figusdevpack: A Python-Based Framework for Audio and Multimedia Newbies

Renato Fabbri
NICS, IA, Unicamp
Rua da Reitoria, 165
Cidade Universitária "Zeferino Vaz"
CEP 13091-970 Campinas/SP,
Brazil,
renato.fabbri@gmail.com

Marília F. Chiozo
IC, Unicamp
Av. Albert Einstein, 1251
Cidade Universitária "Zeferino Vaz"
CEP 13084-971 Campinas/SP,
Brazil,
chiozo.marilia@gmail.com

Abstract

In this work we present a cross-platform open-source Python-based framework for audio and multimedia. This framework, named *figusdevpack*, is designed for reducing application development time and acquainting new users with existing resources for multimedia development. It is structured around a wrapper library, example applications, tutorials and documentation.

Keywords

Python, audio/multimedia development

1 Introduction

To regular users of multimedia creation software, making the transition to a fully scripted environment such as Python can be a gruelling task. The reasons behind such transition usually ranges from curiosity to lack of elegant methods to achieve an intended goal. Giving up the transition halfway through due to time constraints or loss of interest is not unheard of.

Although there are efforts focused on teaching the necessary programming concepts to non-programmers (such as [1] and [2]), existing libraries of interest in multimedia development have, at times, a steep learning curve, and require the user to skim through several pages of documentation before actual usage details of classes and methods become clear. This process is tiresome and inefficient.

Our work intends to address this issue. On the most basic level, a wrapper library aims to provide simpler interfaces to existing multimedia libraries. We call our wrapper library *figuslib*, which is distributed as part of *figusdevpack* (or simply *fdp*). This *fdp* is centered on tutorial windows and example applications and code. Among the included example applications, one can find, along with programs written for the sole purpose of showcasing the wrapper library, use cases drawn from real-life applications, such as the *figuslib* port of [3]. The set of such real-life applications is called FIGUS. Documentation is provided as part of *fdp* for wrapper classes and standalone methods of the library, describing their technical aspects such as the meanings and domains of method arguments and data members. The tutorials, accessible from the main *fdp* frontend along with the example applications and the documentation, are meant to walk the user through the process of writing complete applications and also serves as a guide to the documentation for users who are not familiar with the wrapper library.

SoniPy[4] is an independently developed project that is similar to ours when one compares current end products, yet it holds a fundamental difference in purpose. While *fdp* is aimed primarily at the novice programmer, with secondary goals such as assisting the process of learning how to program and forming a community of users with a common specific interest (audio programming), SoniPy targets developers in a more general way, placing more emphasis on the tools themselves than on the users' backgrounds.

2 Problem Description

In developing audio applications, we encounter three main sub-problems that demand special attention: 1) choosing a suitable approach to sound manipulation for the task at hand; 2) deciding how the user is to control the chosen approach; and 3) deciding on a sound rendering method.

Within *fdp* a developer should be able to manipulate standard Python data structures and, simultaneously, be familiar with at least some basics about the possibilities of the framework being employed. As soon as these goals are achieved, the user should be able to conceive and manipulate appropriate data structures for the intended functionality.

As control interfaces, *fdp* offers, besides standard command-line control, GUI interfaces (pro-

vided by the cross-platform wxPython package), optional Pygame input and visual feedback through Pygame itself or WxMpl[1]. *Mirra*-based visual feedback is planned for the near future, providing 2D graphics via OpenGL.

Finally, *fdp* presents OSC-controlled ChucK and CSound as sound rendering backends for the multimedia developer to work with. OSC interfacing is achieved through the SimpleOSC package, and CSound has its own Python bindings in the form of the Csnd package. The use of OSC interfaces allows for future expansion such that *fdp* support SuperCollider and PureData as backends.

3 Language Choice

The philosophy of code reuse, which has been directing programming language efforts since even before the advent of object-oriented programming, is satisfactorily embodied by the Python programming language. Through its use, developers are able to wrap code fragments as components that can then be reused by other Python developers, who employ the language as the logical glue that holds components together and gives them a larger purpose. However, Python certainly is not the only language to possess such traits. This raises the question of why we chose it over other available languages.

