SONIPY: THE DESIGN OF AN EXTENDABLE SOFTWARE FRAMEWORK FOR SONIFICATION RESEARCH AND AUDITORY DISPLAY

David Worrall, Michael Bylstra, Stephen Barrass, Roger Dean

Sonic Communications Research Group
University of Canberra
worrall@avatar.com.au, mbylstra@digitalfeast.com.au,
stephen.barrass@canberra.edu.au, roger.dean@uws.edu.au

ABSTRACT

The need for better software tools was highlighted in the 1997 Sonification Report [1]. It included some general proposals for adapting sound synthesis software to the needs of sonification research. Now, a decade later, it is evident that the demands on software by sonification research are greater than those afforded by music composition and sound synthesis software. This paper compares some major contributions made towards achieving the Report's proposals with current sonification demands and outlines SoniPy, a broader and more robust model which can integrate the expertise and prior development of software components using a public-domain community-development approach.

[Keywords: sonification software, open source, python]

1. INTRODUCTION

To-date, software for data sonification has been developed either as standalone applications engineered from first principles, sometimes incorporating third-party low-level audio routines, or as more expansive sonification 'environments' that attempt to encapsulate some general principles and procedures that can be adapted for specific sonification projects as the need arises.

The standalone applications tend to be designed for individual experiments entailing clearly defined tasks such as accurate monitoring [2], or graphic user-interaction [3], whereas environments tend to be more expansive projects, often with less deterministic outcomes. They afford greater flexibility than is possible within standalone applications. Some recent environments still in development [4][5] seem to have been designed by first choosing a music composition environment and working backwards, perhaps trusting that the data-processing needs at the 'input end' can be adequately handled by the language tools available from within the particular composition system chosen. Considering that many software tools for music composition have a long gestation period (in the case of Csound [6], about forty years, for example) and are still being actively developed, as can be witnessed, for example, by the daily activity on the developer mailing lists for Csound and SuperCollider [7], this approach is natural and is the approach assumed in the Sonification Report [8].

In addition to the two primarily scripted environments just mentioned, MAX/MSP [9] and its sibling PD [10] are another type, which emphasises graphical user interfaces (GUIs) over program code. Whilst the scripting-verses-GUI debate is still active, it is clear from the large user-base and active development of new patch objects for these platforms that the GUI approach is appealing to some users and perhaps offers a gentler initial learning curve for many exploratory sonification researchers who are visually inclined. In any event, a considerable investment of time is necessary to become proficient in any of these environments and having made the investment, a certain amount of environment "stickiness" is apparent and understandable.

1.1. The First Bottleneck: Data

In data sonification, whilst the input data can be thought of as eventually controlling the sound rendering, the transformations it has to undergo in the interim can be considerable. Such data processing can reasonably include multidimensional scaling, filtering and statistical analysis which itself may itself become the subject of sonification. Also, each input dataset can have potentially unique structural characteristics. Some, such as EEG data, may be multiple channels of AC voltages with a variety of DC-biases and noisiness as determined by the particular data collection setup on a particular patient. Others, such as security data flowing from a market trading-engine, will be massively paralleled, metadata embedded and multiplexed into a single "feed." Difficulties in using such data are compounded when the it needs to be buffered and streamed in non-real-time as is the need for multiple overlays of time sequences of different temporal compressions.

High-level tools for processing such data complexities are rarely, if ever, found in computer music environments, and even less likely if the input data is spatial rather than temporal. When such an a environment is the principle sonification tool, a common response to complex data processing requirements is for someone, where possible, to 'bite the bullet' and write data-processing routines in the language of the composition environment itself. This is currently the approach used by SonEnvir [5] and OctaveSC [11], which both use SuperCollider [7] and also the PD-based Interactive Sonification Toolkit [12]. Whilst SuperCollider's SClang is a very elegant and powerful composition environment that can support the development of data-processing solutions, being unique, it lacks the transportability that more general and widely available tools afford. One consequence of this is that, in projects without a dedicated programmer, practical assistance for what are essentially data processing problems is more difficult to obtain. Data is thus often pre-processed using external tools such as spreadsheets and then read from files by the music composition

environment; a procedure that, whilst it may be appropriate in "limited data" experiments, is at best susceptible to data corruption and of no use if the data is coming from a real-time feed or from a dynamic model [13]

This situation may be characterised as 'data SONIFICATION' i.e. the primary focus is on sound rendering whilst input data is constrained so it can be dealt with adequately by the rendering software. The alternative, an emphasis on data-processing tools at the expense of sound rendering flexibility ('DATA sonification') is no more attractive because the sound palette tends to be small and the range of controls limited, the outcome of which is too-often difficult to listen to over extended periods of time. Some examples of this latter approach include extensions for the Matlab numerical environment [14], AVS Visualization Toolkit [15], Excel spreadsheets [16], and the R statistical analysis package [17]. They provide data handling and processing capabilities but very basic sample-based sound capabilities modeled on the MIDI protocol. What are needed are tools that afford a balanced, equally-flexible approach.

