SONECULES: A PYTHON SONIFICATION ARCHITECTURE

Dennis Reinsch

Ambient Intelligence Group
Faculty of Technology, Bielefeld University
Bielefeld, Germany
dreinsch@techfak.de

Thomas Hermann

Ambient Intelligence Group
Faculty of Technology, Bielefeld University
Bielefeld, Germany
thermann@uni-bielefeld.de

ABSTRACT

This paper introduces sonecules, a flexible, extensible, end-user friendly and open-source Python sonification toolkit to bring 'sonification to the masses'. The package comes with a basic set of what we define as *sonecules* which are sonification designs tailored for a given class of data, a selected internal logic for sonification and offering a set of functions to interact with data and sonification controls. This is a design-once-use-many approach as each sonecule can be reused on similarly structured data. The primary goal of sonecules is to enable novice users to rapidly get their data audible – by scaffolding their first steps into auditory display. All sonecules offer a description for the user as well as controls which can be adjusted easily and interactively to the selected data. Users are supported to get started as fast as possible using different sonification designs and they can even mix and match sonecules to create more complex aggregated sonecules. Advanced users are enabled to extend/modify any sonification design and thereby create new sonecules. The *sonecules* Python package is provided as open-source software, which enables others to contribute their own sonification designs as a sonecule – thus it seeds a growing/growable library of well-documented and easy-to-reuse sonifications designs. Sonecules is implemented in Python using mesonic [1] as the sonification framework, which provides the path to rendering-platform agnostic sonifications.

1. INTRODUCTION

Sonification is a highly interdisciplinary field and sonification designers usually have to combine knowledge in data science, acoustics, sound computation, sound engineering, signal processing, psychoacoustics & perception – and ideally ranging into music and cognition. This makes it difficult to get started with sonification, particularly for those in application domains who just heard about it and want to try the method. While there are many toolkits available to enable specific sonification types such as audification or auditory graphs, an easy-to-apply, flexible, reuse-optimzed, community-accepted general library or framework to support all available sonification methods is missing.

This paper aims at filling the gap with a new approach that positions a sonification library in the center of a highly popular Python community which already offers powerful data science libraries, visualization libraries, user interfaces (widgets) and a language that is rather easy to learn and use. Instead of reinventing wheels we aim at a modular system that focuses on the core of the problem: the glue to interconnect available parts and filling in suitable classes that provide both immediate utility and a scaffold for own development / extension steps. Figure 1 depicts how we see different tools interacting to enable our toolchain of high reuse-potential.

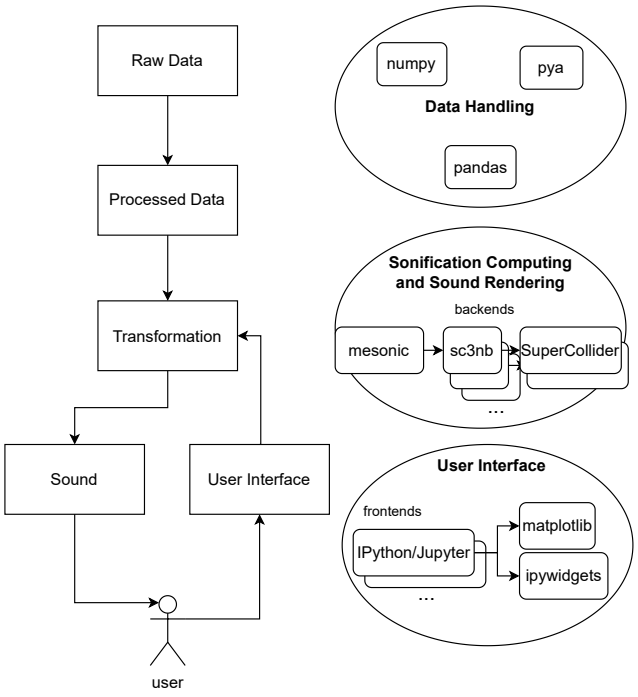

Figure 1: Sonification toolchain: on the left side the single steps of the sonification process are shown while on the right side relevant tools are depicted grouped by usage.

We first review related work on systems to support either end users or experts for creating, using and adapting sonifications. We then state fundamental concepts for our *sonecules* system and specify components to be employed in sonification design. To make things more tangible (resp. audible), we then systematically present a basic set of sonecules that both exemplify design and use of the system. They should be understood as starting point for a growing library of sonification tools that are as easily picked and

used as one can plot data with today's modern plotting packages in the Python ecosystem. We then briefly describe the interface of a sonecule and how it can be used. Finally we wrap up with an extensive discussion of the goals we aim to solve with sonecules and a call for feedback and contribution.