The language is by no means perfect. Several of its downfalls are a result of its interpreted nature and shared with other interpreted languages in some level. Speed is one such downfall. By foregoing compilation (in the sense it goes for compiled languages), interpreted languages place that overhead on load time, execution, or both. Yet, from a novice's point of view, undesirable though it might be, a sacrifice of program speed is often preferrable to a sacrifice of programming speed. A more specific issue that impacts speed is memory management. Automatic garbage collection, for example, while having the obvious advantadge of cutting down on a large number of common memory leaks that are disencouraging to novices, the extreme focus on spatial optimization of some implementations might hurt execution speed in unacceptable ways. There is little that can be done about this issue other than relinquishing some control to the user.

On the other hand, Python combines a sizeable number of libraries for developers to work with, a smooth learning curve, easily-accessible tools and an existing user base in the field of multimedia development. As is the case with other interpreted languages, Python bindings for existing C/C++ libraries can be obtained by tools such as SWIG, which automate most of the process of writing bindings. The presence of a command-line interpreter enables new users to experiment with small segments of code without going through the entire process of preparing a source file and compiling it, which is often required from users of languages such as Java and C/C++. Additionally, as a dynamically typed language with an automatic garbage collection scheme, it enjoys the status of a language that does not force programmers to be particularly attentive to variable definitions and memory allocation issues, both of which can be somewhat overwhelming to novice programmers. Because we focus especially on developers with little or no programming experience, choosing a language with a steep learning curve would defeat the very purpose of our work.

In the field of multimedia development, Python benefits from being a market trend. A larger user base means higher development activity, which translates into faster evolution, more innovations and quicker response to bugs. This factor, added to endorsement by common Linux distributions such as Fedora, makes Python a language that is easily available, and is so at its prime.

4 Components and Tools

At present moment, *fdp* deals with the following components:

- NumPy, adds fast and sophisticated array facilities to the Python programming language. [5]
- Audiolab, allows importing and exporting between sound files and NumPy arrays. [6]
- Chuck, a programming language for real-time audio. [7]
- Csnd, provides a Python interface to the Csound API. [8]
- OSC Protocol, allows multimedia devices to share music performance data in real time over a network. [9]
- SimpleOSC, provides a simple way to use OSC Python implementation. [10]
- Zombi, defines a very basic set of widgets that sends out OSC messages. [10]

[1]The WxMpl module provides wxPython GUI elements with embedded Matplotlib graphics.

- wxPython, a cross-platform wxWidgets wrapper for the Python programming language. [11]
- WxMpl, provides interactive plotting capabilities via the Matplotlib 2D plotting library. [12]
- Pygame, a set of modules written on top of the SDL library and designed for writing games. [13]
- IPython, an enhanced Python shell. [14]
- Easy Install, allows automatic download, building and installation of Python packages. [15]
- Editors: Kate [16] (Linux), Notepad2 [17] (Windows).
- *figuslib*, a wrapper library targeting audio and multimedia.
- FIGUS, a collection of applications built around *figuslib*.
- *fdp*, is *figuslib*, FIGUS, documentation and tutorials.

For cases where the user needs more specific mathematical functions, NumPy is available. Furthermore, through NumPy it is possible to use SciPy tools, such as signal processing Scikits[2]. Audiolab, recently added to Scikits, enables users to quickly import and export between NumPy array objects and different sound file formats.

While Chuck and Csnd target audio rendering, OSC, accessible from Python via SimpleOSC, is a dedicated protocol for communicating between compliant applications.[3] Additionally, Chuck and Csound can be accessed using the `os.system` method from Python Standard Library. [13]

These methods of communicating with sound renderers are readily compatible with Python itself, therefore GUI and graphical utilities can potentially be interfaced with audio – and video – renderers.

The Pygame package provides a Python interface to Simple DirectMedia Layer (more commonly known as SDL), a general purpose library for multimedia applications that has been used for over five years by C/C++ programmers as a cross-platform API that provides high-level access to services such as those offered by Microsoft's low-level multimedia API, DirectX. SDL can be used to handle input (mouse, joysticks and the keyboard) and accelerated graphics. For audio, Pygame offers few and intuitive capabilities such as playback and audio CD-ROM access.

4.1 IPython, Easy Install, Editors

Seeing as this project targets end-users, it is our concern that these users have a starting point when it comes to choosing programming tools such as interpreters and editors.

