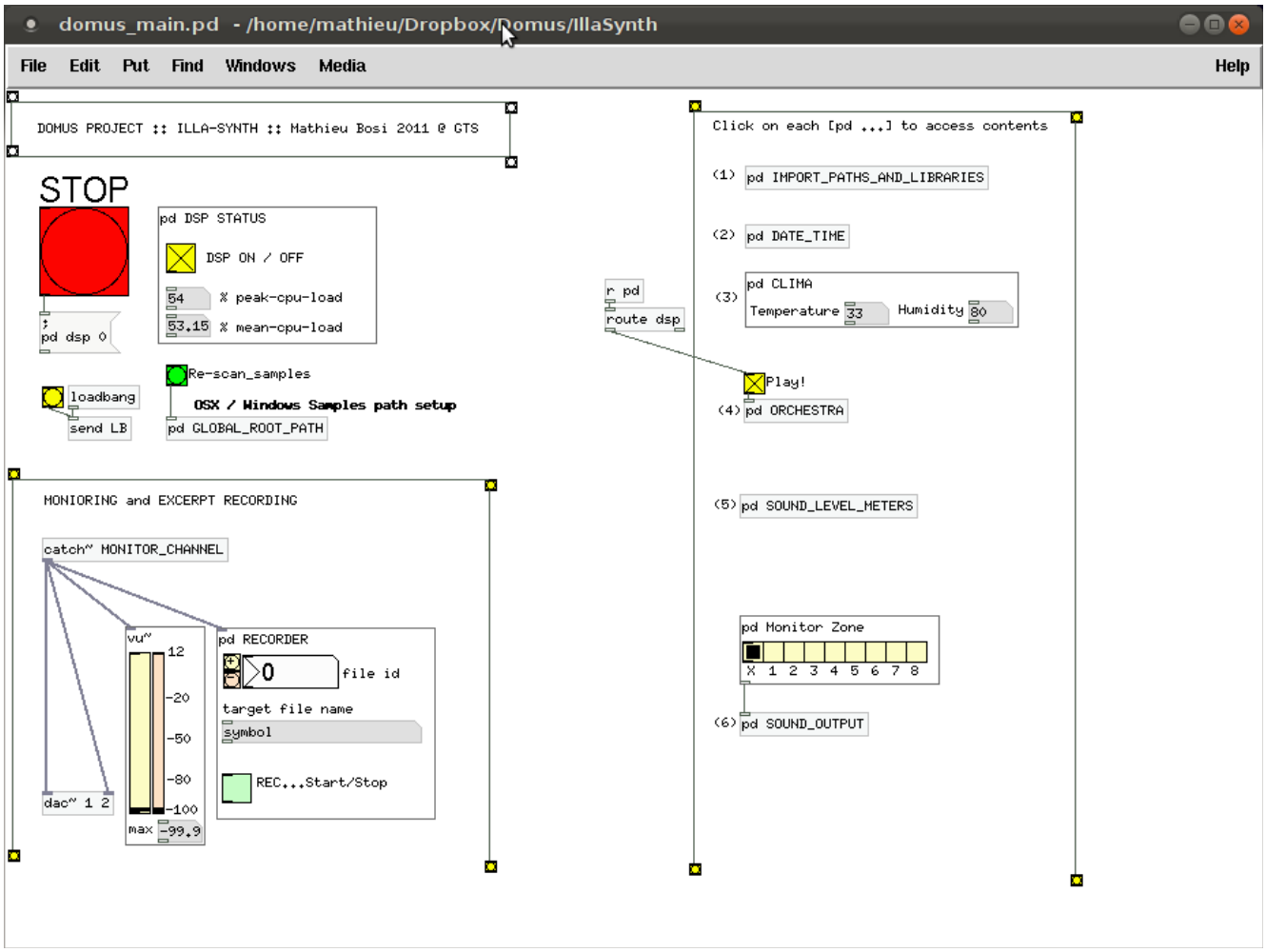


Fig. 3: the main patch encloses all the various sub-systems, like DSP handling, output monitoring/recording, date and time functions, and soundscape generators.

On the **left part** we can see:

- the **DSP handling** part (top)
  - DSP on/off toggle (yellow) and CPU usage statistics
  - STOP button (red) to immediately shut down the DSP

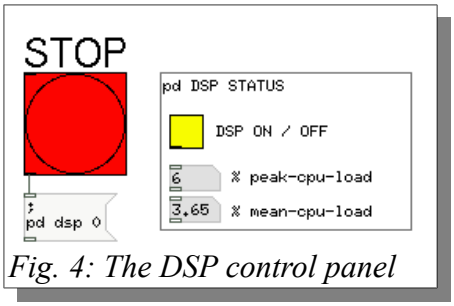


Fig. 4: The DSP control panel

- the **Monitoring and recording** part (bottom) composed of:
  - VU-meter showing the level of the monitored signal
  - a small GUI that allows to record 16-bits mono WAV files. Recordings are saved in the folder `RECORDINGS` with a name like name `rec_`*`nnnn`*`.wav` where the number *nnnn* can be adjusted with the small +/- buttons.

