Meet the Cat: Pd-L2Ork and its New Cross-Platform Version "Purr Data"

Ivica Ico Bukvic
Virginia Tech SOPA ICAT
DISIS L2Ork
Blacksburg, VA, USA 24061
ico@vt.edu

Albert Gräf
Johannes Gutenberg
University (JGU)
IKM, Music-Informatics
Mainz, Germany
aggraef@gmail.com

Jonathan Wilkes
jon.w.wilkes@gmail.com

Abstract

The paper reports on the latest developments of Pd-L2Ork, a fork of Pd-extended created by Ico Bukvic in 2010 for the Linux Laptop Orchestra (L2Ork). Pd-L2Ork offers many usability improvements and a growing set of objects designed to lower the learning curve and facilitate rapid prototyping. Started in 2015 by Jonathan Wilkes, Purr Data is a cross-platform port of Pd-L2Ork which has recently been released as Pd-L2Ork version 2. It features a complete GUI rewrite and Mac/Windows support, leveraging JavaScript and Node-Webkit as a replacement for Pd's aging Tcl/Tk-based GUI component.

Keywords

Pd-L2Ork, Purr Data, fork, usability, L2Ork

1 Introduction

Pure Data, also known as Pd, [15] is arguably one of the most widespread audio and multimedia dataflow programming languages. Pd's history is deeply intertwined with that of its commercial counterpart, Cycling 74's Max [16]. A particular strength shared by the two platforms is in their modularized approach that empowers third party developers to extend the functionality without having to deal with the underlying engine. Perhaps the most profound impact of Pd is in its completely free and open source model that has enabled it to thrive in a number of environments inaccessible to its commercial counterpart. Examples include custom in-house solutions for entertainment software (e.g. EaPd [10]), Unity3D [18] and smartphone integration via libPD [1], an embeddable library (e.g. RjDj [11], PdDroidParty [12], and Mobmuplat [9]), and other embedded platforms, such as Raspberry Pi [4].

Pd's author Miller Puckette has spearheaded a steady development pace with the primary motivation being iterative improvement while preserving backwards compatibility. Puckette's work on Pd continues to be instrumental in fostering creativity and curiosity across generations, and as the library of works relying on Pd grows, so does the importance of conservation and ensuring that Pd continues to support even the oldest of patches. However, the inevitable side-effect of the increasingly conservationist focus of the core Pd is that any new addition has to be carefully thought out in order to account for all the idiosyncrasies of past versions and ensure there is a minimal chance of a regression. This vastly limits the development pace.

As a result, the Pd community sought to complement Pure Data's compelling core functionality with a level of polish that would lower the initial learning curve and improve user experience. In 2002 the community introduced the earliest builds of Pd-extended [13], the longest running Pd variant. There were other ambitious attempts, like pd-devel, Nova, and DesireData [14], and in recent years Pd has seen a resurgence in forks that aim to sidestep usability issues through alternative approaches, including embeddable solutions (e.g. libPd) and custom front ends. Pd-extended was probably the most popular alternative Pd version which continues to be used by many, even though it was abandoned in 2013 by its maintainer Hans-Christoph Steiner due to lack of contributors to the project.

Pd-L2Ork presents itself as a viable alternative which started out as a fork of Pd-extended and continues to be actively maintained. We begin with a discussion of Pd-L2Ork's history, motivation and implementation. We then look at Pd-L2Ork's most recent off-spring nick-named "Purr Data", which has recently been released as Pd-L2Ork version 2, runs on Linux, Mac and Windows, and offers some unique new features, most notably a completely new and improved GUI component. The paper concludes with some remarks on availability and avenues for future developments.

2 History and Motivation

Introduced in 2009 by Bukvic, Pd-L2Ork [2] started as a Pd-extended 0.42.5 variant. The focus was on nimble development designed to cater to the specific needs of the Linux Laptop Orchestra (L2Ork), even if that meant suboptimal initial implementations that would be ironed out over time as the understanding of the overall code base improved and the target purpose was better understood through practice.

An important part of L2Ork's mission was educational outreach. Consequently, a majority of early additions to Pd-extended focused on usability improvements, including graphical user interface and editor functions. While some of these were incorporated upstream, a growing number of rejected patches began to build an increasing divide between the two code bases. As a result in 2010 Bukvic introduced a separately maintained Pd-extended variant, named Pd-L2Ork after L2Ork for which it was originally designed.

