SodaLib: a data sonification framework for creative coding environments

Agoston Nagy
Moholy Nagy University of Design and Arts
Zugligeti út 9
Budapest, Hungary, 1121
stc@binaura.net

Abstract

This paper introduces SodaLib, a sonification framework made for different programming languages [1]. It's aim is to let creative practitioners turn data into sound as easily as possible without interfering their regular workflow and without the need of advanced musical knowledge. SodaLib proposes generic and flexible ways of working with many types of data representations, including measurements by sensors, real-time data feeds, pre-recorded data, or even game and other interaction-driven temporal events. The sound engine is made with Pure Data using LibPd [2] and it can be embedded into applications running on many operating systems from desktop to mobile, including iOS, Android, Raspberry (ARM linux).

Keywords

Data Sonification, LibPd, Creative Applications, Extending Pd

1. Introduction

In the ever-growing area of data driven interfaces (embedded systems, social activities), it becomes more important to have effective methods to analyze complex data sets, observing them from different perspectives, understanding their features and dimensions, accessing, interpreting and mapping them in meaningful ways. With SodaLib, it is easy to map live data (sensors, web apis, etc), large, prerecorded datasets (tables, logs, excel files), or even unusual sources (images, 3D environments) to recognizable audible patterns through a set of sonification methods, including parameter mapping and event based sonification [3].

SodaLib is an open source, cross platform, multipurpose sonification tool for designers, programmers and creative practitioners.

2. Motivation

A personal motivation for creating this library is coming from the discussions of different workshops and courses on creative coding practice, where sound is used in novel ways for communication [4]. By discussing topics on sound, game engines, sonification, VR environments, data transmissions using different sensor systems, it turned out that there is a general need for a multipurpose tool which can be embedded with a few lines of code into the regular workflow of people coming from different backgrounds, letting them focus on application creation combined with creative sound production.

Another, more general need for a high level framework is the lack of well defined structures for representing data with sounds. While visualization has a great and extensive history of representing data through lines, shapes, colors based on cognitive and gestalt elements that are rooted in our visual perception [5], sonification lacks this common language structure and it is deeply overwhelmed with musical concepts and/or advanced synthesis systems that are hard to understand for people who primarily work with interaction and data instead of producing sounds. SodaLib provides an easy way to sonify data from one dimensional discrete events to multidimensional, continuous data streams with just a few lines of code.

The framework does not follow the tradition of most available sound libraries that are about to build sequencers, soft-synth like studio equipments and similar tools that are based on musical symbolic structures. Instead, the system is dealing with data as it

is: flexible, neutral entities in multidimensional contexts. These elements can be sonified in similar ways among all dimensions, regardless of the underlying sound engine construction. The dimensions can be selected based on the relevant contexts of the data, they can be addressed using interchangable values, such as shift in pitch, power in volume and position in pan (including binaural 3D positioning), depth as reverberation size, etc. The dimensions of the system can be extended very easily if needed by adding custom generators and/or mapping functions. However, at the moment, these few parameters seem to fit the basic needs when dealing with human cognitive efforts and the regular perception conditions of the mind.

3. LibPd

SodaLib's core functions are written in Pure Data (vanilla), it is fully compatible with LibPd. LibPd is platform agnostic in terms that it does not rely on any audio API calls directly, instead it can be embedded into many environments with a few binding functions to access low level API calls. It can be easily adapted into many creative coding environments, including c++, java, python, targeting platforms from desktop to iOS, Android, Raspberry PI or server based applications.

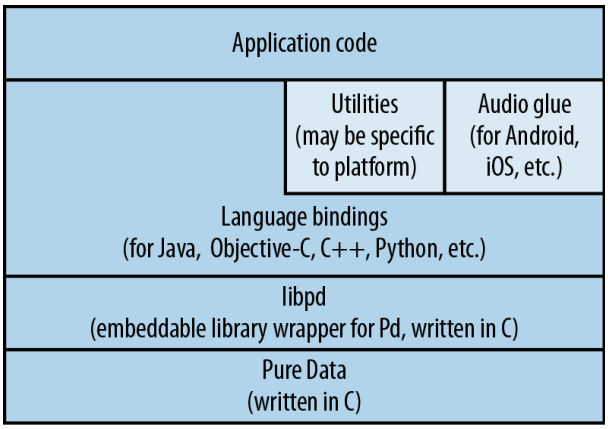

Fig. 1: Layers of LibPd. From the book Making Musical Apps by Peter Brinkmann. Chapter 4, fig. 4-1

When working with different creative coding frameworks within these environments, one can access and use the core Soda objects through a thin, streamlined language specific interface layer, without the need to know how to use Pure Data itself or things like synthesis and musical structures. Instead, users can stay with their preferred environment, keeping focus on application creation: representing data and designing interaction.

4. Implementation

4.1 Architecture of the sound engine

SodaLib consists of a sound engine that is written in Pure Data and a few utility functions that are available within the application code. The Pure Data part is a set of vanilla abstractions that are divided into some utility functions and building blocks. These blocks are dynamically generated sound creators. At the moment, polyphonic sampling, synthesis with basic, yet extendable waveforms and a noise filtering block have been implemented, since these can be applied well for different type of data representations. Please refer to the examples for more on which sound block is best for what type of data.