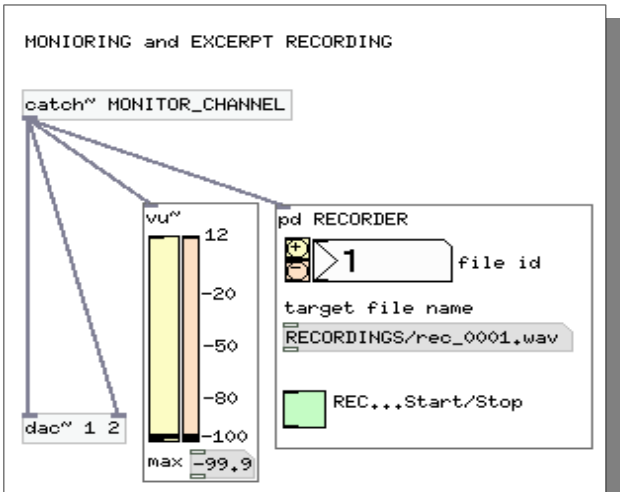


Fig. 5: The monitoring and recording GUI

On the **right part** of the main patch we find all the **core sub-systems** (numbered in the patch, see Fig. 6). Each one of them is briefly reviewed in the following sections.

> ***Hint***: to open a sub-patch or an abstraction, just click on its object-box when edit-mode is disabled. During edit-mode, just use the Ctrl + click combination. To open a *graph on parent* abstraction / sub-patch, you'll always have to right-click on its canavas and then choose the 'Open' option.

.

3.4 Sub-systems patches

Follows here a brief description of each one of the core sub-patches present in the main patch.

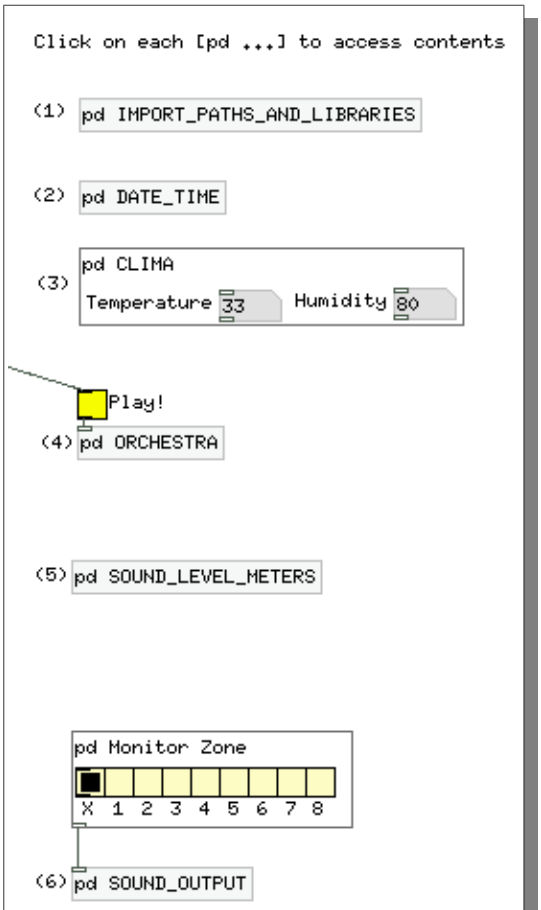


Fig. 6: The right part of the Main Patch

(1) Import libraries and declare custom abstractions paths

(2) Date and time messages generation

(3) Temperature and humidity values retrieval

(4) Sound generation units

(5) Sound level meters, one per zone

(6) Sound output system and zone monitoring options

3.4.1 Import libraries and declare custom abstractions paths (1)

This sub-patch encloses all the needed paths declarations in order to make the various abstractions accessible from anywhere inside the *IllaSynth* patches while retaining the ability to keep them organized into separate folders. For this purpose the [declare] object is used using the *-path* option (notice that the objects arguments coincide with the above described folder names).

Fig. 7: paths and libraries declarations

.

3.4.2 Date and time messages generation and broadcasting (2)

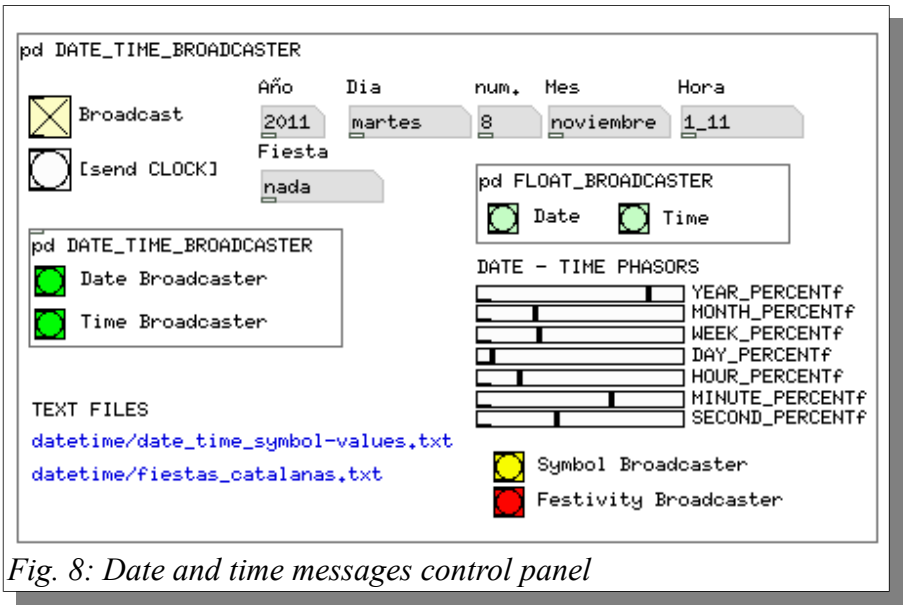


Fig. 8: Date and time messages control panel

This GOP sub-patch encloses all the date and time functionalities. Clicking on the various buttons (bangs) will bring up the enclosed sub-patch. For example, clicking on the "Festivity Broadcaster" button will pop-up the sub-patch responsible for broadcasting the festivities.