Over time, as the project grew in its scope and visibility, it attracted new users, and eventually a team of co-developers, maintainers and contributors formed around it. This is obviously important for the long-term viability of the project, so that it doesn't fall victim to Pd-extended's fate, and thus the development team continues to invite all kinds of contributions.

Pd-L2Ork's philosophy grew out of its initial goals and the early development efforts. It is defined by a nimble development process allowing both major and iterative code changes for the sake of improving usability and stability as quickly as possible. Another important aspect of this philosophy is releasing improvements early and often in order to have working iterations in the hands of dozens of students of varying educational backgrounds and experience, which ensured quick vetting of the ensuing solutions.

Despite an ostensibly lax outlook on backwards compatibility, to date Pd-L2Ork and Purr Data remain compatible with Pd (the `-legacy` flag can be used to disable some of the more disruptive changes). In particular, there haven't been any changes in the patch file format, so patches created in Pd still work without any ado in Pd-L2Ork and vice versa (assuming that they don't use any externals which aren't available in the target environment). Also, communication between GUI and engine still happens through sockets, so that the two can run in separate processes (running the engine with real-time priorities).

Like Pd-extended, Pd-L2Ork provides a single turnkey monolithic solution with all the libraries included in one package. This minimizes overhead in configuring the programming environment and installing supplemental libraries, and addresses the potential for binary incompatibility with Pd.

3 Implementation

Pd-L2Ork's code base increasingly diverges from Pd. It consists of many bug-fixes, additions and improvements, which can be split into engine, usability, documentation, new and improved objects and libraries, scaffolded learning and rapid prototyping. In this section we highlight some of the most important user-visible changes and additions, more details can be found in the authors' PdCon paper [3].

3.1 Engine

Internal engine contributions have largely focused on implementing features and bug-fixes requested by past and existing Pd users. Some of these include patches that have never made it to the core Pd, such as the cord inspector (a.k.a. magic glass), improved data type handling logic, and support for outlier cases that may otherwise result in crashes and unexpected behavior. Additional checks were implemented for the Jack [6] audio backend to avoid hangs in case Jack freezes. Default sample rate settings are provided for situations where Pd-L2Ork may run headless (without GUI), thus removing the need for potentially unwieldy headless startup procedures. The `$0` placeholder in messages now automatically resolves to the patch instance, while the `$@` argument can be used to pass the entire argument set inside a sub-patch or an abstraction.[1] `[trigger]`[2] logic has been expanded to allow for static allocation of values, which alleviates the need for creating bang triggers that are fed into a message with a static value.

Visual improvements: The Tk-based [19] graphical engine has been replaced with TkPath [17] which offers an SVG-enabled antialiased

[1] In Pd parlance, an *abstraction* is a Pd patch encapsulating some functionality to be used as a subpatch in other patches.

[2] Here and in the following we employ the usual convention to indicate Pd objects by enclosing them in brackets.

canvas.[3] A lot of effort went into streamlining "graph-on-parent" (Pd's facility to draw GUI elements in a subpatch on its parent), including proper bounding box calculation and detection, optimizing redraw, and resolving drawing issues with embedded graph-on-parent patches. Improvements also focused on sidestepping the limitations of the socket-based communication between the GUI and the engine, such as keyboard autorepeat detection. As a result, the `[key]` object can be instantiated with an optional argument that enables autorepeat filtering, while retaining backward compatibility.

Stacking order: Another substantial core engine overhaul pertains to consistent ordering of objects in the glist (a.k.a. canvas) stack. This has helped ensure that objects always honor the visual stacking order, even after undo and redo actions, and has paved the way towards more advanced functionality including advanced editing techniques and a system-wide preset engine.

Presets: The preset engine consists of two new objects `[preset_hub]` and `[preset_node]`. Nodes can be connected to various objects, including arrays, and can broadcast the current state to their designated hub for storing and retrieval. Multiple hubs can be used with varying contexts. The ensuing system is universal, efficient, unaffected by editing actions, and abstraction- and instance-agnostic (e.g., using multiple instances of the same abstraction is automatically supported). It supports anything from recording individual states to real-time automation of multiple parameters through periodic snapshots.