Excellent data-processing tools exist in the public domain and are an integral part of much scientific research. Furthermore, they are continually being extended and modernised by teams of developers spread across the world. Yet because of the decision to use a music composition environment for sound rendering, these tools remain inaccessible to most sonification environments. Whether this situation has come about because the data for music composition by computer is mostly internally-generated rather than externally-acquired is open to debate but until sophisticated tools for handling externally-acquired often massively multiplexed datasets can be brought to bear on the acquisition, analysis, storage and re-presentation requirements of the sonification process even before any mapping is undertaken, let alone sound rendering, there is limited chance that such software will enable the "choosing [of] mappings between variables and sounds interactively, navigating through the data set, synchronizing the sonic output with other display media, and performing standard psychophysical experiments" that the Sonification Report [8] envisaged.

1.2. Motivations

Sonic Communications Research Group members undertake a wide variety of sonification tasks, ranging from sound installations, data-bending improvisations and algorithmic synaesthesia [18] to more empirically-based psychological experiments. This range of activity necessitates flexible methods for generating, shaping and translating input data, DSP-level control of sound rendering, as well as methods for collecting and analyzing participant reactions. We first came to consider possible solutions the issues outlined in the previous section as a result of the difficulties we experienced in trying to sonify the same large multidimensional dataset on different hardware platforms, under different versions of operating systems, and with each sonifier having a preference and expertise in a different collection of sound synthesis/music composition programs. Whilst we are all technically literate, we soon realised that if the difficulties *we* experienced were any indication, it must be very difficult for almost everyone to be able to confidently achieve consistent, repeatable results with anything but the simplest datasets. This led us to specifying the requirements for an experimental software sonification framework. Some requirements are clearly identified in the Sonification Report, others of our own concoction. We call the framework SoniPy, in-keeping with the naming convention used for frameworks that extend the Python programming language.

2. SONIPY: CONCEPTS AND REQUIREMENTS

SoniPy is designed to be a heterogeneous software framework for data sonification research and auditory display. It integrates various already existing independent components such as those for data acquisition, storage and analysis, cognitive and perceptual mappings as well as sound synthesis and control, by encapsulating them, or control of them, as Python modules. The choice of Python was not arbitrary, as it possesses all the features of a modern modular programming language that we consider essential for an experimental development environment. Python is

> a general-purpose programming language … which may also serve as a glue language connecting many separate software components in a simple and flexible manner, or as a steering language where high-level Python control modules guide low-level operations implemented by subroutine libraries effected in other languages. [19]

Other descriptors include: simple, but not at the expense of expressive power, extensible, embeddable, interpreted, object-oriented, dynamically typed, upwardly compatible, portable and widely and freely available [20].

2.1. Design Requirements

As the Sonification Report's Sample Research Proposal #3 [1] acknowledges, the development of a comprehensive "sonification shell" is not easy. The depth and breadth of knowledge, and skills required to effect such a project are easily underestimated. Whilst it has been a decade since the Report was published, progress has been slow. This is not to criticise those that have fallen by the wayside, nor those still in development, but to acknowledge both the difficulties involved in such a project and the need for new requirements if such projects are to have a better chance of survival. We briefly address the requirements indicated in the Sonification Report and add some of our own.

Integrability. As discussed earlier with regard to data, due consideration needs to be taken of the requirements of the various components of the sonification and experimentation process. As is the case with most interdisciplinary ventures, each contributing discipline brings its collection of tools, techniques and standards to the venture and they need to be synergistically integrated. A software environment needs to be chosen that supports this goal. It is for this reason we have chosen Python, which can be used to "wrap" independent pieces of conformable software in such a way as to permit data to flow between them. We follow Python convention and call them Modules.

Flexibility. Rather than try to be the "killer application," SoniPy aims to wrap (inherit, or be extended by) the best collection of Modules available to it. These Modules need to have no computational interdependencies, though conceptually they may be similar, thus ensuring that no one of them is indispensable. Each of these Modules has evolved

independently, probably over a considerable period of time. Independent of SoniPy, they have their own ongoing development teams that extend and improve then as well as adapting them to ever-changing hardware and software platforms.