2. RELATED WORK

Many attempts at providing a sonification toolkit were made [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]. But still there is no universally accepted tool that is directly aimed at general/flexible sonification that is used across sonification experts. Especially for the layperson, there is still no clear way to go about getting their data sonified. This is in stark contrast to the visualization community, where within most computer languages there are solid packages to just plot what you need, e.g. think of matplotlib, Plotly, pyqtgraph, VTK, D3.js, to name a few. Before diving into the reasons for this, let's review the state of the art in sonification tools. There are currently two extremes in the sonification domain regarding available tools:

(i) open sound synthesis platforms high-jacked for sonification: One often finds open platforms which allow to create new sonifications from scratch, such as SuperCollider [29] or PureData [30], Max/MSP [31], Csound [32]. These require users to first learn, i.e., acquire a certain prior minimum of knowledge concerning sound synthesis to even get started. And while these tools offer a very rich experience for experts in regards of sound creation these tools can be still cumbersome to work with, as often these are mostly focused on providing sound synthesis tools and less capable of handling data. For instance, SuperCollider even lacks operator priorities making it difficult to code mathematical operations, at least for those accustomed to languages such as Python, Java, C/C++. As data handling is a highly crucial part of the sonification process, users then would be forced to implement data processing functions, which introduces friction in the developing process [16]. The history of SonEnvir [10, 33, 34, 35] provides an example for the approach to anchor a sonification tool directly into an 'originally-meant-for-sound-synthesis'-system.

The counter point are (ii) complex all-in-one graphical sonification design programs: in this case complete/self-contained packages are provided which often come with GUI, data import functions, and one (or few) very specific sonification design(s). Examples for this include the Highcharts Sonification Studio [9], Rotator [22] and the Sonification Workstation [24]. Such systems are much more beginner-friendly as compared to the former discussed class of approaches, but they are quite limited to their specific use cases as they are hard (if not impossible) to extend – or even to be adjusted concerning any parameter values or mappings, if developers did not already offer a control for it. This is because they are custom created programs which often consist of a large code base to integrate the implemented sonification design with specific data loading and processing capabilities and in addition provide a graphical user interface for all this.

To our opinion, both of these extremes create some kind of walled garden which hinders the sonification community to share their methods and grow. We believe that the walls can be teared down by following the example of how computer graphics was made available to users, even end-users that are little tech-savvy. Models are matlab, R, and: the Python, as a sprouting ecosystem for data science, and as a glue language that allows to tie in what is needed. Python as an particularly easy-to-learn computer language makes particular sense as it is highly popular, and comes already with packages for the most frequent and relevant data types and the data processing thereof, including time-series data, geospatial data, unstructured feature bundles etc. – think of numpy, scipy, pandas, scikit-learn, to name a few.

Starting from Python as anchor point for sonification systems is not a novel idea: important references are SoniPy [36, 16], and sc3nb [37]. The latter, however, largely focuses on providing a lean interface to control SuperCollider as the backend-of-choice for creating sonification programs within Python and thus only eases the mixed and matched usage of SuperCollider and Python necessary for sound computing and data processing. This flexibility to use the best of both worlds is welcome for proficient programmers but an obstacle for many end-users.

In an attempt to cut the knot, one aimed at developing a library to simultaneously meet the needs of both these different groups of users – and we *failed*: the problem is just too big to tackle in one go. Instead, we identified elements shared by basically all use cases, involving the wish of flexibility to operate a timeline, to either sonify in real-time or non-real-time, also the need to abstract from details of synthesis systems with the idea of coupling sound computing to the system via *backends*. We have introduced and published this mediating framework titled mesonic [1] elsewhere and mesonic serves here as central piece in our sonification toolchain.

So, with the *sonecules* package we try to avoid the extremes by providing ready-to-use sonification designs which offer users to easily try different kinds of sonification in a playful way, i.e., they can play with the easily adjustable parameters for the single sonifications, compare sonifications, keep the lightweight code (basically invocations of sonecules) as documentation and for later reproduction, and so on. Of course, there are already other approaches to collect sonification designs [38, 39, 40], but they lack the capability to actually use them directly and interchangeably with different data – which is a key point of sonecules. At the same time, each sonecule can, if needed, be modified down to the roots by adapting the code and thus creating new sonecules which are encouraged to be contributed to the sonecules library. The usage of mesonic [1] allows us to keep the relevant code for a sonification small and readable. The idea is to follow the UNIX principle [41], 'do one thing but do it well' – which is commonly used in software engineering to create small and easy-to-maintain building blocks that do one specific task well, which then can be combined as needed to solve complex tasks.

With sc3nb, mesonic and now sonecules, we wish to seed a rich ecosystem for using, sharing and coding auditory displays.

3. FUNDAMENTAL CONCEPTS

Sonifications are never monolithic, they are composites, consisting of data, a transformation (e.g. mapping or model), interaction (optionally), control parameters and their user interface, but most importantly sound computation. Inspired by chemistry where the properties of substances depend on their specific aggregation (chemical bonds) and feature manifold characteristics depending on their interconnection, we invented the word *sonecules* (reminding of molecules) to echo this fact. This ties nicely into the use of our *mesonic* framework - by which we mediate from high-level control to the sound computing side. The name, however, contains the elementary particle 'meson', which indeed points at a level of finer resolution. An accidental coincidence is that the primar-

ily featured sound computing system is *SuperCollider* - the name carrying the association of a particle accelerator, an apparatus to study the smallest existing particles. Perhaps a single sample of a sound signal might be that ultimate elementary particle of sound and be the ultimately lowest level of sound, but let's turn back to the *sonecules* level.

