Camomile: Creating audio plugins with Pure Data

Pierre Guillot

Camomile: Creating audio plugins with Pure Data

Pierre GUILLOT
CICM – EA1572
University Paris 8
Saint-Denis, France
guillotpierre6@gmail.com

Abstract

Camomile is an audio plugin with Pure Data embedded for creating, with patches, original and cross-platform audio plugins that work with any digital audio workstation that supports VST or Audio Unit formats. This paper presents an overview of the current functionalities of Camomile and the possibilities offered by this tool. Following this presentation, the main lines of future development are exposed.

Keywords

Pure Data, Plugin, DAW, VST, Audio Unit

1 Introduction

Camomile[1] is a free, open-source and cross-platform audio plugin with Pure Data[2] [1] embedded, used to control patches inside a large set of digital audio workstations – as long as they support VST[3] or Audio Unit[4] formats. Development for this tool started in spring 2015 with a view to address issues that are related to pedagogical uses, experimental purposes and creation contexts. To satisfy these objectives, several approaches have been explored, resulting in many prototypes that have preceded the current version of the plugin. This entire endeavour, the many functional specifications that have been defined, the major issues that have been encountered – such as support for multiple instances and multithreading in Pure Data, and linking Pure Data with the plugin –, the different solutions that have been proposed and the choices that have been made are all presented in detailed in [2][5]. As most of the technical barriers have been broken down, the main goal of this project is currently to offer a tool that can compete with standard plugins. Hence, following an overview of the many features already offered by Camomile, the paper exposes the remaining work that is needed to complete this plugin, and the perspectives of development.

In practice, Camomile can be viewed as a meta-plugin: a plugin that generates other plugins. To clarify this presentation, the term "meta-plugin" will be used for this plugin – which embeds Pure Data; while the resulting plugins, containing the meta-plugin and patches, and can be used in digital audio workstations will simply be called "audio plugins". Thus, this presentation of Camomile is organised along two distinct but complementary axes. The first axis is focused on the creation of the audio plugin using the meta plugin: defining its functionality, creating patches, setting up features and so on. The second axis focuses on using the audio plugins: support by digital audio workstations, graphical interfaces and so on. Nevertheless, to offer a clear understanding of the defining aspects of each axis, this presentation is inverted. First, audio plugins usage is presented to highlight the features offered to the final user. Secondly, a large set of the features which can be implemented during the creation process will be shown. Following this

[1]The plugin is available in the VST2, VST3 and Audio Unit format for Linux, Windows and MacOS. The binaries and sources are available on the Github repository github.com/pierreguillot/camomile (accessed January 2018). Since the version 1.0.0, the sources are distributed under the license GNU GPLv3. The sources of the anterior versions are distributed under the licence BSD 3.

[2]Pure Data is a free and open-source software, created by Miller Puckette at the University of California, San Diego msp.ucsd.edu/software.html (accessed January 2018).

[3]The digital audio plugin format VST (Virtual Studio Technology) 2 et 3 are developed by the Steinberg GmbH company steinberg.net (accessed January 2018).

[4]The digital audio plugin format Audio Unit is developed by the Apple Inc. company developer.apple.com/audio (accessed January 2018).

[5]The publication also presents the context in which this project took place and in particular the related projects such as PdVST and PdLV2 but also the parallel projects like PdDroidParty and PdParty [4].

presentation, future developments of Camomile and its general perspectives will be exposed.

2 Using plugins

Before presenting the different features available to create audio plugins, it seems necessary to introduce what is ultimately an audio plugin created with Camomile and how this audio plugin appears to the user. Indeed, the architecture of the audio plugin generated with Camomile is a bit particular, and its approach favours sharing patches, abstractions and other documents to be used in conjunction with the meta-plugin to generate plugins, rather than directly sharing audio plugins – mainly because the audio plugins are associated with specific formats and operating systems while the original documents are free from these restrictions, so this approach gives users more freedom. So understanding the architecture of an audio plugin and the result in digital audio workstations will allow users to get a better grasp of the process of creating plugins.

2.1 Generating and loading plugins

The flexibility and the dynamic aspect of the Camomile approach makes it a tool noticeably different from standard plugins. Indeed, the binary files offered in the distribution and the meta-plugins, are not designed to be used directly within a digital audio workstation[6]. They must be used to set up the bond between the digital audio workstation and patches in order to generate the new audio plugins (see Figure 1).