Data structures: Data structures are an advanced feature of Pd to produce visualizations of data collections such as interactive graphical scores. Pd-L2Ork enhances these with the addition of sprites and new ways to manipulate the data.

3.2 Usability

On the surface Pd-L2Ork builds on Pd-extended's appearance improvements. Under the hood, with the canvas being drawn as a collection of SVG shapes, the entire ecosystem lends itself to a number of new opportunities. The most obvious involve antialiased display, advanced shapes (e.g. Bézier curves that are also used for drawing patch cords), support for image formats with alpha channel, and advanced data structure drawing and manipulation using SVG-centric enhancements (Fig. 1).

[3]SVG = Scalable Vector Graphics, a widely used vector image format standardized by the W3C.

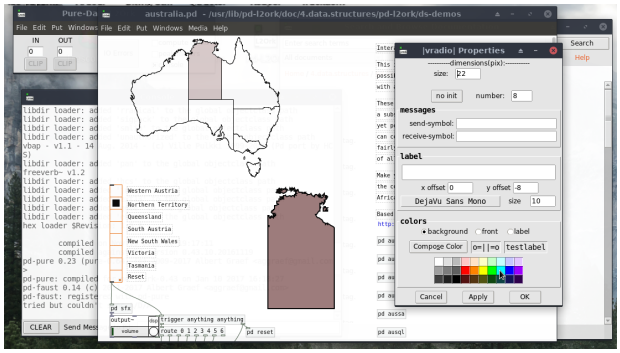

Figure 1: Pd-L2Ork running on Linux.

A majority of usability improvements focus on the editor. The *consistent stacking order* implemented in the engine has served as a foundation for the infinite undo, as well as to-front and -back stacking options that are accessible via the right-click context menu. Lots of improvements and polishing went into the *iemgui objects*, such as improved positioning, enhanced properties dialogs and graph-on-parent behavior.

The old *autotips patch* was integrated (and improved upon). The *tidy up* feature has been redesigned to offer a two-step realignment of objects. (The first key press aligns the objects on a single axis, while the second respaces them, so that they are equidistant from each other.) *Intelligent patching* was implemented to provide four variants of automatically generating multiple patch cords based on user's selection, and to provide additional ways of creating multiple connections (e.g. SHIFT + mouse click). The *canvas scrolling* logic has been overhauled to minimize the use of scrollbars, provide minimal visual footprint, and ensure most of the patch is always visible.

Pd-L2Ork supports *drag and drop* and has support for pasting *Pd code snippets* (using Pd's "FUDI" format) directly onto the canvas. The *copy and paste* engine has been overhauled to improve buffer sharing across multiple application instances. The entire graphics engine is *themeable* and its settings are by default saved with the rest of the configuration files.

3.3 Object Libraries

Apart from the core Pd objects and improvements described in the Engine section above, Pd-L2Ork offers a growing number of revamped

objects while also pruning redundant and unnecessary objects.

Special attention was given to supporting the Raspberry Pi (RPi) platform with a custom set of objects designed specifically to harness the full potential of the RPi GPIO and I2C interfaces, including `[disis_gpio]` and `[disis_spi]` [4]. The cyclone library has received new documentation and a growing number of bugfixes and improvements. Ggee library's `[image]` has received a significant overhaul and became the catchall solution for image manipulation. In addition to the standard Pd-extended libraries, Pd-L2Ork has reintroduced `[disis_munger~]` and an upgraded version of the `[fluid~]` soundfont synth external which depend on the flext library. Other libraries include fftease, lyonpotpourri, and RTcmix~. Pd-L2Ork bundles advanced networking externals `[disis_netsend]` and `[disis_receive]`, convenience externals like `[patch_name]`, and abstractions (e.g., those of the K12 learning module [5], and a growing number of L2Ork-specific abstractions designed to foster rapid prototyping). A few libraries have been removed due to lack of support and/or GUI object implementations that utilize hardwired Tcl-specific workarounds.

3.4 Introspection

Most interpreted languages have mechanisms to do introspection. Pd-L2Ork features a collection of "info" classes for retrieving the state of the program on a number of levels, from the running Pd instance to individual objects within patches. Four classes provide the basic functionality:

- `[pdinfo]` reflects the state of the running Pd instance, including dsp state, available/connected audio and midi devices, platform, executable directory, etc.
- `[canvasinfo]` is a symbolic receiver for the canvas, abstraction arguments, patch filename, list of current objects, etc. The object takes a numeric argument to query the state of parent or ancestor canvases.
- `[classinfo]` offers information about the currently loaded classes in the running instance. This includes creator argument types, as well as the various methods.
- `[objectinfo]` returns bounding box, class type, and size for a particular object on the canvas.