The focus of this paper is the *sonecule*, a single coherent sonification unit. A sonification design entails a concrete usage as in an object-oriented interface. As a good example for a sonecule we can take any specific Model-Based Sonification Design (aka Sonification Model [42]) such as the Data Sonogram Sonifiation model, because it is very clearly described how such a model is constructed. There are several key elements that should be stated:

data we need data to listen to. A sonification is by definition reusable with different data [43]. But it is obvious that we cannot use any data set for a specific sonification technique – for instance for Audification – as there are constraints regarding the kind of data and number of data points are required for a specific method. A sonification design should always clearly state the usable data and the data constraints for the user.

interaction we need to *control* the sonification. The amount of control needed depends again on the specific sonification design: most Model-Based Sonifications won't even sound without a user in the loop. Parameter Mapping Sonifications will often provide the need to adjust several different mappings, and even the simple playback of data (as in audification) requires to be started and thus to be embedded in time. A sonification design should always state how the user can interact with and control it.

algorithm a sonification is defined to be systematic and reproducible [43], which means that each sonification follows some sort of algorithm. The algorithm is the transformation from the data domain into the sound domain and can be very direct as in Audification or very indirect as in some Model-Based Sonifications. A sonification design should always clearly state how data are transformed and how the sound is generated.

purpose a sonification should objectively reflect properties or relations in the input data [43], which means that it makes sense to listen to sound as a way to gain understanding. A specific sonification design should always state what are possible use cases of the sonification.

These key elements should be stated for each sonification design in order to fulfill the requirements stated in the definition of sonification [43]. This will emphasize the usability of sonification as a scientific tool which follows an design-once-use-many approach. Additionally it allows us to create a sonecule which then is a manifestation of the sonification design in code. This offers then a quick way to start experimenting with different sonifications as it becomes obvious – based on the data to be sonified – which sonecules can be selected, how they transform the data and what control capabilities this brings to the listener and what purpose it serves.

This is a refined specification of the taxonomy of sonification as sonecules are sonifications that are more detailed than sonification techniques by directly specifying the underlying algorithm and the possible sonification controls and constraints/requirements for the data. A sonification technique is the abstraction of the sonification design.

Note that a sonification can also consist of multiple sonecules that work together to create a rich sound experience. An example for this is the combination of simple (as we call them) standard Parameter Mapping Sonifications (cf. Sec. 5.1) which can be used together to create more complex Parameter Mapping Sonifications.

A requirement for using a set of sonecules together is that they happen in the same *context*: if we look at the visualization domain it is obvious that multiple single point marks combined in the same space can construct a scatter plot. The shared context of a *figure* – i.e., shared space – binds the individual points together and lets the pattern of point clouds emerge. Likewise a shared time (or timeline, or temporal canvas) serves as context for the domain of sonification. This is in line with ideas from Enge et al. on time as substrate of sonification [44]. An explicit structure for dealing with and sharing/reusing time in sonecules is thus a logical and important step.

mesonic offers an explicit model of the time in form of the *Timeline* class and also implements the idea of the context as a place where the sound objects live, which can also be found in the Web Audio API [45]. This allows sound objects – which are called *Synths* in mesonic – to place so-called *Events* into the same Timeline. Events placed by the different Synths contain information about the control of the sounds which are scheduled in time through this. The actual sound generation is then triggered by the *Playback* object which sends the different Events to the actual *sound rendering backend* of the Context. A more detailed description of the concepts of mesonic such as Timeline and Events can be found in the introduction paper of mesonic [1]. The Timeline and Playback provide the capabilities needed to mix different sonecules and control the sonification by e.g. starting it at a certain time or interrupting it. It furthermore allows us to alter the sonification by temporarily disabling a certain sonecule, or by filtering the Events depending on the sonecule which generated them.

4. CONSTRUCTION OF A SONECULE

The irreducible core of a sonecule is a sonecule ID for identification and functions to temporarily filter or to remove all Events of the sonecule from the Timeline. Both features enable users to create two or more similar sonifications, listen to them separately or together and to remove individual sonecules from the Timeline object without traces. Furthermore, sonecules should check the availability of the necessary Synths in the backend and prepare them accordingly at construction. Additional service functions could be defined, for instance to enable offline rendering of a single sonecule.

Additionally many sonecules could offer further functions such as scheduling of Events at a specific time. However this requires that the required data are already present at the time of scheduling and thus hinders sonecules to use stream data as often produced in biofeedback applications or interactive sonification. To avoid any of alike artificial limitations, we have currently established three different kinds of basic sonification designs or *sonecule types*. Each type can be regarded as a software engineering pattern, which provides a template to define own sonification designs as sonecule. The following section introduces these types and describes their commonalities and differences. Please note that this list is not meant to be exhaustive but rather focuses on sonification designs that are commonly found.

Score-based Sonifications are sonifications which can be computed ahead of time based on all the input data. Typical

examples for this are *Discrete* and the *Continuous Parameter Mapping Sonifications* [46] which based on data spawn multiple (discrete) sound objects or alter an ongoing sound stream (continuously) over time. These kinds of sonecules typically allow the definition of a *schedule* function which will create all Events in the Timeline ahead of time based on the data to be sonified.

