aAaA: an attribute aware abstraction architecture allowing arbitrary argument assignment in Pure Data

José Henrique Padovani[1*]

[1]NICS - Interdisciplinary Nucleus of Sound Communication / State University Of Campinas (UNICAMP)
Music Department / Arts Institute / State University Of Campinas (UNICAMP)
R. Elis Regina, 50 – Cidade Universitária – Campinas, SP – CEP 13083-854

josep@unicamp.br

Abstract

We describe `aAaA`, an abstraction-based extension for Pure Data (Pd) that parses any number of attributes (`@`-initiated symbols) and their associated arguments and routes the entered values to unique labeled receivers. This approach expands the syntax of Pd without the need of compiled libraries, objects and extensions – a feature that can be useful in contexts in which Pd abstractions are embedded in applications, mobile and similar architectures.

1. Introduction

In the last decades Pure Data (Pd) has become a major music and audio programming environment, being extensively used in artistic contexts, pedagogical situations, entertainment products and research projects. Due to its intuitive graphical coding approach (consisting of interconnecting objects, messages and other data structures to route symbolic control informations and digital audio signals) and also to its multi-platform free and open source software licensing, compatibility and distribution, Pd has spread rapidly as a flexible and powerful tool to process and synthesize sounds in real-time [1].

Recently, it also became a very convenient audio framework, being possible to be embedded in other applications and run not only in conventional desktop and laptop computers but, also, in mobile platforms and single-board computers[2, 3]. In these environments, it is usually easier to run Pd "patches" that do not require any compiled extensions. If this is sometimes seen as a limitation – specially taking into account that the increasing use of Pd was largely due to the now discontinued "Pd-extended" fork, that was distributed with a large set of compiled extensions –, it is possible to make Pd abstractions that reproduce the features provided by these extended objects and functions. This can be done with coding/'patching' strategies such as *dynamical patching*, without causing any significant loss of performance[1].

Taking this context into account, this paper presents an abstraction extension entirely created with Pd vanilla resources that expands the argument syntax of Pd abstractions entirely written as patches that use only usual vanilla objects. Developed to be run in Pd vanilla, `aAaA` can be seen as a template abstraction that emulates some of the Max[2] features related to how it deals with objects and their arguments. In some Max objects, it is possible to assign arguments by entering one or more attributes and related values. A Max `[cycle~]` oscillator object, for instance, can be created with arguments such as `@phase`, `@frequency`, `@buffer` and other attribute tags followed by their respective values (a handy syntax feature that makes patching

*Supported by CNPq [Edital Universal 14/2014] and FAEPEX/UNICAMP

[1]Until few years ago, Pd was distributed in two major versions. Pd "vanilla" is the main branch of development, being mostly carried by Miller Puckette with few core objects. The "extended" fork was distributed in a larger package with many compiled extensions and objects. It was mainly maintained by Hans-Christoph Steiner and is now not supported/developed any more. To compensate this, Pd-vanilla has, since the version 0.47.0, its own extensions install manager (deken). It is also worth to mention a new cross-platform Pd branch: Purr Data/Pd-l2ork, which comes with many pre-compiled extensions and features a modern HTML5 based GUI. More information about Pd "flavors" can be reached at `https://puredata.info/`

[2] `http://cycling74.com/products/max`.

more user-friendly/mnemonic and makes possible to enter only the relevant non-default values desired in a specific context)[4].

While similar functionalities have been already implemented in compiled objects and libraries in Pd (as it happens, for example, with the library `cyclone`, designed to clone many Max features in Pd)[5] and while other Pd compiled objects use "flags" to set specific internal parameters and values, `aAaA` implements this same kind of syntax recognition inside Pd by taking advantage of dynamical patching processes.