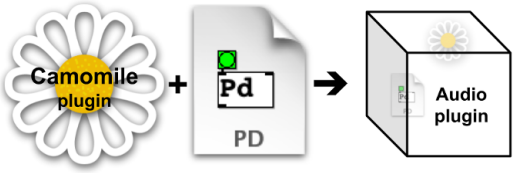

Figure 1: Schematic operation of the generation of an audio plugin from a Pure Data patch and the meta-plugin Camomile.

In practice, building an audio plugin with Camomile simply requires associating one of the meta-plugins provided by the distribution – according to the desired format and the type of plugin[7] – to a main patch and a set of additional and complementary contents – textual description of the audio plugin, abstractions, images and so on. Specifically, the operation consists in renaming the meta-plugin according to the main patch and also consists in respecting a certain hierarchy of the relative paths of the different files by creating a bundle[8]. Once this associated bundle is installed in the appropriate directory[9], the audio plugin is recognised by the digital audio workstations and is supported in the exact same way it would had been if compiled in a conventional manner (see Figure 2)[10]. The digital audio workstation then offers the ability to load one or more instances of the plugin and interacts with them in a conventional way with operations such as creating automations for the parameters, saving and recalling presets and so on.

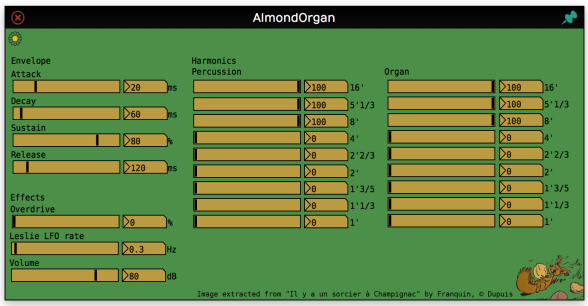

Figure 2: Representation of the graphical interface of the plugin *AlmondOrgan*, given as an example with Camomile, and generated from the patch.

2.2 User interfaces

Apart from the native representation of the digital audio workstation, which usually offers generic interfaces to represent and control parameters, this plugin has its own graphical interface. This window displays a representation of the main patch that potentially includes sliders, buttons, comments, or other user interface

[6]In practice, the digital audio workstation can nevertheless load the meta-plugins but without any patch, and so without any audio engine, they are useless.

[7]The distribution contains meta-plugin for each format – VST2, VST3 or Audio Unit – and each type of plugin – effect or instrument.

[8]On Linux and Windows, the meta-plugin and the patch must be placed in the same folder. On MacOS, all the files must be placed in the OS specific bundle of the plugin. These operations are presented step by step in the documentation of Camomile.

[9]The installation path of a plugin may depend on its format, the operating system or the preferences of the digital audio workstation. The documentation of Camomile helps to carry out this operation.

[10]This feature presented in detail in [2] ensures that the digital audio workstation manages each plugin created with Camomile independently, thus avoiding problems related to the management of presets or parameters but also to the sharing of projects and plugins.

components that are available in Pure Data (see Figure 2). This window makes it possible to represent the sound engine and interact with it, and also to communicate with the plugin. As will be shown later, the graphical user interfaces of the patch can be associated to parameters or specific actions like displaying a dialogue window to open or save files.

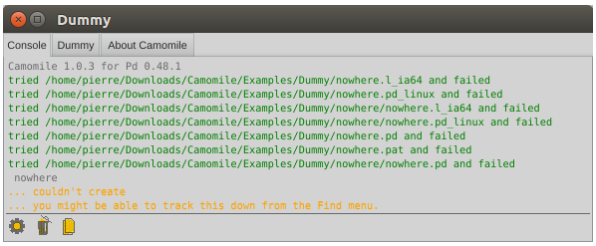

Figure 3: Auxiliary window of a plugin named Dummy illustrating the use of the console and the different types of messages.

In the upper-left corner of the interface, a button representing a chamomile flower is used to display an auxiliary window with three tabs (see Figure 3). The first tab corresponds to a console relatively similar to the one offered by Pure Data. This console receives the messages sent via the object print, the internal warnings of Pure Data – when an abstraction is not found for example – but also additional information related to the operation of the meta-plugin to facilitate debugging the patches. The console also allows you to copy, delete and filter messages according to their importance. The second tab displays information defined by the creator of the patch such as a description of the operations and how to use the plugin but also information related to credits or the plugin version. Finally, the last tab displays information related to Camomile, including legal information and credits related to different dependencies such as Pure Data, libPD[11] [3] and JUCE[12].