Buffer Synthesis Sonifications are sonifications which do make use of buffers filled with data values to control parameters or define sound signals directly, instead of controlling a synthesis via Events by initiative of the Python process running a sonecule. This is usually without alternative if the control rate of the timeline is too low, or if manipulations have to occur at audio rate directly. Many sound computing backends distinguish between a control rate and an audio rate – and shifting control from 'from the outside' to 'inside the synthesis' can simultaneously free resources and increase precision. The most basic example to justify Buffer Synthesis Sonification is Audification [47] – where usually a data-derived signal is stored in a Buffer on the backend for playback. For other examples think of many parameter mapping sonifications, such as those using data to modulate the instantaneous frequency of a sine wave (aka FM synthesis).

Trigger Sonifications are those usually event-like sonifications that depend on a condition to trigger its playback. The most straight-forward examples are alarms, warnings, the techniques being Earcons, Musicons, Auditory Icons, Parameterized Auditory Icons, yet also designs at the intersection to Parameter-Mapping Sonification, such as the AIB (auditory information buckets), introduced in [48]. And also Model-Based Sonifications [42] fall into this category as they can be setup and then triggered by user interaction.

5. SONECULES - BASIC STANDARD LIBRARY

The following sections introduces selected sonecules. Each sonecule is briefly described in terms of the sonification design and technique and the underlying requirements on the data.

5.1. Score-based Sonifications

Standard Discrete Parameter Mapping Sonification (SDPMSon) are sonifications which belong to the Parameter Mapping Sonification (PMSon) technique. The standard part of the name stems from the limitation of using a single synth for the parameter domain and the discrete part means that each data point (row of a data set in table form for example) creates its own sound event where individual parameters of the sound result from a mapping function applied to the respective data point. This allows users to sonify all kinds of high-dimensional data sets, i.e. data tables with N rows and d columns.

Standard Continuous Parameter Mapping Sonification (SCPMSon) are as well as SDPMSon sonifications which belong to the PMSon technique. The difference here is that only a single yet possibly multi-parameter continuous sound stream is generated which is then updated via the mapping from single data points along the playback. This usually limits the usage to data sets that can be sorted along a property such as time or location.

These two can be regarded as starting point for further more specific sonification designs. It is also intended that multiple instances of the aforementioned sonecules can be used together to create more sophisticated Parameter Mapping Sonification designs, which then could potentially be wrapped again into a new even more complex sonecule.

Other sonecules to be included as Score-based Sonifications that are more specific are the Vocal Mapping Sonification, a special case of SCPMSon, using a formant-based synthesis model [49]. And also the implementation of SingingFunctions sonifications which are another special case of SCPMSon that allow distinguishing function shapes as auditory gestalts by listening to the properties being sonified [50].

5.2. Buffer Synthesis Sonifications

Simple Audification (SA) is a sonecule for the most basic audification of a univariate data set (resp. time series). The sonecule offers a simple playback control (start, pause, rate, ...) and also allows certain signal conditioning capabilities such as resampling and time-scale modification. As always with Audification this requires sequence (e.g. time series) data with enough samples to deliver sufficiently long audio data.

Multivariate Audification is an extension of the SA sonecule which allows to use multivariate time series data for audification. It can be imagined as a bank of SA instances to play the different channels simultaneously, one-shot or looped, while allowing to toggle dimensions, mixing, and to set/-manipulate loop properties such as begin & end or center & width.

Interactive Audification is again an Audification but using Granular Synthesis, with, for instance, the `TGrains` UGen offered by SuperCollider, to free Audification from automatic time progression. This allows users to scan the plot and probe the current underlying patterns interactively as they control the time progression, e.g. by moving the Mouse pointer along the x-axis. Here it could again make sense to offer a simple (single channel) and multivariate (multichannel) version as sonecules to allow optimization in the backend, as unfortunately the TGrains UGen in SuperCollider only works with mono audio buffers.

Timbral Sonification is another sonecule of the Buffer Synthesis Sonification category but as example for a Parameter Mapping Sonification: The Timbral Sonification is created by mapping values of multivariate time series to corresponding amplitudes of harmonics to a fundamental frequency so that patterns manifest as timbre changes. The user can here control the assignment order of the different data columns to the fundamentals arbitrary or by assigning them based on the amount of variance in the column.

Time-variant Oscillator Bank Mapping is again a sonecule for multivariate time series. The idea here is that we create a pitch mapping on a variable number of independent oscillators. Mapping a data vector results in vector-component depending pitch deviations from center notes for each oscillator in the bank. Center pitches are set musically equidistant between a minimal and maximal MIDI note number. Possible controls are the assignment order of the different data columns to the oscillators, as well as the de-/activation

of additional features such as an excitatory mode, where component value changes over time would drive (excite) the amplitude, resulting in a sonification that accentuates moments of variation.

5.3. Trigger-based Sonifications

Event-based Sonification is a sonecule that allows to specify a condition function and a sound to be played when the condition is met. Examples for these sounds are Auditory Icons and Earcons which might be selected or parameterized based on the condition result on the input data.

Data Sonogram Sonification Model is a sonecule representing a basic example sonecule from the family of Model-Based Sonifications. The Data Sonogram is an example for excitatory sonification models where *punctual*, i.e., localized and instantaneous, injection of energy initiates a process in model space which – according to some dynamical laws – yields acoustic reactions representing the sonification. In case of the data sonogram, the excitation is the triggering of a 'shock wave' in data space which expands isotropically and causes distance-to-data-points related activation of their acoustic contributions. Adjusting the shock wave speed appropriately, a spatial scan may last few seconds, allowing to quickly and intuitively tap and explore data in a closed-loop discrete interaction pattern.