While the introspection provided by these classes is relatively rudimentary, it alleviates the need for a large number of external libraries that add missing core functionality. For example, Pd-L2Ork ships with several compiled externals whose purpose is to fetch the list of abstraction arguments. These externals all have different interfaces and are spread across various libraries. Having one *standard* built-in interface for fetching arguments that behaves similarly to other introspection interfaces improves the usability of the system. Furthermore, opening up rudimentary introspection to the user increases the composability of Pd. Functionality that previously only existed inside the C code can now be implemented as an abstraction (i.e., in Pd itself). These don't require compilation and are more accessible to a wider number of users to test and improve them.

4 Purr Data a.k.a. "The Cat"

Despite all of the improvements it brings to the table, Pd-L2Ork still employs the same old Tcl/Tk environment to implement its graphical user interface. This is both good and bad. The major advantage is compatibility with the original Pd. On the other hand, Tcl/Tk looks and feels quite dated as a GUI toolkit in this day and age. Tcl is a rather basic programming language and its libraries have been falling behind, making it hard to integrate the latest GUI, multimedia and web technologies. Last but not least, Pd-L2Ork's adoption was severely hampered by the fact that it relies on some lesser-used Tcl/Tk extensions (specifically, Tk-Path and the Tcl Xapian bindings) which are not well-supported on current Mac and Windows systems, and thus would have required substantial porting effort to make Pd-L2Ork work there.

Purr Data was created in 2015 by Wilkes to address these problems.[4] The basic idea was to replace the aging Tcl/Tk GUI engine with a modern, open-source, well-supported cross-platform framework supporting programmability and the required advanced 2D graphical capabilities, without being tied into a particular GUI toolkit again.

[4] Readers may wonder about the nick-name of this Pd-L2Ork offspring, to which the author in his original announcement at `http://forum.pdpatchrepo.info` only offered the explanation, "because cats." Quite obviously the name is a play on "Pure Data" on which "Purr Data" is ultimately based, but it also raises positive connotations of soothing purring sounds.

Employing modern web technologies seemed an obvious choice to achieve those goals, as they are well-supported, cross-platform and toolkit-agnostic, programmable (via JavaScript), and offer an extensive programming library and built-in SVG support (as a substitute for Pd-L2Ork's use of TkPath which incidentally follows the SVG graphics model).

There are basically two main alternatives in this realm, nw.js[5] a.k.a. "node-webkit" and Electron[6]. These both essentially offer a standalone web browser engine combined with a JavaScript runtime. nw.js was chosen because it offers some technical advantages deemed important for Purr Data (in particular, an easier interface to create multi-window applications and better support for legacy Windows systems).

So, in a nutshell, Purr Data is Pd-L2Ork with the Tcl/Tk GUI part ripped out and replaced with nw.js. Purr Data's GUI is written entirely in JavaScript, which is a much more advanced programming language than Tcl with an abundance of libraries and support materials. Patches are implemented as SVG documents which are generally much more responsive and offer better graphical capabilities than Tk windows. They can also be zoomed to 16 different levels and themed using CSS, improving usability. The contents of a patch window is drawn and manipulated using the HTML5 API. Thus the code to display Pd patches is very portable and will work in any modern GUI toolkit that has a webview widget.

There are also some disadvantages with this approach. First, Tcl code in Pd's core and in the externals needs to be ported to JavaScript to make it work with the new GUI; we'll touch on this in the following subsection.

Second, the size of the binary packages is much larger than with Pd-L2Ork or Pd-extended since, in order to make the packages self-contained, they also include the full nw.js binary distribution. This is a valid complaint about many of the so-called "portable desktop applications" being offered these days, but in the case of Purr Data it is mitigated by the fact that plain Pd-L2Ork is not exactly a slim package either.

Third, the browser engine has a much higher memory footprint than Tcl/Tk which might be an issue on embedded platforms with very tight memory constraints.