3 Creating plugins

Building a digital audio plugin with Camomile requires proper communication between the patch – the core of digital audio processing – and the digital audio workstation through the meta-plugin. For this purpose, Camomile offers several interfaces to use and handle a wide range of the usual features of digital audio plugins, such as parameters management, reading information from the play head, or creating the graphical user interface. These interfaces cover two aspects of plugin creation: properties definition for the plugin – such as its ability to handle MIDI events or the number and nature of its parameters – and communication between the patch and the digital audio workstation through the meta-plugin – so that the digital audio workstation or the plugin can interact with the patch and reciprocally the patch with the digital audio workstation – for example, to send and receive digital audio signals but also MIDI events, or to control parameters.

3.1 Plugin properties definition

To ensure optimal functioning within digital audio workstations, audio plugin properties are defined using a text file named after the meta-plugin and the main patch[13]. This properties file follows a syntax relatively similar to the FUDI[14] protocol where each line corresponds to a new statement and ends with a semicolon. So each statement can be used to define or to complete a feature or a property of the plugin. In order to ensure the proper functioning of the plugin, the console displays a warning if some properties have been wrongly defined, duplicated or omitted. Although in practice there is no hierarchy, these properties of the plugins can be organised according to categories.

First, properties are used to define general information, which is needed to generate the audio plugin and for it to function properly in digital audio workstations; such as the type of the plugin – to inform the user which meta-plugin to use for generating the plugin[15] – or the compatibility number – that corresponds to the version of the plugin with which the patch has been created and that is used to ensure compatibility with the patch[16].

[11]libpd is wrapper that turns Pure Data into an embeddable audio library libpd.cc (accessed February 2018).

[12]JUCE is an application programming interface oriented towards digital audio signal processing distributed by ROLI company juce.com (accessed January 2018).

[13]The documentation offers a full explanation on how to create and to use the properties file.

[14]FUDI is a network protocol invented by Miller Puckette for Pure Dat a en.wikipedia.org/wiki/fudi (accessed February 2018).

[15]The types can be effect or instrument and if the meta-plugin is not coherent with the type defined in the properties file, then the console displays a warning.

[16] If the version of the meta-plugin used is inferior to the compatibility version, then the console displays a warning.

Properties can also be used to activate extra functionalities that are originally deactivated for reasons of efficiency, for example if the audio plugin needs to handle MIDI events, play head information, or key event.

An important part of the options is focused on audio signal processing, like latency, which is implied by the plugin when using an FFT for example, or audio tail length – the time during which the output still produce audio after the input has been stopped – for reverberation effect for example. But the main audio property defines the audio buses supported by the plugin – the audio input and output configurations. The different audio plugin formats support dynamic audio buses layout, as well as multichannel and side-chains. Camomile offers a syntax that helps using these features. Thereby, an audio plugin can support several layouts of multichannel buses, for a sound spatialisation plugin for example, or the enabling or disabling of side-chains, for a compressor for example, so the process of the patch can be adapted depending on the buses layout submitted by the digital audio workstation[17] .

Another important aspect of an audio plugin is related to the control protocol of its state by the digital audio workstations using parameters. A parameter represents one or several aspect of the audio engine with a numerical value – that can be saved, restored, automated, etc. by the digital audio workstation. Camomile offers the possibility to create highly-developed parameters with names, labels, ranges of values, steps and so on to improve their use, their representation and their meaning.

At last, properties are used to define additional attributes which are not necessary for the proper functioning of the plugin, but which can be essential to its ease of use, such as the description displayed by the plugin in its tab on the auxiliary window, the reference to an image file that the plugin displays as background of the graphical interface or an option to automatically reload the patch when it has changed – useful during the creation process.

[17]All the audio buses layouts supported by the audio plugin must be defined at the first loading, so to support dynamic changes but also some specificities such as extra buses for side-chaining, this property must be pre-defined. More complex cases, like when the additional buses configurations depend on the main bus configuration, still need to be investigated. Furthermore, future versions could support a text description of the buses, like quadraphonic or ambisonic, to improve the specification of the configurations accepted by the plugin.

3.2 Communication between the plugin and the patch

Communication between the patch and the digital audio workstation through the meta-plugin is, for its part, ensured via a set of conventions and practices. First of all, the messages sent and received by the meta-plugin to and from the patch are synchronised sequentially to the audio thread depending on an order defined arbitrarily[18]. Overall, the meta-plugin first sends its messages, such as parameter values or MIDI events, then it processes the patch's digital audio chain, and finally it retrieves the messages sent from the patch to its address[19].