As sonecules are implemented using object-oriented programming methods, the class hierarchy and inheritance offers us to naturally create a Model-Based Sonification sonecule class as it's own pattern where derived sonecules will be subclasses that share for example the triggering interface or plotting function, but replace the inner model logic to create for instance the GNG-Sonification or Data Crystallization Sonification model [42] as their own and novel sonecules.

Proto-Sonification / Auditory Canvas is a family of sonecules that can be used for providing context to the sonification by allowing the auditory representation of begin/end markers and ticks as discrete sounds or by using for example a continuous sound as a reference line. Even labels and the title could be added as spoken descriptions of the sonecule. These parts are useful extensions in general and in the case of spoken descriptions especially aim to improve the sonification for visually impaired persons. They can be differentiated from the Event-based Sonifications as they are not based on a specific data sample but depend on overall characteristics of the sonification such as the duration.

6. THE SONECULE INTERFACE

The following section provides an overview of the object-oriented interface of sonecules and how it can be used. Each sonecule is a class. This enables to use inheritance of the basics from the sonecule base class and also enables to define families of sonecules which share common function such as for example Model-based Sonification sonecules that can reuse the functions for integrating the interaction into the sonifcation model.

The following properties and functions are part of the sonecule interface:

construction enables the user to define the audio context and therefore the timeline that is used for the sonification. If no specific audio context is provided a default one will be used. This allows adding multiple sonecules to the same timeline and creating a complex sonification out of simpler units. The construction of the sonecule also initializes the necessary synths and buffer instances and allows the user to provide first parameters like for example an overall amplitude factor of the sonification or a target domain for a mapping.

sonecule ID each sonecule has a unique ID which allows to filter events tagged with this ID from the sonecule temporarily using the **active** property of the sonecule. Additionally a sonecule offers a **reset** function to remove its events from the timeline completely.

update the controls of a sonecules are accessible via the update function which allows for example to specify updated mapping parameters, to change the selection of data for the sonification or simply to alter the amplitude.

schedule allows to specify the time point where the sonification should start and what data are used. Note that Trigger Sonifications typically can't be scheduled as they depend on real-time data – however if recordings of such interactions are available, a sonecule could provide a **schedule** function were the recorded data can be provided to enable re-listening these interaction while changing the controls of the sonecule.

The following code demonstrates how sonecules can be used in pseudo code where the single command lines are meant to be executed interactively in a Jupyter Notebook or using another REPL.

```
# Create a sonecule using its constructor
sonecule1 = SoneculeXY(**sonecule1_controls)

# Schedule it at the beginning of the timeline
sonecule1.schedule(0.0, data)

# Listen to the sonfication using the playback
pb = sonecules.get_playback()
pb.start()

# Update controls and re-listen
sonecule1.update({"amplitude": 0.4})
pb.start()

# Create and schedule another sonecule
sonecule2 = SoneculeYX(**sonecule2_controls)
sonecule2.schedule(0.0, data, params)

# We can now listen to both sonecules at once
pb.start()

# Deactivate one sonecule temporarily
sonecule1.active = False
pb.start()

# Reschedule sonecule1 after sonecule2
sonecule1.schedule(sonecule2.duration + 0.5, ...)
sonecule1.active = True

# Listening to the sonecules one after the other
    in a loop
pb.loop = True
pb.start()
```

```
# Stop listening using the playback
pb.stop()
```

A demonstration and media samples can be found online in the supplementary material https://doi.org/10.4119/unibi/2979096. Additionally we keep lots of details to the Jupyter notebooks that illustrate how to use the sonecule, which are provided on our GitHub[1]

7. DISCUSSION

In this paper we presented *sonecules* as well-defined sonification designs packaged in Python to be quick-and-easy to use. Sonecules is implemented in Python as it is a very user-friendly ecosystem with great community support [51] and enables researchers from diverse domains to use their favorite data processing tools together with sonecules. It may look at first sight like sonecules only targets users that are new to the domain of sonification and are looking for a starting point to sonify their data. However we argue that also experts and the auditory display community benefit from the ideas behind and the availability of sonecules, for the following reasons:

(a) **flexibility** and **portability**: we have designed sonecules so that it is platform- and backend-independent, which should enable everyone to use it. sonecules rely on Python and (via mesonic) on different backends so that it may run on most computer systems (for sure the big OS Windows, MacOS and Linux, but also potentially on embedded systems), and as the different backends are capable of using the same sonecules this could make it straightforward to bring sonification to where ever it will be needed.

(b) **availability** and **distribution**: The distribution of sonifications is especially hard [17] as we are obviously not able to print the sonifications and even audio files are limiting as they do not allow to use the sonification with compatible data sets and altered parameters. sonecules allow the distribution of sonification designs as reusable package. Our goal is that the collection of sonecules will continue to grow as other sonification researchers join us to distribute their own sonification designs as sonecules, so that the toolbox gets ever fuller for the benefit of all.