2. Pd abstractions and arguments

A parsing process that iterates given attributes and arguments of an abstraction is not trivial to be constructed with Pd patches. Indeed, as Pd is a graphical language, processes like iterative loops or parsing processes are constructed in a very different way than similar processes programmed in functional or object-oriented code based languages. Particularly regarding argument parsing, if it is desired that some abstraction uses arguments that are informed in the moment of the object instantiation, it is necessary, in order to use these values to some purpose, to know exactly how many arguments will be used and in which order they will be informed. In this way, one can use Pd objects and *dollarsign* arguments to retrieve the values informed by the user/developer when he instantiates the object. If the abstraction patch has a `[float $1]` and a `[symbol $2]` inside, these two objects will store respectively a number and a string that can be then used internally to change parameters of the algorithm.

Briefly, the `aAaA` algorithm parses an arbitrary number of symbols and floats given as the objects arguments, and groups them in lists for each *attribute* – a pseudo-type that consist in a symbol/string that begins with the `@` character. These lists are used internally to route the attribute-nestled parameters to the appropriate receivers (`[receive]`/`[r]` objects). Arguments given before any `@`-initiated symbol are internally grouped in a single list to be sent to its respective receiver object, being possible to split them in individual symbols, *unpack* or subject them to other parsing/processing mechanisms.

In the following sections, further details are giving explaining each step of the algorithm and exemplifying the application of `aAaA` in a simple synthesis abstraction based object.

3. aAaA: any Arguments and Attributes

Internally, `aAaA` has a 'subpatch' (a container where a sub-process runs) named `[pd aAaA-kernel]` (Fig. 1). This subpatch encapsulates the mechanism that parses arguments such as floats, symbols and attributes and sends them to the appropriate `[receive]` objects that may used those values if they are informed by the user/developer.

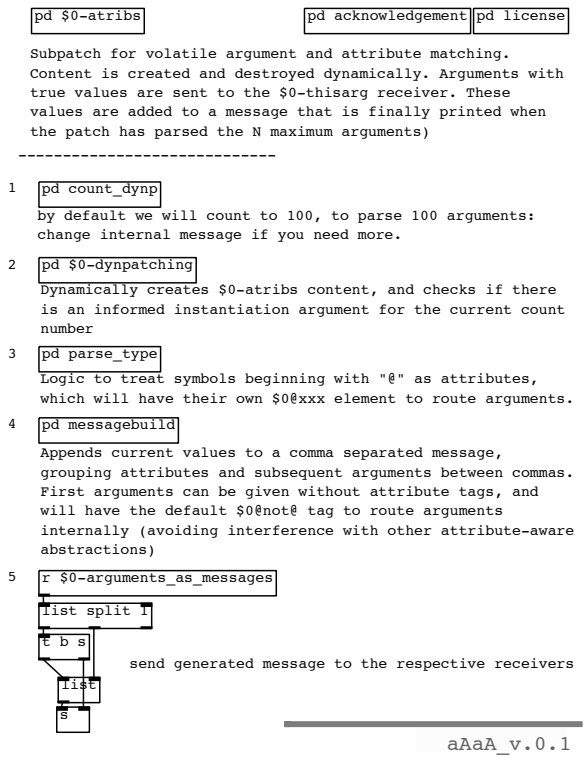

Figure 1: [pd aAaA-kernel] subpatch content

The `[pd aAaA-kernel]` 'subpatch' is designed be copied to any Pd abstraction being written, enabling it to parse attributes/arguments internally.

`[pd $0-atribs]` (Fig. 2) is a blank container whose content will be iteratively populated and destroyed with dynamically created objects.

This is done to retrieve each of the arguments given in the instantiation of the `aAaA`-enabled abstraction, according to a counter that is driven in the `[pd count_dynp]` sub-patch (fig. 2). This patch controls the main counter, and send ordered messages to initiate the whole process, to execute the dynamical patching mechanism and to inform which is the current argument index from 1 to N (that has the default value of 100, which can be changed for specific situations).