Extensibility. In the situation where no Module exists for a particular task, a new Module can be designed in the knowledge that it will fit seamlessly within the existing Modular framework. This implies all Modules need to be thread compliant.

Accessibility. It is desirable that as many Modules as possible be known in their own right. This reduces learning overhead for all users and enables work that may have already been undertaken with those tools to be accommodated within the SoniPy framework.

Portability. SoniPy needs to be able to be instantiated on all major platforms. Furthermore, it is desirable, in certain applications, for Modules to be instantiated on different machines, in different locations and networked together: that is, be heterogeneous.

Availability. To protect both authors and users, SoniPy needs to be freely available with a minimum of restrictions. There are numerous licensing flavours for public-domain software whose source-code is made generally available, as outlined by the GNU organisation [21]. Because SoniPy employs heterogeneous components, the license of each component carries through into SoniPy in a way that is standard practice in the software industry. The SoniPy-specific components will be issued under GPL General Public License Version 2 [22] thus encouraging the sharing and free-exchange of these tools in the community at the same time as enabling restrictions to be applied for individual projects as confidentiality agreements demand. SoniPy's sources and documentation can be freely downloaded from its Sourceforge Internet repository [23].

Durability. SoniPy needs to *survive*. Whilst survival can never be guaranteed, in complex projects such as this, maximum risk-mitigation is essential. SoniPy is unlikely to survive if it remains the effort of a very small group. Essential to this is the community involvement and support in ongoing improvement and development: the very conditions under which SoniPy's independent modules have been, and continue to be, developed.

2.2. Integration Through Wrapping

Although Python comes with an extensive standard library and there is a good resource of external Python libraries, we are not limited to using Python libraries. A powerful feature of Python is its set of well-defined interfaces to other languages. Libraries written in most languages can be integrated through Python by 'extending' it [24]. The basic principle of SoniPy is to use Python to 'wrap' independent software that can be compiled with python bindings in such a way that data can flow between them. Quite a few tools exist for the (semi-) automatic generation of Python bindings, such as the Simplified Wrapper and Interface Generator (SWIG) [25].

Some applications provide Python Application Program Interface (API) libraries; other applications need to have a Python API written in order to use it. Although some others embed Python, either by bundling it as an interpreter or by invoking the Python interpreter installed on the user's system as a basic API [26]. We mention embedding in this context because, whilst it may be useful in its own right, it does not provide the interface flexibility needed by SoniPy. SoniPy requires an application to provide Python bindings so that Python can be *extended* by the application.

3. THE DESIGN OF SONIPY

The SoniPy design specifies five Module Sets communicating over two different networks: the SonipyDataNetwork (SDN) and the SonipyControlNetwork (SCN). Modules are grouped according to their role in the data sonification process: Data Processing (DP), Conceptual Modeling (CM), Psychoacoustic Modeling (PM), Sound Rendering (SR) and Monitoring & Feedback (MF). Depending on the dictates of a particular project, modules in a Set may be instantiated on different machines. A particular Module Set may be empty, i.e. contain no modules, or a particular module may belong to more than one Module Set.

3.1. Inter-Module Communication: The Networks

SoniPy's modular design makes it well suited for the instantiation of all selected modules on a single processor or, in order to take advantage of the computing power that multiple CPUs and machines can afford, the distribution of modules over multiple CPUs, a LAN or the Internet. We are currently extending Python 2.4 under OSX 10.4.9 but workable alternatives will flow as the development team expands.

Python's platform independence enables SoniPy to be distributed over a heterogeneous network. Other potential uses of a distributed approach include mobile phone sound-rendering and the processing of data remotely under local control, perhaps with the result being sent to another site for mapping, and psychoacoustic adjustment before being rendered to sound.

We are in the process of testing different approaches to distributing the computing in order to maximally benefit from the trade-off between performance (including real-time latency, data throughput and CPU overhead issues), ease-of-use, maintainability, reliability (over a network), scalability and heterogeneity; that is, the ability for non-Python third-party applications or devices to communicate with SoniPy modules. [27]. Communication technologies being tested range from class inheritance, Sockets, OSC and MIDI through to network audio mixing, using Netjack [28] for example.