(c) **reproducibility**: relying on sonecules for sonification would add the benefit of being able to better share the experience and interaction (and not only the ideas and sounds as it is commonly done in papers). Anyone could more easily reproduce the sonifications of others, as starting point for their own improvements. But also other user groups such as lecturers and data journalists could use sonecules to quickly introduce students or readers to the idea of sonification using data relevant to them. This could help the sonification community to increase awareness.

(d) **benchmarking, enhancing scientific standards and development**: comparing one's own sonifications to the state-of-the-art is currently practically impossible because to do so, all candidates must be reimplemented, yet papers often lack critical details to do so, let alone the resources to do it. Publishing sonification designs as sonecules solves that problem and hopefully enhances research practices as new researchers can start from others' work.

(e) **dissemination** for decades we long for the 'killer application' which irrevocably kills any doubt against sonification. The number of such experiments could grow steeply, having a package that application-domain researchers in every walk of science and industry could quickly and easily use, and thereby both the acceptance of auditory display practices and the chances to collect 'killer applications'.

We provide sonecules as initial idea and as a collection of sonification designs. We hope that the sonification community will use these as starting point for many more sonification designs. We are looking forward to further discussions on GitHub around each sonecule and hope sonecules provides a platform for these discussions, as it is a community effort to represent the complete sonification domain and set suitable default parameters for sonification designs.

In our ongoing work we will introduce more sonecules as well as likely upstream improvements for the mesonic library. The introduction of alternative sound synthesis backends seems to be a crucial step towards the uptake as an open platform and it has the potential to allow sonecules to be used in the web browser without the need to re-implement the logic and data processing.

8. CONCLUSION

Sonecules provide multiple sonification designs in a single interface that is more accessible than prior Python libaries such as sc3nb [37] or mesonic [1], yet it still integrates very well into the vast ecosystem that can be found in Python around data handling and processing. The design-once-use-many approach aims to scaffold method and application development and rapid prototyping of sonification experiments. This is beneficial for both, experts and new users of sonification, as it follows the Unix principle and focuses on providing sonifications as an clearly-defined interface which is appealing for end-users who want to get started quickly. Additionally, with sonecules we also introduce a new taxonomy that goes deeper, i.e., is more specific than the already known taxonomy of sonification techniques, and it aims at providing software engineering patterns for sonification. This should in turn allow a more concise development of new sonification designs, because a sonecule that facilitates its distribution will be quickly reusable and reproducible by peers.

All in all, sonecules certainly needs time to mature and – as always in open-source – the project's success depends on the reception and adoption of the community. sonecules is open to contribution and aims to provide an exchange platform for sonification designs with the potential to seed resp. grow into a standard library of sonification.

9. REFERENCES

[1] D. Reinsch and T. Hermann, "Interacting with sonifications-The mesonic framework for interactive auditory data science," in *Proceedings of the 7th Interactive Sonification Workshop (ISon 2022)*. Hanse Wissenschaftskolleg (HWK), Delmenhorst, Germany: Zenodo, Jan. 2023, pp. 65–74, publisher: Zenodo.

[2] S. Barrass, "Personify: A Toolkit for Perceptually Meaningful Sonification," in *Proceedings of the Australian Computer Music Association Conference 1995*, Melbourne, July 1995.

[3] C. M. Wilson and S. K. Lodha, "Listen: A Data Sonification Toolkit," in *Proceedings of the 3rd International Con-*

[1] sonecules GitHub https://github.com/interactive-sonification/sonecules

ference on Auditory Display (ICAD 1996), Palo Alto, California, USA, 1996.

[4] S. K. Lodha, J. Beahan, T. Heppe, A. Joseph, and B. Zane-Ulman, "MUSE: A Musical Data Sonification Toolkit," in *Proceedings of the 4th International Conference on Auditory Display (ICAD 1997)*, Palo Alto, California, USA, 1997.

[5] A. J. Joseph and S. K. Lodha, "Musart: Musical audio transfer function real-time toolkit," in *Proceedings of the 8th International Conference on Auditory Display (ICAD 2002)*. Kyoto, Japan: Georgia Institute of Technology, July 2002.

[6] O. Ben-Tal, J. Berger, B. Cook, M. Daniels, G. Scavone, and P. Cook, "SonART: The Sonification Application Research Toolbox," in *Proceedings of the 8th International Conference on Auditory Display (ICAD 2002)*, Kyoto, Japan, 2002.

[7] J. A. Miele, "Smith-Kettlewell display tools: A sonification toolkit for Matlab," in *Proceedings of the 9th International Conference on Auditory Display (ICAD 2003)*, Boston, Massachusetts, USA, 2003, pp. 288–291.

[8] B. N. Walker and J. T. Cothran, "Sonification Sandbox: A graphical toolkit for auditory graphs," in *Proceedings of the 9th International Conference on Auditory Display (ICAD 2003)*, Boston, Massachusetts, USA, 2003, pp. 161–163.

[9] S. J. Cantrell, B. N. Walker, and Ø. Moseng, "Highcharts Sonification Studio: An Online, Open-Source, Extensible, and Accessible Data Sonification Tool," in *Proceedings of the 26th International Conference on Auditory Display (ICAD 2021)*. virtual: International Community for Auditory Display, June 2021, pp. 210–216.