[5] `https://nwjs.io/`

[6] `https://electron.atom.io/`

So far, none of these issues has turned out to be a major road-block in practice. The most serious issue we're facing right now probably is that externals using Pd's Tcl/Tk facilities need to have their GUI code rewritten to make it work with Purr Data; this is a substantial undertaking and thus hasn't been done for all bundled externals yet.

4.1 Implementation

Using JavaScript in lieu of Tcl as the GUI programming language poses some challenges. Tcl commands with Tk window strings are hard-coded into the C source files of Pd. This means that any port to a different toolkit must either replace those commands with an abstract interface, or write middleware that turns the hard-coded Tcl strings into abstract commands. Given the complexity of Tcl commands in both the core and external libraries, that middleware would essentially have to re-implement a large part of the Tcl interpreter.

Consequently, Purr Data opted for the former approach of directly implementing an abstract interface. This takes the form of a JavaScript API providing the necessary GUI tie-ins to the engine and externals, which is called from the C side using a new set of functions (`gui_vmess` et al) which replace the corresponding functions of Pd's C API (`sys_vgui` etc.). As already mentioned, this means that externals which use these facilities need to have their GUI code rewritten to make it work with the new GUI. (Affected externals will work, albeit without their GUI features.)

Adding to the porting difficulty is the fact that Pd has no formal specification, and its GUI interface follows no common design pattern for 2D graphics. For example, the graph-on-parent window appears at a glance as a viewport that clips to a specified bounding box. However, the bounding box itself behaves inconsistently–for built-in widgets like `[hslider]` or `[bng]` it clips (per widget, not per pixel), but for graphed arrays, data structure visualizations, and widget labels it does no clipping at all.

To get to grips with these problems, Purr Data's JavaScript GUI implementation draws and manipulates Pd patch windows using the HTML5 API, which is widely documented and used. The Pd canvas itself is implemented as an SVG document. SVG was chosen because it is a mature, widely-used 2D API. Also, larger canvas sizes have little to no performance impact

on the responsiveness of the graphics. Since Pd patches can be large, this makes SVG a better choice for drawing a Pd canvas than the standard HTML5 canvas.

4.2 Leveraging HTML5 and SVG to Improve Pd Data Structures

Purr Data employs a small subset of the SVG specification to implement quite substantial improvements to data structure visualization. Inheriting from a pre-existing standards-based 2D API has several advantages over an ad-hoc approach. First, if implemented consistently, the existing SVG documentation can be used to test and teach the system. Second, it is not necessary to immediately understand all the design choices of the entire specification in order to implement parts of it. Since those parts have been used and tested in a variety of mature applications, it makes it easier to avoid mistakes that often riddle designs made by developers who aren't graphics experts. Finally, there is less risk of a standards-based API becoming abandoned than a more esoteric API.

To improve data structure visualizations, several `[draw]` commands were added to support the basic shape/object types in SVG. The currently supported types are `circle`, `ellipse`, `rect`, `line`, `polyline`, `polygon`, `path`, `image`, and `g`.[7] Each has a number of methods which map directly to SVG graphical attributes. Methods were also added for Document Object Model (DOM) events to trigger notifications to the outlet of each object.

The screenshot in Fig. 2 shows the "SVG tiger" drawn from a few hundred paths found inside the `[draw g]` object. Even though the drawing is complex, Purr Data caches the bounding box for the tiger object to prevent the hit-testing from causing dropouts. One can mouse over the tiger and trigger real-time audio synthesis.

It is also possible to set parameters for most of the `[draw]` attributes. For instance, the message `opacity z` can be sent to set a shape's opacity to be whatever the value of the field `z` happens to be for a particular instance of the data structure. As soon as the value of `z` changes, Pd then automatically updates the opacity of the corresponding shape accordingly.

[7]The latter `g` element denotes a "group", which is implemented as a special kind of subpatch that allows the attributes of several `[draw]` commands to be changed simultaneously.

Figure 2: Interactive SVG data structure.

4.3 Custom GUI Elements

As the SVG tiger example shows, Purr Data makes it possible to bind HTML5 DOM events to SVG shapes. Reporting the events is not enabled by default, but can be switched on by simply sending the appropriate Pd message to the `[draw]` object, such as the `mouseover 1` message in Fig. 2. Each `[draw]` object has an outlet which then emits messages when events like mouse-over, movements and clicks are detected.