- Hyperlinks ([pddplink] objects, in the bottom left part) have been provided to the local files containing the festivities definition, in `datetime/fiestas_catalanas.txt`
- the list of broadcast messages, in `datetime/date_time_symbol-values.txt`.

Hint: in *IllaSynth* many messages are broadcasted by mean of the [sv *broadcast_address*] abstraction (short for **s**end+**v**alue). These messages can be accessed in two ways:

- using the [lv *broadcast_address*] abstraction (lv stands for **l**ist-**v**alue)
- using the standard PureData receive system, with [receive *broadcast_address*]

Festivities declaration

Festivities are declared in the `datetime/fiestas_catalanas.txt` text file. Each festivity occupies a line that contains 3 values: **month name**, **day of month** number, and **festivity name**. The current contents of the file are the following:

```
abril       23    san_jordi
junio       24    san_juan
septiembre  11    dia_nacional_cataluña
enero       1     año_nuevo
enero       6     los_reyes
mayo        1     dia_internacional_trabajo
agosto      15    asuncion_maria
octubre     12    fiesta_nacional_españa
noviembre   1     todos_los_santos
diciembre   6     constitucion_española
diciembre   8     inmaculada_concepcion
```

If the month and day match with any of the entries, the festivity name is automatically broadcasted. This behavior can be used for triggering special soundscapes.

Date and time broadcast symbols

Follows here a list of all the broadcast date/time messages. This reference list can be found in the `datetime/date_time_symbol-values.txt`.

Integer values (no postfix letter)

• YEAR	current year	e.g. 2011
• MONTH_YEAR	month of year	between 1 and 12
• DAY_YEAR	day in a year	between 1 and 365 (366 in leap years)
• DAY_MONTH	day in a month	between 1 and 28 to 31
• WEEK_YEAR	week of year	between 1 and 51
• DAY_WEEK	day of week	between 1 and 7
• DAYS_IN_MONTH	day of month	between 1 and 28-31
• HOURS_DAY	hour of day	between 0 and 24
• MINUTES_HOUR	minutes in hour	between 0 and 60
• SECONDS_MINUTE	seconds in minute	between 0 and 60
• MSECS	milliseconds	between 0 and 1000
• DAYLIGHT_SAVING	daylight-saving time	1 if daylight-saving, 0 otherwise

Floating point values (ending with 'f')

• MONTH_YEARf	month in a year	between 1 and 12
• DAY_YEARf	day in a year	between 1 and 365 (366 in leap years)
• DAY_MONTHf	day in a month	between 1 and 28 to 31
• YEAR_PERCENTf	year completion	between 0 (beginning) and 1 (end)

• MONTH_PERCENTf	month completion	between 0 (beginning) and 1 (end)
• YEARf	float year number	sum of YEAR and YEAR_PERCENTf
• DAY_WEEKf	float day of week	between 0 and 7
• WEEK_PERCENTf	week completion	between 0 (beginning) and 1 (end)
• DAY_PERCENTf	day completion	between 0 (0:00) and 1 (24:00)
• MINUTES_HOURf	minutes in hour	between 0 and 60
• HOUR_PERCENTf	hour completion	between 0 (hour beginning) and 1 (hour end)
• MINUTE_PERCENTf	minute completion	between 0 (beginning) and 1 (end)
• SECONDS_MINUTEf	seconds in a minute	between 0 and 60
• SECOND_PERCENTf	fraction of second	between 0 (beginning) and 1 (end), with millisecond resolution

Symbol values (ending with 's')

• YEAR+MONTH+DAYs	symbol-date	e.g. the symbol 2011_26_01
• MONTH+DAYs	symbol month-day	e.g. 12_30 (for december 30th)
• HOURS+MINUTESs	symbol-hour	e.g. 14_30 (for 14:30)
• MONTH_NAMEs	month name	month spanish name
• DAY_WEEK_NAMEs	day of week name	day spanish name
• FIESTAs	festivity name	one of the values in the 3rd column of `datetime/fiestas_catalanas.txt`

3.4.3 Temperature and humidity values retrieval (3)

This sub-patch periodically reads the humidity (in percentage) and temperature (in Celsius degrees) values. This is achieved by reading two special files named `temperaturaOK.tsv` and `humedadOK.tsv`. These files are not visible in the *IllaSynth* path but are anyway accessible. See the actual file-reading code inside the [clima] abstraction.
These values are broadcasted with the [sv] abstraction and can be used to influence the sound generation processes.