Referring to Figure 1, which is a diagrammatic representation of the way Module Sets interrelate, it can be observed that SoniPy's modules operate through two networks: Data and Control. The SonipyDataNetwork (SDN) is topologically configured as a Bus whilst the SonipyControlNetwork (SCN) is a star configuration. This is analogous to the signal and control busses of an automated audio mixing desk. Control routing uses the same network technology as Data, though the destinations may be different. For example data from a DP module may be sent to a CM module on the SDN bus, under the control of an MF module communicating on the SonipyControlNetwork, without the data itself needing to go through an MF module. Sonipy Controls need to be XML compliant and each Module Set may itself be the hub of a network of processors of topology unknown to the SCN router.

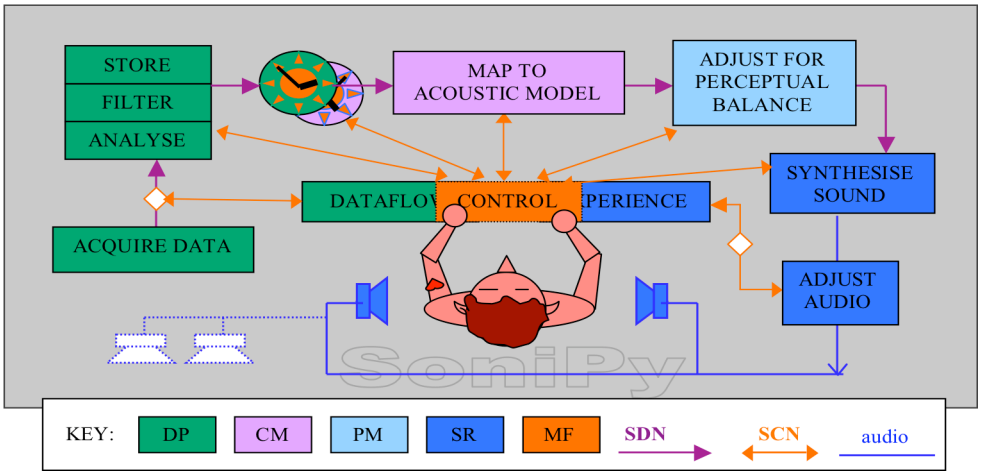

Figure 1. *Flow Diagram of SoniPy's Five Module Sets and Two Networks*

3.2. The Modules

Whenever possible, local modules are instantiated as library extensions to the Python programming framework. Should a module be instantiated remotely, that instantiation and the control of information to and from it remains under the control of the MF module that initiated the instantiation, where the main application loop as well as thread and GUI controls resides.

Module Sets	Data via SoniPy Data Network (SDN)	Controls via SoniPy Control Network (SCN)
DP	Input: Raw data to be sonified Processes: Analysis, filtration, translation, storage Output: Modified Data to be sonified, Metadata	Process selection & OP switching options State (active, waiting, idle)
CM	Input: Data from DP, model selection algorithms Processes: Model Selection Output: Selected Model(s)	Process selection & output switching options State (active, waiting, idle)
PM	Input: Model control parameters Processes: Psychoacoustic transforms Output: Modified model control parameters	Process selection, & OP switch options Model instantiation map for render(s)) State (active, waiting, idle)
SR	Input: Synthesis models and controls Processes: Render sound Output: Sound	Audio controls: mute, unmute, gain etc Audio control changes State (active, waiting, idle)
MF	Input: Login, config. & process startup Processes: Initiate, route, record activity, monitor usage, system/networks state Output: UI, feedback, log of activity	State (active, waiting, idle, off) Resource usage, UI monitoring.

Table 1. *Overview of some key features of SoniPy Module Sets. Key; DP: Data Processing. CM: Conceptual Modeling, PM: Perceptual Modeling, SR: Sound Rendering, MF: Monitoring and Feedback, SDN: SoniPy Data Network, SCN: SoniPy Control Network.*

Table 1 provides an overview of some Key features of the Module Sets.

3.2.1. Data Processing

As the principal data processing activity in data sonification is taking place inside the listeners, the role of sonification is to prepare source data in a format that enables the listener to extract information, and in interactive systems, to take account of their feedback. SoniPy's Data Processing (DP) modules consist of a number of Object-Oriented classes which themselves inherit Data Classes and Control Classes according to the form and location of the raw data and its intended destination. Class methods include those for

- Interpolated lookup and mappings, for Auditory Icons and Earcons, for example,
- Writing data to and extracting it from storage (memory, database and/or flat file) for pre-processing or multi-stream playback,
- Audification - writing data in formats acceptable as direct input to audio hardware,
- Simultaneous handling of multiple time-locked streams, such as from biomedical monitors,
- Deconstruction, analysis and filtering, including of complex meta-tag embedded multiplexed streams, such as a data feed from a stock-market trading engine,
- Model-based sonification involving user feedback, and
- Simulation of data feeds, including buffering with time compression and expansion.