As defined by libpd, in a similar way to the applications PdParty or PdDroidParty, most of the communication can be handled within the patch using native objects: the objects *adc~* and the *dac~* for the audio signals[20], the objects *notein*, *noteout*, *ctlin*, *ctlout* and so on for the MIDI events and the objects *key*, *keyup* and *keyname* for the keyboard events. Furthermore, using a 'bus' receiver makes it possible to retrieve information about the current audio buses layout of the plugin when the audio starts – for example, to adapt the audio process. Using a 'play head' receiver during processing can be used to retrieve information such as tempo, time signature of the current bar, current position of the play head and so on, which could be indispensable for some synthesisers.

[18]Even if each Pure Data instance – each meta-plugin – can run in a separate thread, an instance can only be modified by only one thread, otherwise the behaviour is undefined and so potentially different from the one offered by the Pure Data application.

[19]The specific order of each message according to its type is fully explained in the documentation.

[20]In order to use directly the patch as an abstraction within the Pure data application, replacing the objects *adc~* and *dac~* by the objects *inlet~* and *outlet~* has been considered. Nevertheless, this solution didn't seem desirable because it prevents to receive or to send the audio signals from inside subpatches or abstractions and it makes more complicated the dynamic patching that could be useful to adapt the process to the audio buses layouts submitted by the digital audio workstations. Furthermore, the implementation of the meta-plugin becomes much more complex implementation especially to manage the audio block size in the main patch that would be no more necessarily predefined.

The patch is also used to define the plugin's graphical user interface. The bounds of the patch's visible area within the plugin interface is defined by the properties of the patch when using it as a graphical abstraction[21]. The graphical objects – such as the number box, the slider, the comment and so on – inside the area will be recreated by the plugin's interface and directly linked to their original object in such a way that no additional operation is necessary to communicate with the patch via the plugin (see Figure 4).

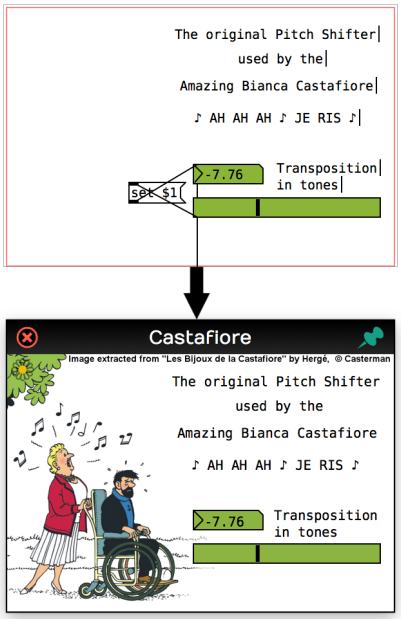

Figure 4: The part of the patch Castafiore that defined the graphical interface of the eponym plugin.

Inspired from the approach defined in PdParty, the plugin offers a replacement for the native Pure Data mechanisms such as the one offered by the objects *openpanel* and *savepanel* by displaying a dialog window to select files from the disk.[22]

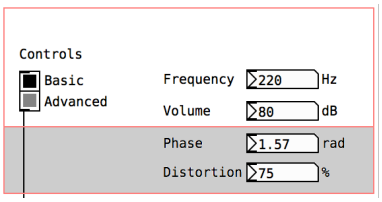

Figure 5: An example of a patch that illustrates the dynamic graphical interface of the plugin.

Moreover Camomile makes it possible to completely and dynamically redefine the graphical interface by changing its size, the objects and so on – a useful feature used to adapt the interface to the modes and requirements of the audio plugin (see Figure 5).

A specific aspect to the implementation of audio plugins is parameter management. Parameters values can be received using a 'param' receiver. But one could also want to modify the value of a parameter with the graphical interface – to record automations for example. This operation requires first to notify the digital audio workstation that the parameter will change, then to change the value – once or several times – and finally to notify the digital audio workstation that parameter modification has ended. If there are several graphical user interfaces and several parameters, these transactions become complicated to implement. Nevertheless the distribution offers a set of abstractions which can be directly connected to graphical objects to facilitate the setup of such mechanisms (see Figure 6).

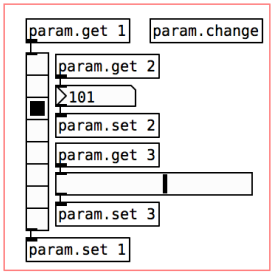

Figure 6: An example of a patch that shows how to link three graphical objects with three parameters. The abstractions *param.get* and *param.set* manage respectively the getting and the setting of each parameter and the abstraction *param.change* manages the global distribution of the messages within the patchs.