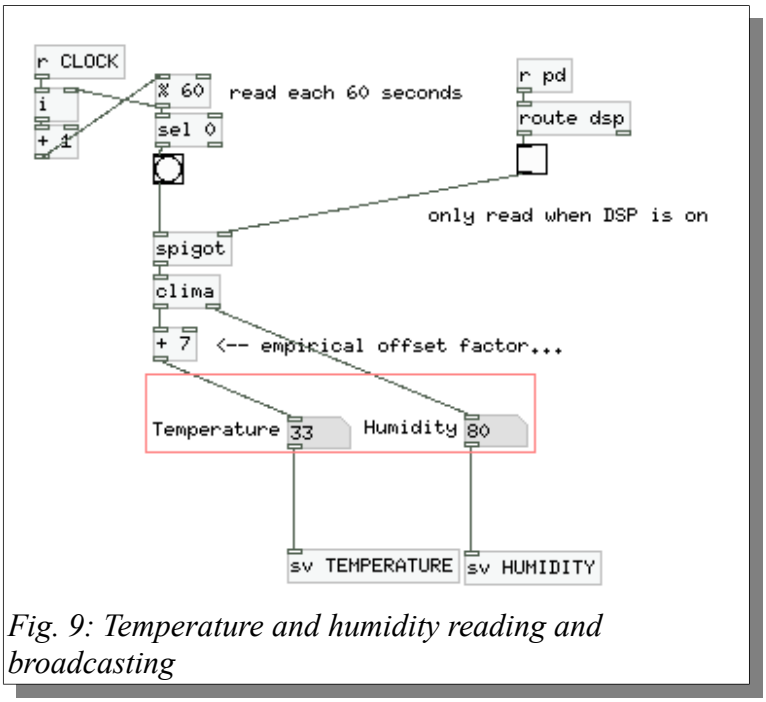


Fig. 9: Temperature and humidity reading and broadcasting

3.4.4 Sound generation units: “orchestra” (4)

This is the sub-patch where all the actual sound generation takes place, let's call it an “orchestra”. In this orchestra all the music and sound generating units can be divided into groups or categories, always following the encapsulation design principle. Each category will be now briefly described.
For details on each category's contents and implementations, please refer to their single patches.

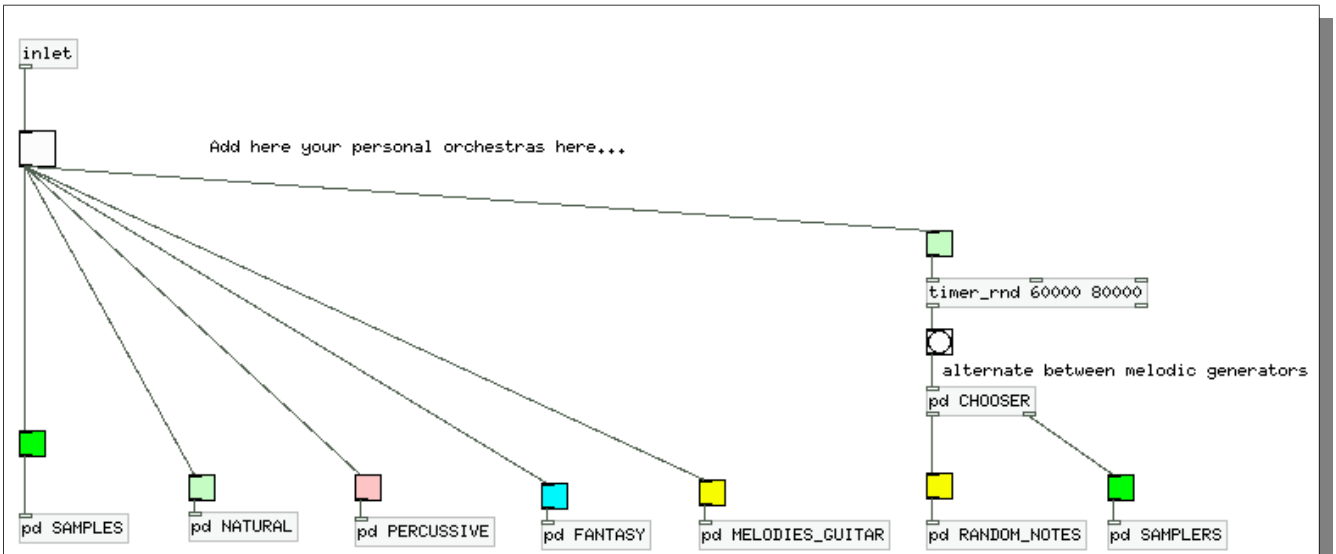


Fig. 10: The sounds and music generation "orchestra" is here

Samples

in this sub-patch there are various sample playback units. The samples playback units use the samples located into the `samples` folder. At the moment this folder contains the following folders-categories:

- **birds** contains various birds recordings
- **water** contains recordings of various water sounds
- **objects** recordings of various objects
- **piano** recordings of various short piano improvisation excerpts

These sounds have been taken from:

- the previous D.O.M.U.S. version
- the www.freesound.org website (all sounds are under the Creative Commons license)

Details on adding more samples can be found in section 4.2 (page 19).

Natural

Here various natural sounds synthesizers have been implemented, for instance cicadas, frogs, locusts, etc. The models have been adapted from those described in [1]. The various generators are activated depending on the time of day.

Percussive

Various synthesis models of percussive sounds have been implemented in this part of the orchestra, always following the techniques described in [1].

Fantasy

This part of the orchestra is devoted to the generation of imaginary sounds, for instance the underwater voices of whales.