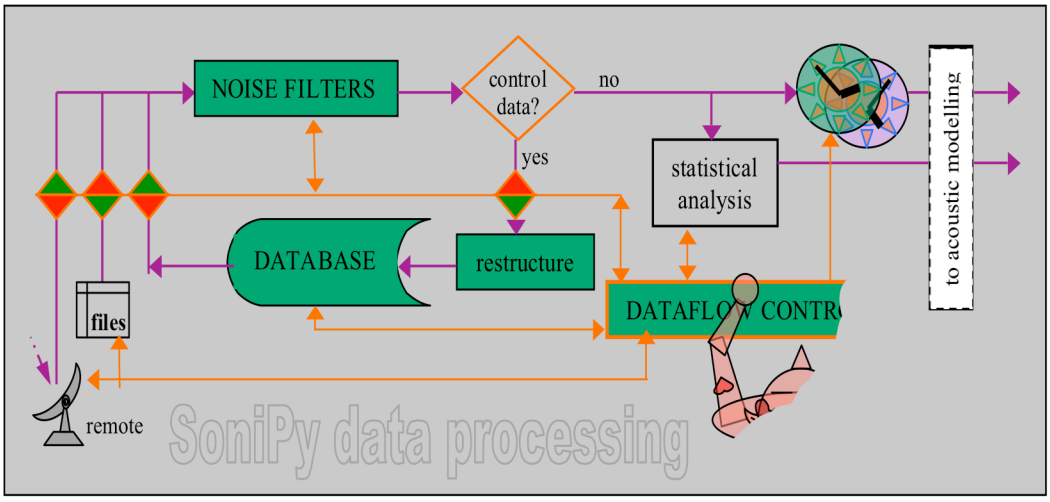

Figure 2. *A representation of one configuration of SoniPy's Data Processing modules*

Data and Control Class instances can be manipulated with SciPy (Scientific Python) [29] a collection of open-source libraries for mathematics, science, and engineering. The core library is NumPy (Numeric Python) that provides convenient and fast N-dimensional array manipulation. Figure 2 is a diagrammatic representation of one way of configuring SoniPy's Data Processing modules under SCN control.

3.2.2. Conceptual Modeling and Data Mapping

Information mapping is divided into separate cognitive and psychoacoustic stages. The cognitive stage involves the design of "sound schemas" with semiotics, metaphors and metonyms relating to the task, and aesthetic and compositional aspects relating to genre, culture and palette. Decisions have to be made about functionality, aesthetics, context, learnability, expressiveness, and device characteristics. These decisions are typically draw from existing knowledge and theories from relevant sciences, arts and design. The consequences of these compounding decisions are difficult to predict empirically: one of the reasons why sonification is currently more of an heuristic art than a science. Nevertheless, as different conceptual models are developed, some based in cognition, others culturally determined, they can be integrated into SoniPy using the wrapping techniques outlined.

One example is the TaDa method; a design approach to sonification that provides a systematic user-centred process to address the multitude of decisions required to design a sonification [30]. TaDa starts from a description of a use case scenario, and an analysis of the user's task, and the characteristics of the data. This analysis informs the specification of the information requirements of the sonification.

SoniPy's support for the TaDa method will be through a python-based GUI that captures a user scenario and provides standard TaDa fields for analysis. This GUI is connected using the SDN to a mySQL database that contains about 200 stories about everyday listening experiences, analysed using the TaDa data-type fields. This database, called Earbenders, is a case-based tool for looking up "sound schemas" at the cognitive design stage [31]. In future, a python interface to the SonificationDesignPatterns [32] wiki could be developed as an alternative Pattern Language approach for cognitive level design.

3.2.3. Psychoacoustic Modeling

The Psychoacoustic Modeling stage involves the systematic mapping of information relations in the data to perceptual relations in the sound schema [33]. SoniPy provides support for this by allowing interactive reconfiguration of the mapping from information relations to auditory relations. Changes in this mapping cause the automatic remapping of source information through psychoacoustic algorithms (using NumPy and SciPy) to produce new sounds and/or rendering controls. For example a change from categorical to ordered information could automatically produce a remapping from a categorical sound (e.g. instrument, object, stream) to an ordered property of a sounding object (e.g. length, excitation, distance).