It goes without saying that this considerably expands Pd's capabilities to deal with user interactions, e.g., if the user wants to modify elements of a graphical score in real-time. But it also paves the way for enabling users to design any kind of GUI element in plain Pd, without having to learn a "real" programming language and its frameworks.

For instance, Fig. 3 shows a collection of three knobs drawn using the new SVG `[draw]` commands, whose values (represented by the `r` field in the `nub` data structure, which is linked to the rotation angles of the knobs) can be manipulated by dragging the mouse up or down. The values can then be read from the data structure using Pd's built-in `[get]` object and used for whatever purpose, just like with any of the built-in GUI elements.

Pd offers a rather limited collection of built-in GUI elements to be used in patches, and extending that collection needs a developer proficient in both C and Tcl/Tk. Purr Data's new SVG visualizations totally change the game, because any Pd user can do them without specialized programming knowledge. We thus expect the

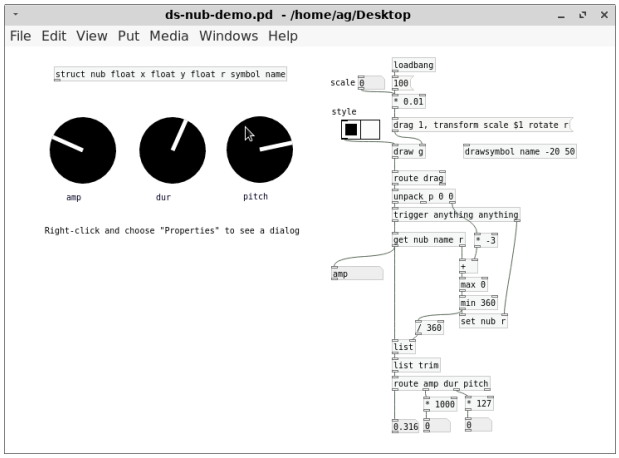

Figure 3: Custom GUI elements.

facilities sketched out above to be used a lot by Pd users who want to enrich their patches with new kinds of GUI elements. As soon as it becomes possible to conveniently package such custom GUI elements as graph-on-parent abstractions, we hope to see the proliferation of GUI element libraries which can then be used by Pd users and modified for their own purposes.

5 Getting Pd-L2Ork

The sources of Pd-L2Ork and Purr Data are currently being maintained in two separate git repositories.[8] There are plans to merge the two repositories again at some point, so that both versions will become two branches in the same repository, but this has not happened yet.

For Purr Data there is also Github mirror available at `https://agraef.github.io/purr-data/`. This is mainly used as a one-stop shop to make it easy for users to get their hands on the latest source and the available releases, including pre-built packages for Linux, macOS and Windows.

Because of Pd-L2Ork's addons and its comprehensive set of bundled externals, the software has a lot of dependencies and a fairly complicated (and time-consuming) build process. So, while the software can be built straight from the source, it is usually much easier to use one of the available binary packages:

- Virginia Tech's official Pd-L2Ork packages are available at `http://l2ork.music.vt.edu/main/make-your-own-l2ork/software/`.
- Jonathan Wilkes' Purr Data packages can be found at `https://github.com/agraef/purr-data/releases`.
- JGU also offers Pd-L2Ork and Purr Data packages for Ubuntu and Arch Linux. Web links and installation instructions can be found at `http://l2orkubuntu.bitbucket.org/` and `http://l2orkaur.bitbucket.org/`, respectively.

The JGU packages can be installed alongside each other, so that you can run both "classic" Pd-L2Ork and Purr Data on the same system. (This may be useful, e.g., if you plan to use Pd-L2Ork's K12 mode which has not been ported to Purr Data yet.) We mention in passing that JGU's binary package repositories also contain Pd-L2Ork and Purr Data versions of the Faust and Pure extensions which further enhance Pd's programming capabilities.[9]

6 Future Work

After Purr Data's initial release as Pd-L2Ork 2.0 in February 2017, "classic" Pd-L2Ork has become version 1.0 and went into maintenance mode. While development will continue on the Purr Data branch, we will keep the original Pd-L2Ork available until all of Pd-L2Ork's features have been ported or have suitable replacements in Purr Data.

Purr Data has matured a lot in the past few months, but like any project of substantial size and complexity it still has a few bugs and rough edges we want to address after the initial release, in particular:

- Port the remaining missing features from Pd-L2Ork (autotips and K12 mode).
- Port legacy Tcl code that is still present in the GUI features of some of the 3rd party externals.
- Some code reorganization is in order, along with a complete overhaul of the current build system.