Guitar melodies

In this sub-patch various guitar melodies are generated in various scales, rhythms and tonalities. The guitar sound is obtained by mean of a guitar sound synthesizer.

Random notes

Here some simple pseudo-random notes are generated, to increase the variety of the melodic content.

Samplers

Samplers are sets of samples containing the recording of various instruments for a range of MIDI note numbers. The existing samplers have been adapted from these already present in the previous version of D.O.M.U.S. For details on working with samplers, please refer to section 4.3 (page 21).

3.4.5 Sound output system and zone monitoring options (6)

In this sub-patch, all the signal buses sent from the orchestra are received, mixed, limited and output to the 8 channels of the sound card.

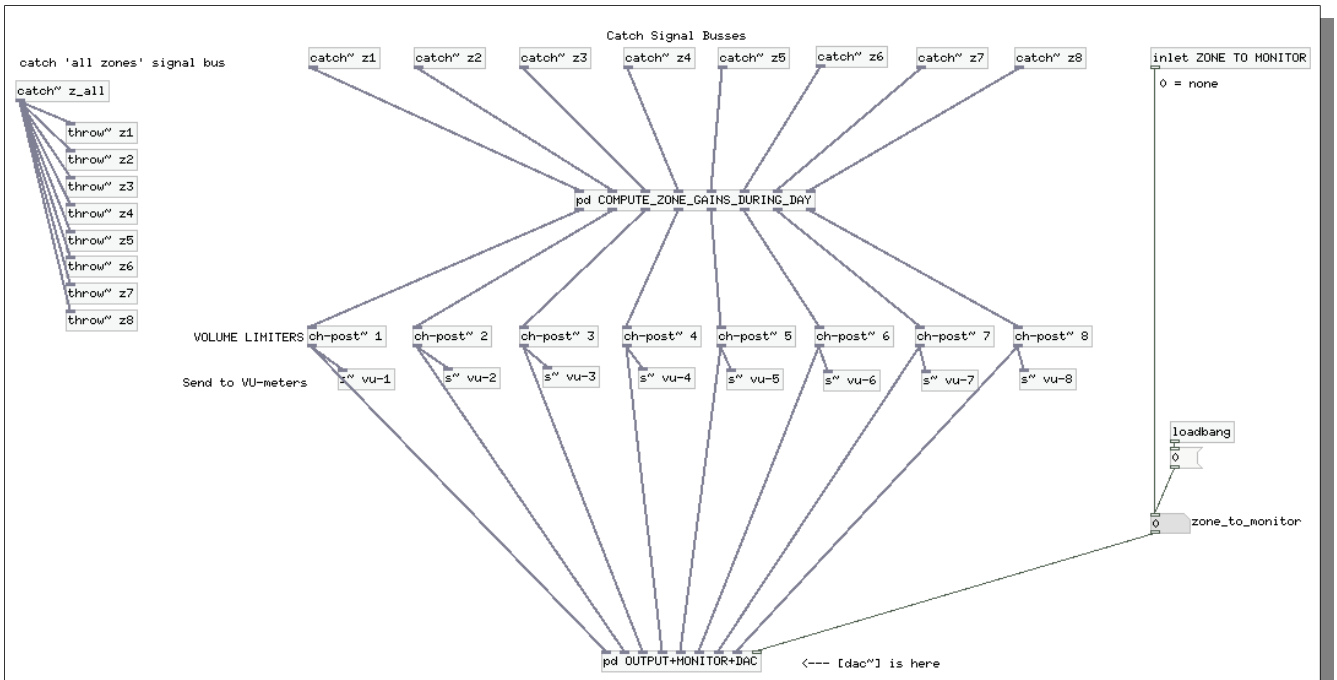


Fig. 11: The sound output system

From top to bottom, each bus/channel *(zs1, ..., zs8) is* in turn:

1. received with [catch~]
2. the gain of each channel is adjusted depending on the time of day (see the "Zone gains computing" paragraph below)
3. the volume is limited so that no saturation can happen
4. the signal level is sent to a visual monitoring stage (per channel VU-meter)
5. finally each channel is routed to its zone.

Always here, the all-channels bus *zs_all* is received and selective optional channel monitoring is performed (right part of the patch).

Zone gains computing

Depending on the amount of persons present in the commercial center, volume needed to maintain an audible sound level varies significantly. The gain factors for each channel ultimately depend on the time of day. This temporal gain information is enclosed in the sound output system sub-patch (see Fig. 11, right below the [catch~] objects).