3.2.4. Sound Rendering

SoniPy provides a sonifier access to many more options than if a music composition or sound synthesis environment was chosen before beginning the development of other aspects of the data sonification framework. For low level audio work, a Portaudio [34] module can be used for audification and as the basis for the development of other such modules should the need arise. SndObj [35] is a middle-level toolkit also immediately available in the same manner. In a similar vein the STK toolkit [36] appears to be wrappable, as does the higher-level RTCmix C++ library [37]. At the time of writing the Csound developers are working on a version of Csound5 that has an embedded Python API.

Whilst some high-level applications such as Max/MSP and PD, are unlikely to become toolkits, it is still possible to use them by instantiating them independently and communicating with them via OSC [38] and MIDI [39]. SuperCollider3 is a

special case because of its inherent modularity: the sound-rendering component (SCsynth) can be instantiated as a separate program to the language (SClang). Communication between this 'external' SCsynth occurs over OSC using the SoniPy framework as an alternative to Sclang. If very low-latency is a requirement, such as may be the case for interactive sonifications, the pySCLang module enables direct communication with an instantiation of the SC language and its internal sound renderer.

Non-operating system dependent Text-To-Speech synthesis is available through Python wrappings of, using Espeak [40] or Festival [41]. We are currently building PySpeak, a Python API to an OSX thread-compliant version of Espeak.

3.2.5. Monitoring, Feedback and Evaluation

Monitoring and Feedback of SDN and SCN can happen via the Python interpreter. Having access to an interpreter in order to build a complete sonification by iteratively building on small tests is a powerful aspect of Python. Heterogeneous connectivity also allows the consequences of decisions at each stage to be tested on a compound design, thus enabling better understanding and control of non-linear and emergent effects in an overall design.

For GUI, wxPython provides access to wxWidgets [42] and wxGlade [43] can assist in more-rapid development of GUIs by automatically generating Python control code and separating the GUI design and event-handling code. If a relatively consistent interface across all hardware platforms is more desirable, Tcl [44] GUI building tools are available through native Python modules.

By including an Evaluation Module, it will be possible to use SoniPy to design different types of empirical experiments, and conduct and analyse the results within a single framework. A user-contributed library of experiments for evaluating a sonification design could assist in developing some standards for measuring the functionality, aesthetics, learnability, effectiveness, accuracy, expressiveness and other aspects of a design. These evaluations could thus assist in choosing between different designs for a particular sonification task.

4. EXAMPLE

The following example illustrates the SoniPy Framework in action. The first task is to accept a multiplexed meta-tagged data stream from the Australian Stock Exchange (ASX) trading engine. The ASX is medium-sized exchange, on which about 3,500 securities are traded. It generates about 100 MB of trading text data daily, making it impractical to hold enough data in RAM to do all the calculations necessary.

The data is processed into a MySQL database using an Object-Relation Mapping paradigm supported by the **sqlobject** module [45]. This abstracts the handling of the dataset, providing an interface between the tables and indices database paradigm and Python's object-orientation. Other modules (such as **mySQLdb**) exist if direct interaction with the database server in MySQL code is more appropriate.

A list of securities that meet, or are likely to meet, the criteria necessary for a sonification event to initiated, is held in RAM and processed as a multidimensional array using the **numpy** module. When the criteria are met this data is also used as some of the input parameters to the sound renderer. The **pyspeak** module is invoked to synthesise the name of the security being newly rendered.

In this example, the sound is rendered by the Supercollider 3 'external synthesis engine, scsynth, with which the python **scsynth** module bi-directionally communicates using the **OSC** protocol. This permits the use of synthdefs (synthesis definition algorithms) that are capable of responding to the criteria as established or as modified in real-time. Other synthesis options, such as the lower-level **pysndobj** or **pyaudio** (the python interface to portaudio) are possible, as is the **libsndfile** library.