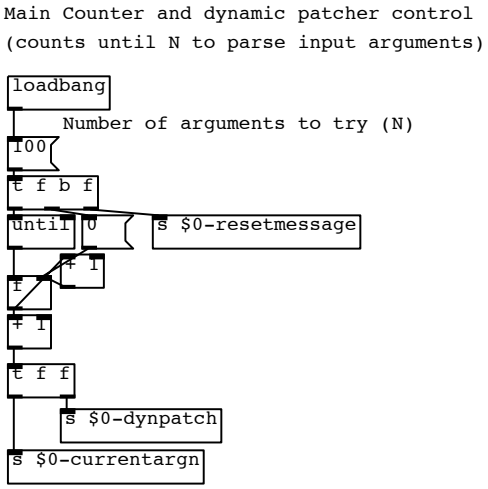

Figure 2: [pd count_dynp] content

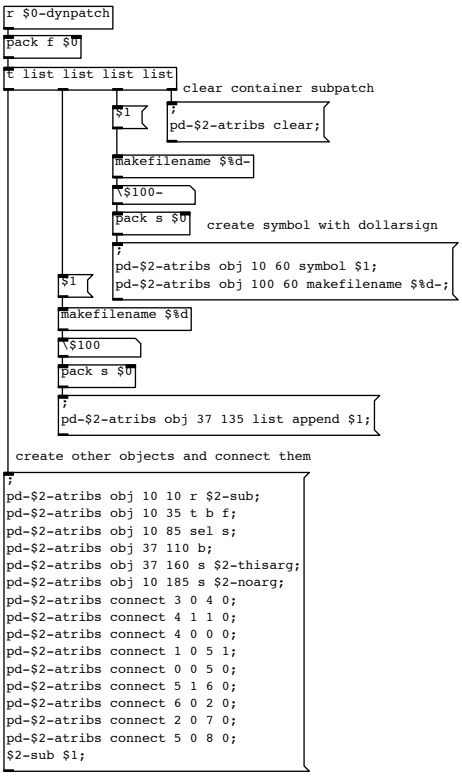

Figure 3: [pd $0-dynpatching] content

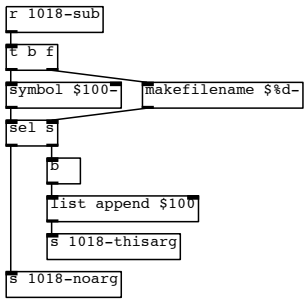

Figure 4: The dynamically created [pd $0-atribs] subpatch content at the moment that the counter has reached the number 100

The `[pd $0-dynpatching]` (fig. 3) process creates and deletes dynamically the content of the abstraction `[pd $0-atribs]` (fig. 4) and retrieves the argument in the current index informed by the main counter, checking if it really was informed in the moment of object instantiation.

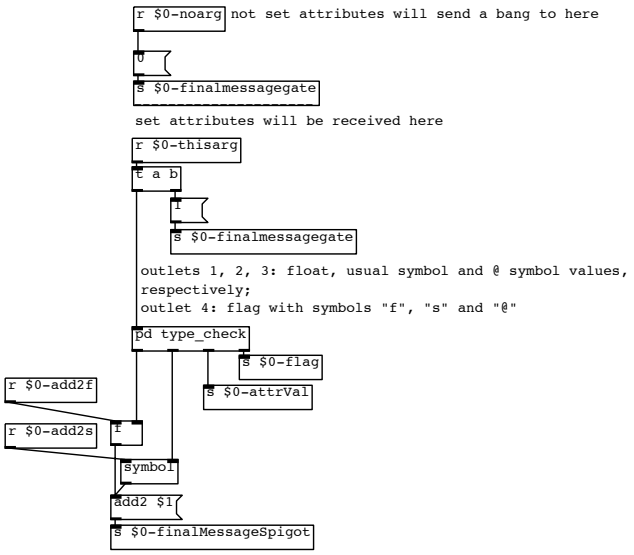

Figure 5: The [pd parse_type] content.