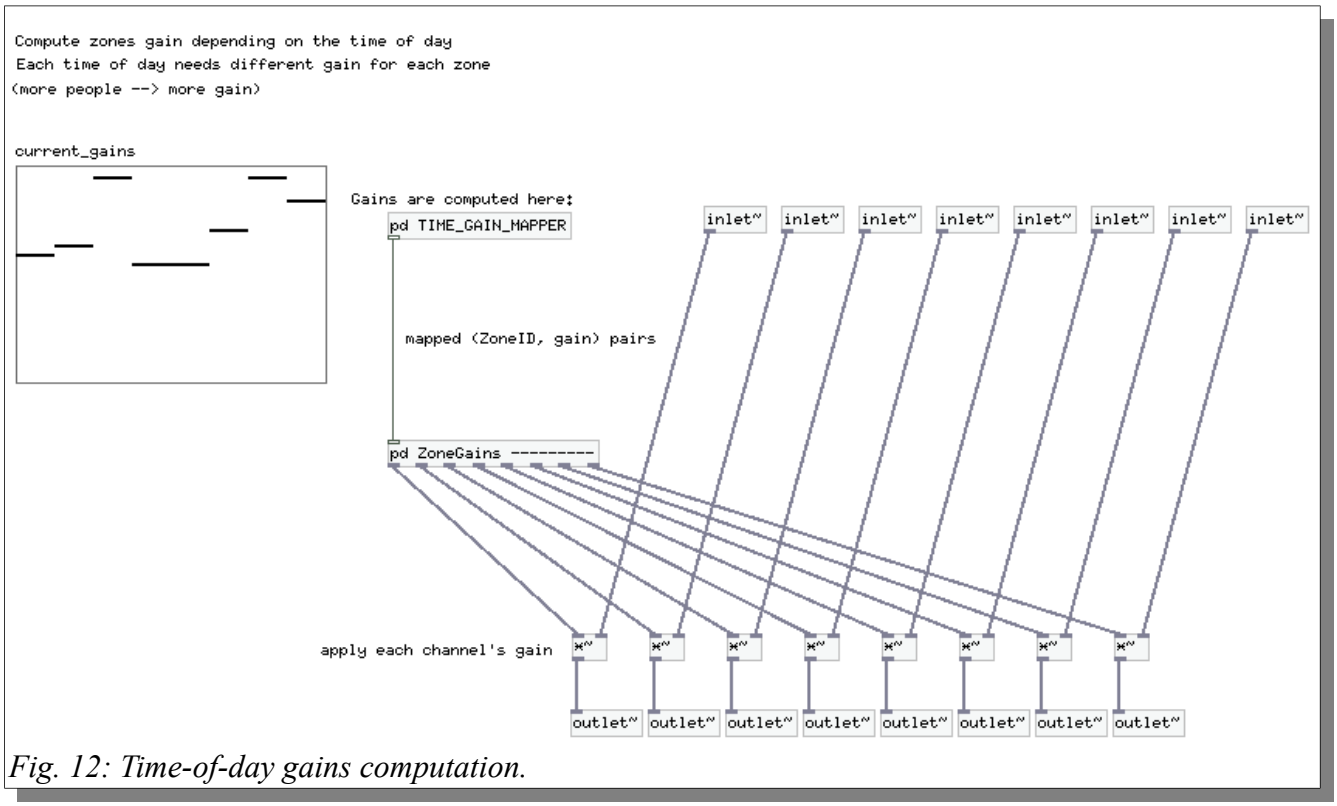
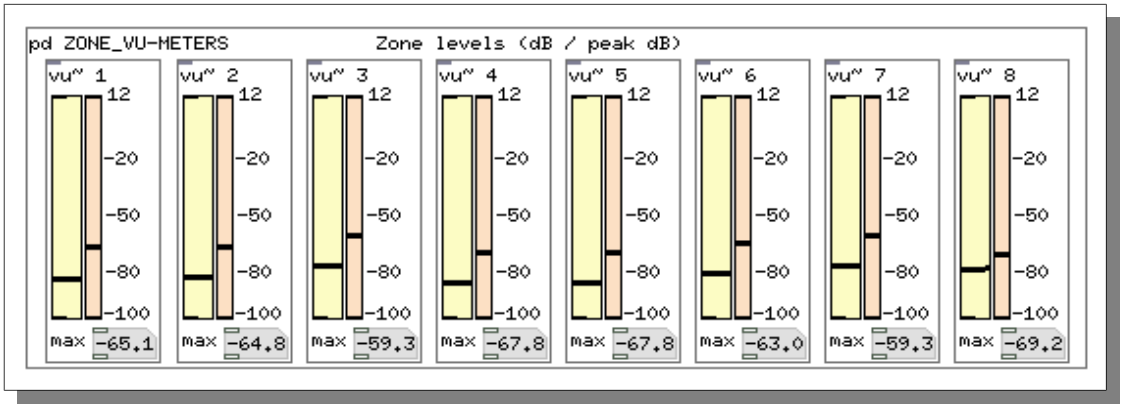


Fig. 12: Time-of-day gains computation.

3.4.6 Sound level meters, one per zone (5)

To monitor in software the output level of each channel, a VU-meter is provided for each zone. Sound intensities are measured in dB (where 0 dB corresponds to the maximum digital signal value of 1.0). [vu~] are implemented as custom GOP abstractions to increase the *PureData* GUI efficiency.



4 Sound-file resources in *IllaSynth*

IllaSynth has been designed to enable an easy integration of compressed sound samples sets, both in the format of normal individual sound samples and sample-banks, for sampled musical instruments.

4.1 Compressed sound format: Ogg Vorbis

Sampled sounds have an important role in *IllaSynth*. The Ogg Vorbis compression format has been chosen to alleviate disk space and access time restrictions for long samples.
All the sound files used in the system (both samples and samplers) have been encoded using these parameters: ***80 kbps, 44100 Hz, mono****,* which offer a compression ratio of about 1:10 without a meaningful loss of sound quality.

4.2 Use of the [sam~] abstraction with audio files

The [sam~] GOP abstraction is used to replay Ogg audio files. We can find an example of its use in the SAMPLES abstraction found in the “orchestra” sub-patch.