The python code example illustrates how SoniPy combines Python code, imported 3rd party modules and user-defined scripts.

```
# :::::::::::: import the modules needed ::::::::::::::::::::::
# for Australian Stock Exchange (ASX) Trading Engine datafeed.

import DataFeeder
import StreamMonitor
       # uses matplotlib for graphical presentation
       # of trading engine stats
import DataToDB
       # includes an import of the sqlobject Module
       # which provides OO Class structure of the
       # Relation-to-Object-Mapping necessary for
       # handling DBs. For simple DB calls, use mySQLd
       # to simply pass mySQL commands to the server.
import SecurityMaps
       # mapping OO classes for ASX securities.
       # this Module calls the numpy module for
       # fast multi-dimensional array processing.
import PsychoFilters
       # a set of filters for transforming mapped data
       # Used to adjust Security Maps for psychophysical
       # non-linearities, condition enhancement etc
import SoundRenders
       # A set of synthesis engine interfaces and
       # synthesis definitions (instruments) appropriate
       # for this sonification. Includes OSC and MIDI,
       # and the phoneme synthesiser pyspeak.

# :::::::::::: ASX sonification threads ::::::::::::::::::::::
ASXFeed=DataFeeder(feedURL='http://localhost')
       # Establish connection to datafeed. The dtafeed
       # URL could be an internet,LAN address or filepath.
       # Uses Python build-in modules for low-level
       # data transfer to RAM buffer.
StreamMonitor.monitorStream (ASXFeed)
       # Monitor data stream.
DataToDB.MultiplexStreamtoDB(ASXFeed, ConnectParams)
       # Start processing ASXFeed into DB. Uses a MySQL DB
       # DB client connection as specified in ConnectParams.
       # Other servers are possible, including remote ones.
ASXalerts=SecurityMaps.Indicator(stocks, UpBollinger(20,9,2))
       # a FIFO of alerts for securities currently trading
       # outside the specified upper Bollinger band. The FIFO
       # is constantly updated whilst DataFeeder is active.
SoundRenderers.BinauralOut='TRUE'
       # Process sound output will for binaural listening.
while (StreamMonitor.StillActive):
    for security in ASXalerts:
        if security not in renderList:
            pyspeak(security)          # announce new security
            SoundRenders.scsynth(security, stock.scsynthdef)
```

Code Example 1. *The Sonipy Framework in action. "#" is the Python comment character.*

5. CONCLUSION

Sonification research is an interdisciplinary activity and in the past, tools for undertaking it have either been discipline-specific, modified to accommodate the interconnections, ad-hoc collections of tools or stand-alone programs developed for a specific task. Because SoniPy's open architecture design can integrate modules conforming to widely accepted inter-process computation standards (wrappable libraries), it will be possible for it to grow in most directions its user-community needs it to.

Instead of sonification researchers trying to make a decision about which particular piece of software to use for an experiment, based hopefully on a best-fit evaluation of existing software capabilities, SoniPy can afford a continuity a framework of both existing and developing of tools not currently available. This should assist individual endeavour and promote the independent evaluation of empirical experimentation necessary for scientific validation. There is currently, for example, a dearth of good public-domain software for experimental psychology involving sound and the integration of such a Module Set into SoniPy would be a welcome addition.

By establishing SoniPy an open source project we hope to share our work with others who in return will contribute to a framework that is capable of great flexibility and general usefulness to the sonification community.

6. APPENDIX OF ACRONYMS USED

AC – Alternating Current
API – Application Program Interface
CM – Conceptual Module
CPU – Central Processing Unit
DC – Direct Current
EEG – Electroencephalogram
GNU – GNU is not Unix
GUI – Graphic User Interface
LAN – Local Area Network
MF – Monitoring and Feedback
MIDI – Musical Instrument Digital Interface
OSC – Open Sound Control
SCN – SoniPy Control Network
SDN – SoniPy Data Network
SQL – Structured Query Language
SR – Sound Rendering
SWIG - Simplified Wrapper Interface Generator
XML – eXtensible Markup Language

7. REFERENCES

(all URLs current as at 20070201)

[1] G. Kramer et al. *Sonification Report: Status of the Field and Research Agenda. Prepared for the National Science Foundation by members of the International Community for Auditory Display*. 1997. Located at http://www.icad.org/websiteV2.0/References/nsf.

[2] C. Chafe and R. Leistikow. "Levels of Temporal Resolution in Sonification of Network Performance," in *Proceedings of the 2001 International Conference on Auditory Display,* Espoo, Finland, July 29-August 1, 2001

[3] B. Walker and J.T. Cothran, "Sonification Sandbox: A Graphical Toolkit for Auditory Graphs," in *Proceedings of ICAD 2003, Boston,* 2003.

[4] S. Pauletto and A. Hunt, "A Toolkit for Interactive Sonification," in *Proceedings of ICAD 04-Tenth Meeting of the International Conference on Auditory Display,* Sydney, Australia, July 6-9, 2004.

[5] A. de Campo, R. Frauenberger and R. Höldrich, "Designing a generalized sonification environment" in *Proceedings of ICAD 04-Tenth Meeting of the International Conference on Auditory Display,* Sydney, Australia, July 6-9, 2004.