The `[pd parse_type]` (fig. 5) sub-patch checks if the argument is a floating point number (`f`), a usual symbol/string (`s`) or an attribute (`@`) – i.e., a symbol that begins with the char `@`. The flags `f`, `s` and `@` are sent to the following processes, being used to organize a growing list of comma separated messages that will be sent at the end of the parsing process to local receivers named according to the attribute tags.

The [pd messagebuild] (fig. 6) process organizes a big comma separated list, that will route attribute-grouped arguments together, sending then to a final receiver – see stage 5. in the fig. 1. This final receiver routes the individual messages to specific receivers that will be named according to the attributes, preceded by the unique abstraction wildcard $0, which is instantiated locally for each abstraction in Pd. This enables the use of multiple aAaA-enabled abstractions side-by-side, and allows to create scalable, flexible, and structured objects capable of dealing with the instantiation arguments in a very intuitive way.

Figure 6: The [pd messagebuild] sub-patch content.

4. Using aAaA: an example

aAaA is distributed with two abstractions. aAaA-example.pd (fig. 7) uses the described [pd aAaA-kernel] (fig. 1) mechanisms to create a table lookup synthesis process with two amplitude modulation oscillators; and aAaA-help.pd which instantiates the previous abstraction with a series of attributes and arguments that are internally assigned to the respective objects and data structures that are used in the synthesis process.

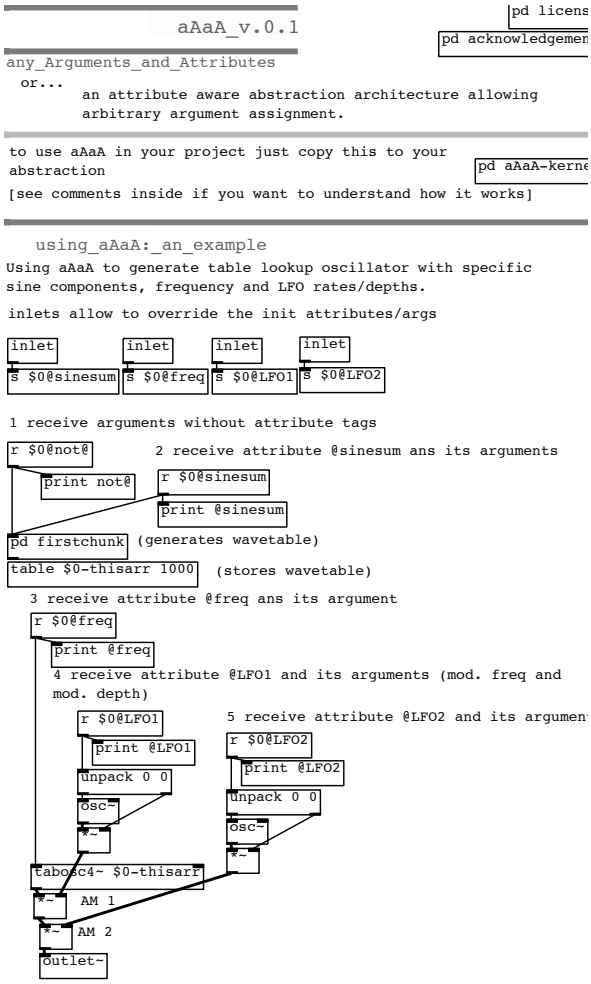

Figure 7: The aAaA-example.pd abstraction, using the aAaA kernel to parse parameters of a table lookup synthesis process.

4.1. aAaA-example.pd

The aAaA-example.pd (fig. 7) abstraction uses the messages sent by the [pd aAaA-kernel to control different parameters of the synthesis algorithm. The first attribute is @sinesum, with arguments that are also programmed to be received without any attribute tag. These arguments are internally received by objects [r $0@sinesum] and [r $0@not@] (a special tag used for arguments without any attribute specification). They are used in the sub-patch [pd firstchunk] to populate the table $0-thisarr$, that will

be continuously read by a quadratic interpolating table oscillator (`[tabosc4~]`) with a frequency given by a floating number preceded by the `@freq` attribute tag (and that will be routed to the `[r $0@freq]` object).