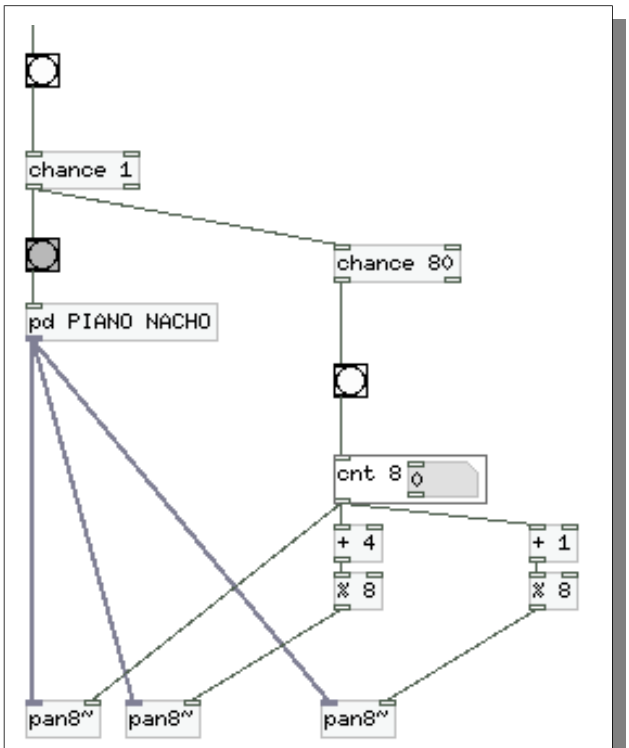


Fig. 13: The [sam~] abstractions are enclosed into a proper sub-patch (screenshot from [pd SAMPLES] in the "orchestra")

Fig. 14: The contents of the [pd NACHO PIANO] sub-patch in Fig. 13

In Fig. 14 We can see how the [sam~] abstraction can be used. Each time a bang is received, a random sample is played from a given folder. The abstraction's parameters are the path of the samples folder from where to pick the random file (relative to *IllaSynth* software root directory) and the duration of the fading time to apply at the beginning and end of each sample. The left output is the output signal and the right one sends a bang when playback of current file ends. This output is often used to generate a probabilistic chain triggering of more samples. The abstraction also gives a feedback on the file currently being played and the playback level and status.

4.2.1 Updating the various *samples* folders

To add more samples to an already existing samples folder it's only necessary to put the new files there. The [sam~] abstraction automatically accesses any new added file without the need of reloading the *IllaSynth* software.
In the case of adding a new folder/category, it will be necessary to:

1. add the folder and the sound samples, encoded in Ogg Vorbis format
2. add a new [sam~] instance(s) referring to the new folder.

The already existing [sam~] sub-patches (for instance [pd NACHO PIANO]) can be used as a template, or new configurations can be created.

4.3 Using [sampler~] and the polyphonic sampler [sampler4x~]

The [sampler~] and [sampler4x~] abstractions are based on [sam~] and can be used to automatically trigger the playback of musical instruments sample banks. These samplebanks are basically sets of files having as name:

`<instrument name>_<n>.ogg` (e.g. `PIANO_60.ogg`)

where <n> corresponds to the sample MIDI note number. The [sampler] abstraction accept the following initialization parameters:

- samplers folder where to look in (in our case the `samplers` folder)
- sampler <*instrument-name*> (e.g. PIANO or ARPA)
- valid MIDI note range (min, max)

The abstraction receives the MIDI note number to be played and reproduces the corresponding sample file. [sampler4x~] is just a wrapper around 4 instances of [sampler~] to enable a polyphony of 4 notes. The active instrument-name and range can be changed at run time. For more details on this, please refer to the [pd SAMPLERS] sub-patch (Fig. 15) in the "orchestra" sub-patch.