One interesting direction for future research is leveraging the new SVG visualizations as a means to create custom GUI elements in plain Pd, i.e., as ordinary Pd abstractions. This will make it much easier for users to create their

[8] cf. `https://github.com/pd-l2ork/pd` and `https://git.purrdata.net/jwilkes/purr-data`

[9] Grame's Faust and JGU's Pure are two functional programming languages geared towards signal processing and multimedia applications [7, 8].

own GUI elements, and will hopefully encourage community contributions resulting in libraries of custom GUI objects ready to be used and modified by Purr Data users.

With the expansion onto other platforms, Pd-L2Ork's key challenge is ensuring sustainable growth. As with any other open-source project of its size and scope, this can only be achieved through fostering greater community participation in its development and maintenance, so please do not hesitate to contact us if you would like to help!

7 Acknowledgements

The authors would like to thank the original Pd author Miller Puckette, numerous community members who have complemented the Pd ecosystem with their own creativity and contributions, including Hans Christoph Steiner and Mathieu Bouchard. We would also like to thank the L2Ork sponsors and stakeholders without whose support Pd-L2Ork would have never been possible nor sustainable.

References

[1] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth, and H.-C. Steiner. Embedding pure data with libpd. In *Proceedings of the Pure Data Convention*, volume 291. Citeseer, 2011.

[2] I. Bukvic, T. Martin, E. Standley, and M. Matthews. Introducing L2ork: Linux Laptop Orchestra. In *Interfaces*, pages 170–173, 2010.

[3] I. Bukvic, J. Wilkes, and A. Gräf. Latest developments with Pd-L2Ork and its development branch Purr-Data. PdCon 2016, New York, NY, USA. `http://ico.bukvic.net/PDF/PdCon16_paper_84.pdf`, 2016.

[4] I. I. Bukvic. Pd-L2ork Raspberry Pi Toolkit as a Comprehensive Arduino Alternative in K-12 and Production Scenarios. In *NIME*, pages 163–166, 2014.

[5] I. I. Bukvic, L. Baum, B. Layman, and K. Woodard. Granular Learning Objects for Instrument Design and Collaborative Performance in K-12 Education. In *New Interfaces for Music Expression*, pages 344–346, Ann Arbor, Michigan, 2012.

[6] P. Davis and T. Hohn. Jack audio connection kit. In *Proc. Linux Audio Conference, LAC*, volume 3, pages 245–256, 2003.

[7] A. Gräf. Signal Processing in the Pure Programming Language. In *Proceedings of the 7th International Linux Audio Conference*, pages 137–144, Parma, 2009. Casa della Musica.

[8] A. Gräf. Pd-Faust: An integrated environment for running Faust objects in Pd. In *Proceedings of the 10th International Linux Audio Conference*, pages 101–109, Stanford University, California, US, 2012. CCRMA.

[9] D. Iglesia. MobMuPlat (iOS application). *Iglesia Intermedia*, 2013.

[10] K. Jolly. Usage of pd in spore and darkspore. In *PureData Convention*, 2011.

[11] J. Kincaid. RjDj Generates An Awesome, Trippy Soundtrack For Your Life. `http://social.techcrunch.com/2008/10/13/rjdj-generates`.

[12] C. McCormick, K. Muddu, and A. Rousseau. PdDroidParty-Pure Data patches on Android devices. *Retrieved January*, 21, 2014.

[13] [PD-announce] MacOSX installers for pd 0.36 and pd 0.36 extended (CVS).

[14] pd forks WAS : Keyboard shortcuts for "nudge", "done editing". `http://permalink.gmane.org/gmane.comp.multimedia.puredata.general/79646`.

[15] M. Puckette. Pure Data: another integrated computer music environment. *In Proceedings, International Computer Music Conference*, pages 37–41, 1996.

[16] M. Puckette. Max at seventeen. *Computer Music Journal*, 26(4):31–43, 2002.

[17] TkPath. `http://tclbitprint.sourceforge.net/`.

[18] Unity - Game Engine. `https://unity3d.com`.

[19] B. B. Welch. *Practical programming in Tcl and Tk*, volume 3. Prentice Hall Upper Saddle River, 1995.