The attributes `@LFO1` and `@LFO2` expect, each one, two floating point arguments that control, respectively, the modulation frequency and the modulation depth of two amplitude modulation processes. These values are received by the `[r $0@LFO1]` and the `[r $0@LFO2]` objects, being unpack and appropriately routed internally to drive the modulation oscillators.

4.2. aAaA-help.pd

The `aAaA-help.pd` abstraction (fig. 1) exemplifies how objects created with the `aAaA-kernel` attribute and argument parsing mechanism can take advantage of the `aAaA` attribute syntax.

Given that the `aAaA-example.pd` and the `aAaA-kernel` processes use `$0` wildcards – which corresponde to individual numbers that identify each abstraction instance – to name objects, tables and data structures, `aAaA`-enabled abstractions don't interfere in each other. In other words, if two objects share the same attribute tags names (in the given example, for instance, the attribute `@LFO1`), only the internal receiver identified by the individual number of `@0` followed by the attribute symbol `@LFO1` (thus, the receiver `$0@LFO1`) will receive the arguments intended to be used internally.

5. Conclusion

`aAaA` is capable of expanding the coding syntax of Pd without requiring any special libraries and extensions: a feature that allows, among other things, that `aAaA`-enabled abstractions to be easily embedded in applications and hardwares that use `libpd` as an audio processing/synthesis framework. As the dynamical process of parsing attributes and arguments occurs in the instantiation stage of the abstractions, the use of `aAaA` does not imply in any loss of performance.

The same strategy to route `@` initiated symbols in `aAaA` could be used to recognize other pseudo-types. It would be possible, for instance, to use the commonly used hyphen symbol (−) to designate flags that should be treated in a special way. Likewise, one could designate a set of symbols to instantiate special data structures like matrices or trees.

`aAaA` is part of a larger set of abstractions, objects and resources currently being developed in the Interdisciplinary Nucleus of Sound Communication (NICS). This set of resources are aimed to expand the coding possibilities of Pd and are designed to be easily adapted to specific circumstances and to be able to be embedded in heterogeneous platforms, applications, hardwares and operational systems. In a more general perspective, the objective is to develop resources to design structured abstractions whose mechanisms and general behavior can be specified in a more structured way, just like code based computer music languages usually feature.

6. Acknowledgments

`aAaA` is based in previous abstractions and strategies developed by IOhannes m zmölnig that were distributed in the Pd mail-list[3].

References

[1] Miller Puckette. Etude de cas sur les logiciels pour artistes: Max/MSP et Pure Data. In David-Olivier Lartigaud, editor, *Art++*. HYX, Orléans, July 2011.

[2] Peter Brinkmann, Dan Wilcox, Tal Kirshboim, Richard Eakin, and Ryan Alexander. Libpd: Past, Present, and Future of Embedding Pure Data. *PdCon2016~*, 2016.

[3] Peter Brinkmann. *Making Musical Apps: Real-Time Audio Synthesis on Android and iOS*. O'Reilly Media, Sebastopol, Calif, 1 edition edition, March 2012.

[3] Relevant discussions can be found at the following archive links: `https://lists.puredata.info/pipermail/pd-list/2008-10/065465.html` and `https://lists.puredata.info/pipermail/pd-list/2016-08/115936.html`.

[4] MSP reference - cycle~ - sinusoidal oscillator. https://docs.cycling74.com/max7/maxobject/cycle~.

[5] Alexandre Torres Porres, Derek Kwan, and Mathew Barber. Cloning Max/MSP Objects: A Proposal for the Upgrade of Cyclone. *Pd-Con2016~*, 2016.