As of the current *fdp* version, we strongly recommend the use of IPython as an interpreter, EasyInstall and the editors Kate (on Linux) or Notepad2 (on Windows). We intend to include options for automatically installing IPython, and Kate or Notepad2, while EasyInstall is already bundled with *fdp* due to being our Python module installer of choice for this project.

4.2 figuslib, FIGUS, *fdp*

figuslib is a wrapper library built around selected functionalities of the libraries listed above. It is a structured pool of contributed code segments striving towards ease of use of *fdp* components within audio and multimedia applications.

FIGUS is a set of applications using *figuslib* and *fdp* components. This set is comprised of developers' and users' contributions, such as open source applications using figuslib and external multimedia modules.

fdp is *figuslib* and FIGUS together with tutorial texts and scripts. These resources are intended to be presented as "wizards", in clean step-by-step expositions, building a knowledge base around the usage of *figuslib* and external multimedia modules not yet wrapped by *figuslib*. These texts, scripts and "wizards" are being designed to be collaboratively expanded upon by *figuslib* developers and users.

Summarizing, these three pieces of software exist for mainly one purpose: aggregating contributions of developers and users to components of *fdp*.

4.3 Example

As an example of some of the components involved, we present the following code, which captures mouse data from a wxPython object

[2] Scikits are independent projects that are too specialized to live in SciPy itself. [18]

[3] OSC also acts as the core of the DSSI plugin API, an evolution of the LADSPA API. LADSPA and DSSI are linux API meant for audio processing and synthesizing.

and sends it by means of OSC messages to an OSC-aware application.

```
# Importing wxPython and simpleOSC to
    the script
import wx, osc

class MyFrame(wx.Frame):
 """Subclass of Frame in which we
    read mouse
 positions (X, Y) and send them
    through simpleOSC
 to localhost , port 9000, address /
    sinus"""

 def __init__(self):
   wx.Frame.__init__(self, None, -1,
       "My Frame", size = (600, 500))
   panel = wx.Panel(self, -1)

   # Binding the mouse movement event
       : wx.EVT_MOTION
   # to the "OnMove" function of our
       MyFrame class
   panel.Bind(wx.EVT_MOTION, self.
       OnMove)

   wx.StaticText(panel, -1, "Pos: ",
       pos = (10, 12))
   self.posCtrl = wx.TextCtrl(panel,
       -1, "", pos = (40, 10))

 def OnMove(self, event):
   # Getting the position of the
       mouse move event
   pos = event.GetPosition()
   # Setting its values on the
       TextCtrl object self.posCtrl
   self.posCtrl.SetValue("%s, %s" % (
       pos.x, pos.y))

   # Sendind the position values
       through OSC
   # To localhost, port 9000 (
       simpleOSC default for sending)
   # and /sinus address (i.e. what is
        requested of the server)
   osc.sendMsg("/sinus", [pos.x, pos.
       y])


if __name__ == '__main__':
 osc.init()
 app = wx.PySimpleApp()
 frame = MyFrame()
 frame.Show(True)
 app.MainLoop()
```

With an OSC-aware sound synthesis application, this script could be a starting point from where a user wishing to use the mouse device as a sound synthesis controller can build a more complex application. The documentation and examples bundled with *fdp* aim to enable a user to reuse code similar to this after skimming through only a few documentation topics that are directly related to the intended task. For a proper introduction to the libraries used by *fdp*, we intend to provide practical examples and documentation covering from uses with barely any integration between components to uses that fully integrate several components in order to perform complex tasks.

5 *fdp* Focus on End-User

Although *figuslib* relies mainly on third-party Python packages that generally target developers, *fdp* is designed with end-users in mind. Well-documented example scripts exploring different functionalities provided by *figuslib*, links to related projects and further documentation aims at guiding new users through the transition to Python multimedia development, as long as they are willing to read and experiment.

Even so, the idea of providing non-programmers (or those new to Python multimedia development) with a well-documented framework can be helpful at usage stages later than introduction. The complexity of most libraries nowadays requires even seasoned programmers to visit the documentation and look at code examples often, save for a few frequently-used fragments of code. It is our intention that the *fdp* framework relieve users of the need to mentally index documentation and code examples by adopting an organization intuitive enough that even first-time users should be able to "hit" the desired section of the documentation with few or no "misses" at all.