[6] http://www.counds.com/

[7] http://www.audiosynth.com/

[8] Kramer, G. et. al. 1997. op. cit. Section 5.3. "Developing Sonification Tools"

[9] http://www.cycling74.com/

[10] http://crca.ucsd.edu/~msp/software.html/

[11] T. Hermann. 2006. http://www.sonification.de/projects/sc3/index.shtml/

[12] S. Pauletto, S. and A. Hunt. "A toolkit for interactive sonification," in *Proceedings of ICAD 04-Tenth Meeting of the International Conference on Auditory Display,* Sydney, Australia, July 6-9, 2004.

[13] T. Bovermann, T. Hermann, and H. Ritter. "Tangible Data Scanning Sonification Model*." Proceedings of the 12th International Conference on Auditory Display*, London, UK June 20 - 23, 2006

[14] J.A. Miele. "Smith-Kettlewell display tools: a sonification toolkit for Matlab," in *Proceedings of the 2003 International Conference on Auditory Display, Boston, MA, USA, 6-9 July* 2003.

[15] B. Kaplan. "Sonification in AVS," in *AVS '93, Walt Disney World, Lake Buena Vista, FL, May 24-26* 1993.

[16] T. Stockman, G. Hind, G. and C. Frauenberger. "Interactive Sonification Spreadsheets*," in Proceedings of ICAD 05-Eleventh Meeting of the International Conference on Auditory Display,* Limerick, Ireland, July 6-9, 2005.

[17] M. Heymann and M. Hansen M. A new set of sound commands for R. Sonification of the HMC algorithm. in *Proceedings of the Acoustical Society of America (ASA),* 2002.

[18] R.T. Dean, M. Whitelaw, H. Smith and D. Worrall. "The Mirage of Algorithmic Synaesthesia: Some Compositional Mechanisms and Research Agendas in Computer Music and Sonification." *Contemporary Music Review* 25, 311-327. 2006.

[19] A. Watter, G.Van Rossen, J. Ahlstrom, J. *Internet Programming with Python*. M&T Books, NY, NY. 1996.

[20] http://www.python.org/

[21] http://www.gnu.org/philosophy/categories.html

[22] http://www.gnu.org/copyleft/gpl.html

[23] http://sonipy.sourceforge.net/

[24] M. Lutz. *Programming Python*. O'Reilly & Associates. 1996. Chapter 14, p505.

[25] http://www.swig.org/

[26] ibid, Chapter 15, p571.

[27] G. Coulouris, G. Dollimore, G. and T. Kindberg. *Distributed systems: concepts and design*. Addison-Wesley, Boston, MA. 2005.

[28] http://netjack.sourceforge.net/

[29] http://www.scipy.org/

[30] S. Barrass. TaDa! Demonstrations of Auditory Information Design, in *Proceedings of the Third International Conference on Auditory Display ICAD'96*, Xerox PARC, Palo Alto, California. 1996.

[31] S. Barrass. "EarBenders: Using Stories About Listening to Design Auditory Interfaces," in *Proceedings of the First Asia-Pacific Conference on Human Computer Interaction APCHI'96,* Information Technology Institute, Singapore. 1996.

[32] S. Barrass. "Sonification from a Design Perspective", Invited Keynote, in *Proceedings of the Ninth International Conference on Auditory Display, ICAD 2003*, Boston USA. 2003.
[33] S. Barrass. "Sculpting a Sound Space with Information Properties," *Organised Sound* 1(2):125:136 1996. Cambridge University Press, UK.
[34] P. Burk and R. Bencina. "PortAudio–An open source cross platform audio API" in *Proceedings of the ICMC, La Habana, Cuba*. 2001.
[35] V. Lazzarini. "The Sound Object Library," *Organized Sound* 5(1):35-49. 2000. Cambridge University Press, Cambridge, UK.
[36] G.P. Scavone and P. Cook, P. "RtMidi, RtAudio and a synthesis toolkit (STK) update," in *Proceedings of the 2005 International Computer Music Conference,* Barcelona, Spain, 2005.
[37] http://www.rtcmix.org/
[38] M. Wright, A. Freed, and A. Momeni. "Open sound control: State of the art 2003," in *Proceedings of the 2003 Conference on New Interfaces for Musical Expression*. National University of Singapore , Singapore. 2003.
[39] http://www.midi.org/
[40] http://espeak.sourceforge.net/
[41] http:// www.cstr.ed.ac.uk/projects/festival/
[42] http://www.wxwidgets.org/
[43] http://wxglade.sourceforge.net/
[44] http://www.tcl.tk/
[45] http://www.sqlobject.org/