[10] A. de Campo, C. Frauenberger, and R. Höldrich, "Designing a Generalized Sonification Environment," in *Proceedings of the 2004 International Computer Music Conference (ICMC 2004)*. Miami, Florida, USA: Michigan Publishing, 2004.

[11] F. C. Ciardi, "sMax: A Multimodal Toolkit for Stock Market Sonification," in *Proceedings of the 10th International Conference on Auditory Display (ICAD 2004)*. Sydney, Australia: Georgia Institute of Technology, 2004.

[12] S. Pauletto and A. Hunt, "A Toolkit for Interactive Sonification," in *Proceedings of the 10th International Conference on Auditory Display (ICAD 2004)*, S. Barrass and P. Vickers, Eds. Sydney, Australia: International Community for Auditory Display, 2004.

[13] J. W. Bruce and N. T. Palmer, "SIFT: Sonification Integrable Flexible Toolkit," in *Proceedings of the 11th International Conference on Auditory Display (ICAD 2005)*, Limerick, Ireland, 2005, pp. 256–259.

[14] R. M. Candey, A. M. Schertenleib, and Wanda L. Diaz Merced, "xSonify Sonification Tool for Space Physics," in *Proceedings of the 12th International Conference on Auditory Display (ICAD 2006)*, London, UK, 2006, pp. 289–290.

[15] B. Garcia, W. Diaz-Merced, J. Casado, and A. Cancio, "Evolving from xSonify: A new digital platform for sonorization," *EPJ Web of Conferences*, vol. 200, p. 01013, 2019.

[16] D. Worrall, "Overcoming software inertia in data sonification research using the SoniPy framework," in *Proceedings of the International Conference on Music Communication Science (ICoMCS)*. Sydney, Australia: Citeseer, 2007, p. 180.

[17] F. Dombois, O. Brodwolf, O. Friedli, I. Rennert, and T. Koenig, "SONIFYER: A Concept, a Software, a Platform," in *Proceedings of the 14th International Conference on Auditory Display (ICAD 2008)*, Paris, France, 2008.

[18] K. A. Beilharz and S. Ferguson, "An Interface and Framework Design for Interactive Aesthetic Sonification," in *Proceedings of the 15th International Conference on Auditory Display (ICAD 2009)*. Copenhagen, Denmark: Re:New Digital Arts Forum, 2009.

[19] J. W. Walker, M. T. Smith, and M. Jeon, "Interactive Sonification Markup Language (ISML) for Efficient Motion-Sound Mappings," in *Human-Computer Interaction: Interaction Technologies - 17th International Conference*, ser. Lecture Notes in Computer Science, M. Kurosu, Ed., vol. 9170. Los Angeles, CA, USA: Springer, 2015, pp. 385–394.

[20] M. Jeon, M. T. Smith, J. W. Walker, and S. A. Kuhl, "Constructing the Immersive Interactive Sonification Platform (iISoP)," in *Distributed, Ambient, and Pervasive Interactions*, ser. Lecture Notes in Computer Science, N. Streitz and P. Markopoulos, Eds. Cham: Springer International Publishing, 2014, pp. 337–348.

[21] S. Landry, M. Jeon, and J. Ryan, "A broad spectrum of sonic interactions at immersive interactive sonification platform (iISoP)," in *Proceedings of the 21th IEEE Virtual Reality Conference (Workshop in Sonic Interactions in Virtual Environments)*, Minneapolis, Minnesota, USA, 2014.

[22] J. M. Cherston, "Auditory display for maximizing engagement and attentive capacity," Thesis, Massachusetts Institute of Technology, 2016.

[23] J. Cherston and J. A. Paradiso, "Rotator: Flexible Distribution of Data Across Sensory Channels," in *Proceedings of the 23rd International Conference on Auditory Display (ICAD 2017)*. State College, Pennsylvania, USA: The International Community for Auditory Display, June 2017, pp. 86–93.

[24] S. Phillips and A. Cabrera, "Sonification Workstation," in *Proceedings of the 25th International Conference on Auditory Display (ICAD 2019)*. Newcastle upon Tyne: Department of Computer and Information Sciences, Northumbria University, June 2019, pp. 184–190.

[25] H. Lindetorp and K. Falkenberg, "Sonification for Everyone Everywhere - Evaluating the WebAudioXML Sonification Toolkit for Browsers," in *Proceedings of the 26th International Conference on Auditory Display (ICAD 2021)*. virtual: International Community for Auditory Display, 2021, pp. 15–21.

[26] H. Lindetorp and K. Falkenberg, "WebAudioXML: Proposing a new standard for structuring web audio," in *Proceedings of the 17th Sound and Music Computing Conference*. Torino, Italy: Zenodo, June 2020, pp. 25–31.

[27] H. Lindetorp and K. Falkenberg, "Audio Parameter Mapping Made Explicit Using WebAudioXML," in *Proceedings of the 18th Sound and Music Computing Conference*, 2021.

[28] H. Lindetorp and K. Falkenberg, "Putting Web Audio API to the test: Introducing WebAudioXML as a pedagogical platform," in *Proceedings of the International Web Audio Conference*, ser. WAC '21, L. Joglar-Ongay, X. Serra, F. Font, P. Tovstogan, A. Stolfi, A. A. Correya, A. Ramires, D. Bogdanov, A. Faraldo, and X. Favory, Eds. Barcelona, Spain: UPF, July 2021.