6 Cross-Platform Open-Source Software Approach

The components used in the development of *fdp* are open-source, as will be *fdp* itself as of its initial public release. Source code documentation serves not only the purpose of aiding the maintenance cycle, but it also aims to let users learn from the very source code of *fdp*. This is true of the *figuslib* wrappers as well as of the example applications and the *fdp* frontend.

The open-source license allows users to contribute to the documentation and code, assuming a cooperative work stance towards developers. This is also a design goal of *fdp* as a whole: inviting users to experiment with new, unfore-

seen combinations of *figuslib* components and allowing them to play a role that goes one step further from simple tests and bug reports.

7 Conclusions and Further Work

At its current development stage, the *fdp* effort is focused on providing the end-user with a comprehensive set of tools (in the form of packages), documentation and examples. Once the bare essentials have been covered, we will move towards broadening the set of choices given to the user in different areas. As mentioned in previous sections, these include visual feedback through Mirra, a OpenGL-based 2D graphics package, and the option to use SuperCollider and PureData as sound rendering backends. Another inclusion that has been discussed by the developers is ScyPy integration. The SciPy library is built to work with NumPy arrays, and provides several user-friendly and efficient numerical routines such as routines for numerical integration or optimization.

Collecting voluntary user feedback and adapting *fdp* to meet the needs of users will become of vital importance in the development process. This is especially true in the case of new users who have not yet been exposed to the existing resources available to assist the transition to Python multimedia development. The initial public release of *fdp* is scheduled to mid-February, 2008, as an open-source project under the General Public Licence (GPL), hosted at SourceForge[4].

8 Acknowledgements

Our thanks go to the developers of all software used in this project, as well as to Professor Adolfo Maia Júnior. Special thanks go to Rohit Gupta for all the invaluable tips.

References

[1] Renato Fabbri and Fábio Furlanete. Python for audio manipulation, 2007. Partially presented at the 5^{th} Linux Audio Conference. An ongoing Portuguese version is available at: http://estudiolivre.org/tiki-index.php?page=python-e-som-tutorial%.

[2] Dive into python, 2007. Available at: http://www.diveintopython.org/.

[3] *Applications of Group Theory on Granular Synthesis*, 2007. Available at: http://lcpd.ime.usp.br/~marioct/anais-sbcm2007-sem-capa.pdf.

[4] *Overcoming Software Inertia in Data Sonification Research Using the SoniPy Framefork*, 2007. Available at: http://www.avatar.com.au/sonipy/Worrall_SonipyICoMCS07.pdf.

[5] Numpy, 2007. Available at: http://numpy.scipy.org/.

[6] David Cournapeau. Audiolab, 2007. Available at: http://www.ar.media.kyoto-u.ac.jp/members/david/softwares/audiola%b/.

[7] Ge Wang and Perry Cook et al. Chuck webpage, 2007. Available at: http://chuck.cs.princeton.edu/.

[8] Introduction to using csnd: the python wrapper for csound, 2007. Available at: http://www.avatar.com.au/sonify/csnd.

[9] The Center For New Music and Audio Technology (CNMAT). opensoundcontrol.org, 2007. Available at: http://opensoundcontrol.org/.

[10] ixi audio. Ixi backyard, 2007. Available at: http://www.ixi-software.net/content/backyard.html.

[11] wxpython homepage, 2007. Available at: http://www.wxpython.org.

[12] Ken McIvor. Wxmpl, 2005. Available at: http://agni.phys.iit.edu/~kmcivor/wxmpl/.

[13] Pygame homepage, 2007. Available at: http://www.pygame.org.

[14] Ipython, 2007. Available at: http://ipython.scipy.org/moin/.

[15] Easy install, 2007. Available at: http://peak.telecommunity.com/DevCenter/EasyInstall.

[16] Anders Lund, Christoph Cullmann, and Joseph Wenninger. Kate homepage, 2007. Available at: http://kate-editor.org/.

[17] Florian Balmer. Notepad2, 2007. Available at: http://www.flos-freeware.ch/notepad2.html.

[18] Scikits, 2007. Available at: http://scipy.org/scipy/scikits/wiki.

[4] http://sourceforge.net/projects/fdpack/