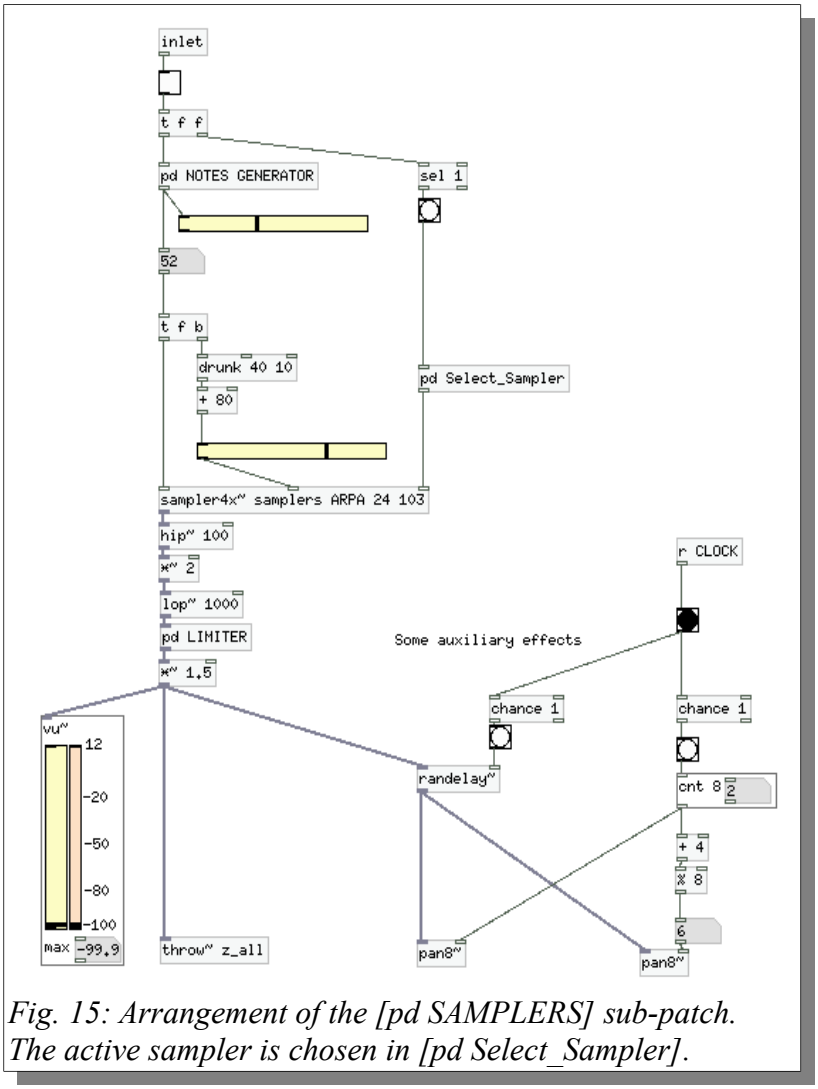


Fig. 15: Arrangement of the [pd SAMPLERS] sub-patch. The active sampler is chosen in [pd Select_Sampler].

4.3.1 Updating the *samplers* folder

New samplers can be put into this folder (or in another one), the only requirement for the samples names to be in the already mentioned way: `<instrument name>_<n>.ogg`.
After adding the new sampler's files, to add it to the pool of triggered samplers just edit the [pd Select_Sampler] sub-patch (Fig. 16).

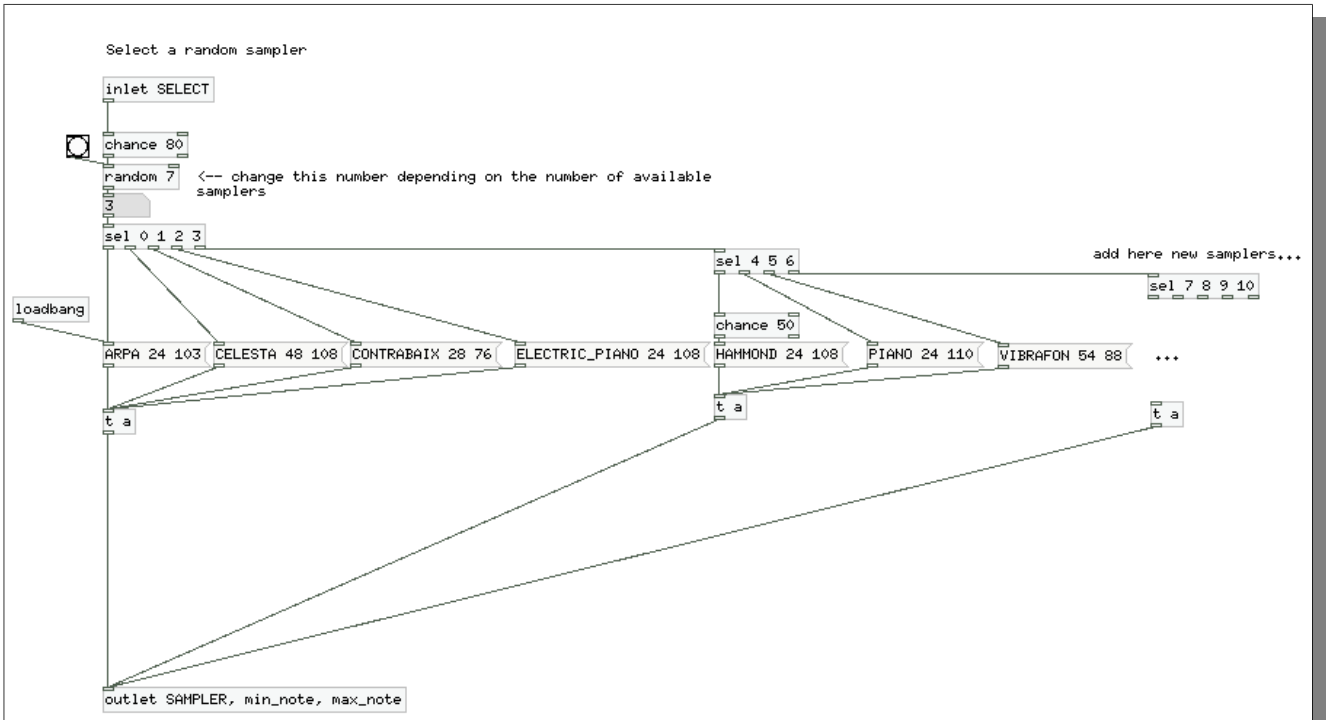


Fig. 16: The [pd Select_Sampler] sub-patch, from the [pd SAMPLERS] orchestra sub-patch.

References

[1] Andy Farnell, *Designing Sound - Practical synthetic sound design for film, games and interactive media using dataflow.* London: Applied Scientific Press Ltd, 2008

[2] www.projectedomus.blogspot.com (various references in the posts by Mathieu Bosi)