At last, when saving the digital audio workstation project or creating a new program, the meta-plugin automatically stores parameter states in an XML file. In addition, Camomile offers a mechanism to save and recall additional data via the patch that can be represented by parameters such as a system path or the content of an audio buffer.

With all these functionalities, Camomile offers the ability to create complex audio plugins with a large set of advanced features. Nevertheless, many developments can still be considered.

[21]The graphical abstractions are activated with the Graph-On-Parent option.

[22]A similar mechanism has been implemented to display the floating window of the object *array*. And the same feature is considered for the object *text*.

4 Perspectives

First of all, some native features of Pure Data relative to the graphical user interfaces are missing or can be improved, such as the implementation of the graphical object VU-meter or the improvement of the rendering of the graphical object labels. In order to get closer to standard plugins, it would be interesting to investigate the use of external images, which would replace drawing the graphical objects – using an image for the background of the object and one or more images for the foreground depending on the type of interface, it would be really easy to customise its representation. Another approach to offer more possibilities would be to implement the graphical part of the data structure of Pure Data [5], to draw and interact with more personal and original interfaces.

Support for external libraries is also very in demand by users. This feature could be a great improvement, this way someone could use an external as the audio processor of the plugin – optimizing the processes – and the patch as the interface with the meta-plugin and the digital audio workstation. Unfortunately, dynamic library loading seems to be restricted by the way Pure Data is embedded inside the meta-plugin[23] and by the fact that some of them are not directly compatible with multiple instance support[24]. Thus, direct integration of the most widespread libraries like the Cyclone [6][25] or the Zexy[26] libraries inside the plugin is considered. Nevertheless, this requires checking the compatibility of all objects and these dependencies could make Camomile difficult to maintain[27].

Offering a version of the plugin in the LV2[28] format is also considered, however the differences with the VST and Audio Units formats raise compatibility problems that still need to be explored.

5 Acknowledgements

The author would like to thank the whole community of Pure Data and libpd developers, especially Miller Puckette and Dan Wilcox, for their advice and explanations as well as the users of Camomile for their great feedback and suggestions. The author would like to also acknowledge the CICM and especially Alain Bonardi and Eliott Paris for their interest in the project, their comments and their advices.

References

[1] M. Puckette. 1997. Pure Data: Another Integrated Computer Music Environment *Proceedings of the Second Intercollege Computer Music Concerts,* p. 37-41, Tachikawa, Japan.

[2] P. Guillot. 2018. Camomile, Enjeux et Développements d'un Plugiciel Audio Embarquant Pure Data. *Actes des Journées d'Informatique Musicale*, Amiens, France.

[3] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth and H.-C Steiner. 2011. Embedding Pure Data with libpd. *Proceedings of the Pure Data Convention*, Weimar, Germany.

[4] D. Wilcox. 2016. PdParty: An iOS Computer Music Platform using libpd. *Proceedings of the Pure Data Convention*, New York, USA.

[5] M. Puckette. 2007. Using Pd as a score language. *Proceedings of the International Computer Music Conference*, p. 184-187, Göteborg, Sweden.

[6] A. Torres Porres, D. Kwan and M. Barber. 2016. Cloning Max/MSP Objects: A Proposal for the Upgrade of Cyclone. *Proceedings of the Pure Data Convention*, New York, USA.

[7] I. I. Bukvic, A. Gräf and J. Wilkes. 2017. Meet the Cat: Pd-L2Ork and its New Cross-Platform Version "Purr Data". *Proceedings of the Linux Audio Conference*, Saint-Étienne, France.

[23] The reason of this restriction still need to be investigated.

[24] If a library goes beyond the 'public' API of Pure Data and uses internal structures that deal with the multiple instance support, some problems may occur.

[25] github.com/porres/pd-cyclone (accessed March 2018).

[26] The Zexy library is developed by IOhannes m zmölnig puredata.info/downloads/zexy (accessed March 2018).

[27] Using a monolithic approach by including the libraries [Bukvic & al., 2017] is one of the causes of the abandonment of the Pure Data variant, Pd-extended, puredata.info/downloads/pd-extended (accessed January 2018) originally maintained Hans Christoph Steiner.

[28] The LV2 format by D. Robillard is the successor of the LADSPA plugin format lv2plug.in/ns (accessed March 2018).