[29] J. McCartney, "SuperCollider: A new real time synthesis language," in *Proceedings of the 1996 International Computer Music Conference (ICMC 1996)*. Hong Kong: Michigan Publishing, 1996, pp. 257–258.

[30] M. Puckette, "Pure Data: Another integrated computer music environment," in *Proceedings of the Second Intercollege Computer Music Concerts*, Feb. 1970, pp. 37–41.

[31] M. Puckette, "Combining Event and Signal Processing in the MAX Graphical Programming Environment," *Computer Music Journal*, vol. 15, no. 3, pp. 68–77, 1991.

[32] B. Vercoe, "Csound: A manual for the audio processing system and supporting programs," Technical report, MIT Media Lab, Tech. Rep., 1986.

[33] A. de Campo, C. Frauenberger, and R. Höldrich, "SonEnvir - a Progress Report," in *Proceedings of the 2005 International Computer Music Conference (ICMC 2005)*. Barcelona, Spain: Michigan Publishing, 2005.

[34] A. de Campo, C. Dayé, C. Frauenberger, K. Vogt, A. Wallisch, and G. Eckel, "Sonification as an Interdisciplinary Working Process," in *Proceedings of the 12th International Conference on Auditory Display (ICAD 2006)*, London, UK, 2006, pp. 28–35.

[35] A. de Campo, "An Interdisciplinary Approach to Sonification of Scientific Data," Ph.D. dissertation, 2009.

[36] David Worrall, Michael Bylstra, Stephen Barrass, and Roger Dean, "SoniPy: The Design of an Extendable Software Framework for Sonification Research and Auditory Display," in *Proceedings of the 13th International Conference on Auditory Display (ICAD 2007)*, Montréal, Canada, 2007, pp. 445–452.

[37] T. Hermann and D. Reinsch, "Sc3nb: A Python-SuperCollider Interface for Auditory Data Science," in *Proceedings of the 16th International Audio Mostly Conference (AM '21)*. virtual/Trento Italy: ACM, Sept. 2021, pp. 208–215.

[38] S. Lenzi, P. Ciuccarelli, H. Liu, and Y. Hua, "Data sonification archive," http://www.sonification.design, 2020.

[39] S. Barrass, "EarBenders: Using stories about listening to design auditory interfaces," in *Proceedings of the First Asia-Pacific Conference on Human Computer Interaction APCHI*, vol. 96, 1996.

[40] S. Barrass, "Sonification Design Patterns," in *Proceedings of the 9th International Conference on Auditory Display (ICAD 2003)*, Boston, Massachusetts, USA, 2003, pp. 170–175.

[41] O. M. Ritchie and K. Thompson, "The UNIX time-sharing system," *The Bell System Technical Journal*, vol. 57, no. 6, pp. 1905–1929, July 1978.

[42] T. Hermann, "Model-based sonification," in *The Sonification Handbook*, T. Hermann, A. Hunt, and J. G. Neuhoff, Eds. Berlin, Germany: Logos Publishing House, 2011, ch. 16, pp. 399–427.

[43] T. Hermann, "Taxonomy and Definitions for Sonification and Auditory Display," in *Proceedings of the 14th International Conference on Auditory Display (ICAD 2008)*, Paris, France, 2008.

[44] K. Enge, A. Rind, M. Iber, R. Höldrich, and W. Aigner, "It's about Time: Adopting Theoretical Constructs from Visualization for Sonification," in *Proceedings of the 16th International Audio Mostly Conference (AM '21)*, ser. AM '21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 64–71.

[45] "Web Audio API," https://www.w3.org/TR/webaudio/, 2021.

[46] F. Grond and J. Berger, "Parameter mapping sonification," in *The Sonification Handbook*, T. Hermann, A. Hunt, and J. G. Neuhoff, Eds. Berlin, Germany: Logos Publishing House, 2011, ch. 15, pp. 363–397.

[47] F. Dombois and G. Eckel, "Audification," in *The Sonification Handbook*, T. Hermann, A. Hunt, and J. G. Neuhoff, Eds. Berlin, Germany: Logos Publishing House, 2011, ch. 12, pp. 301–324.

[48] T. Hermann, M. H. Hansen, and H. Ritter, "Sonification of Markov Chain Monte Carlo Simulations," in *Proceedings of the 7th International Conference on Auditory Display (ICAD 2001)*, Espoo, Finland, 2001, pp. 208–216.

[49] T. Hermann, G. Baier, U. Stephani, and H. Ritter, "Kernel Regression Mapping for Vocal EEG Sonification," in *Proceedings of the 14th International Conference on Auditory Display (ICAD 2008)*, Paris, France, 2008.

[50] F. Grond and T. Hermann, "Singing Function: Exploring Auditory Graphs with a Vowel Based Sonification," *Journal on Multimodal User Interfaces*, vol. 5, no. 3-4, pp. 87–95, May 2011.

[51] J. M. Perkel, "Programming: Pick up Python," *Nature*, vol. 518, no. 7537, pp. 125–126, Feb. 2015.