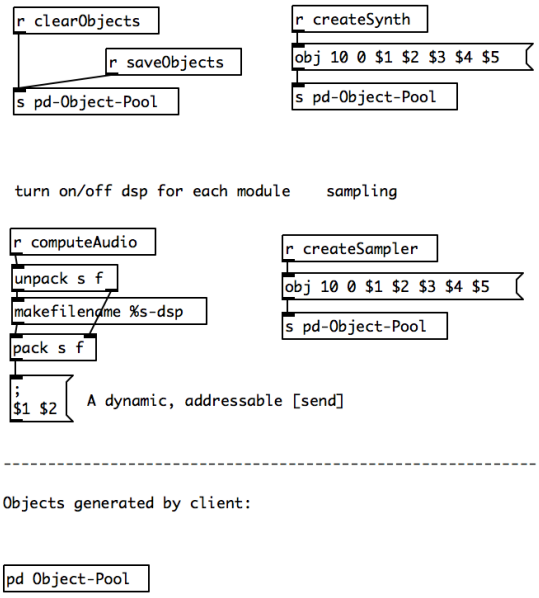

Fig. 2: Part of SodaLib's main.pd patch. Lists received from the client code are interpreted as object creation arguments to Pd. Each object can be addressed and manipulated during runtime, routing is based on it's name

The internal, flexible modularity is made possible by dynamic patching. This means, the building blocks are generated and saved on the fly, the interconnections between the

different elements are made without chords, they are based on a simple internal addressing system, where each dynamically generated abstraction can be addressed via it's name that it has got during creation time.

The benefit of this concept is that the objects can be created and addressed directly from the application code, which is completely different from the sound generator code. This idea is based on the idea of libPd, where values can be sent to the sound engine via messages: if a function that sends data to libPd has a receiving pair, the data can be processed within LibPd.

SodaLib extends the concept of this system to a more abstract level: the readymade, complex sound rendering objects can be created via simplified message packages that are received in Pure Data.

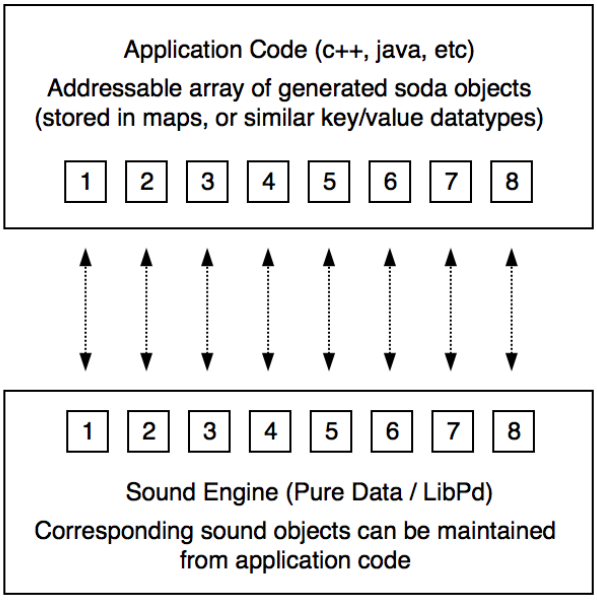

Fig. 3: Each Soda Object in the application side has a corresponding sound object in Pd. Modifying parameters of an object in the application code affects directly its sound generator in the Pd side.

4.2 Binding functions and object management in different languages

Since the messages addressed to the sound generators can be accessed via simple lists on the Pd side, the framework has some binding functions on the application side. All these functions are acting as a communication layer that can take some parameters, make a list from them and forward it to the sound engine. The code is taking care of the already existing Soda objects beyond these functions: when generating an instance, each object is put into an easily addressable data container (HashMap in Java, Map in C++) where objects can be addressed as key / value pairs. Keys are the names of the objects and the values are the referred objects themselves. This comes really handy when working with multiple objects at once. For example, if someone has a larger array of data, it is easy to generate a sound object for each data point in the array and address its corresponding sound object based on the index of the data in the array.

4.3 Example application code

Creating and manipulating custom sound objects can be made with just a few lines of code. While the following examples are taken from the syntax of C++ (OpenFrameworks [6]), similar functions and container datatypes can be used with other coding frameworks, such as Java (Processing), Python, Swift, C# (Unity), etc. In the first example, we create a custom polyphonic sampler object and change its pitch. First, initialize Sodalib. This function creates the sound engine and loads the main puredata patch to the system:

```
soda.init();
```

Then, create a sound generator, with the name *mySample*, loading the file *sound.wav*, with polyphony of 10. This last parameter is depending on the loaded patch type, it can have different meaning when making synths or other generator objects.

```
soda.createSampler("mySample","sounds/
sound.wav",10);
```

Later on, when the program runs, the sound object can be manipulated by the following code:

```
soda.set("mySample")->shift(0.5)
->play();
```

Here, the *shift* value of the object is modified. 1 is original speed, 0.5 is half speed, etc.

A more interesting example would be to create several objects and control them based on incoming data values. Let's create a lot of soda objects now, this time some sonic texture generators

```
for(int i=0; i<100; i++) {
       soda.createTexture("soda-" +
       ofToString(i),60);

       soda.set("soda-" + ofToString(i))
       ->shift(i/100);
}
```

Now, we have 100 objects. These are generating noise with a bandpass filter (with a bandwidth of 60 Hz). Notice that we are using the *i* integer for naming and accessing our objects. This integer is parsed to a string using the *ofToString()* function. By changing their shift parameter after creation, each object will have different filter coefficents, that goes from 0 – 1. In SodaLib, ideally all function parameters should be between 0 – 1. This range is mapped to meaningful values in the sound objects. Volume goes between 0 (minimum) to 1 (maximum value), such as pan from 0 (left) to 1 (right), filter coefficient goes from 0 (0 Hz) to 1 (20 KHz) etc. This concept can be strange from a musical point of view at first glance, but this pattern is effective for keeping the parameter dimensions interchangeable and flexible.

We can access and manipulate any of the created objects by referring to their names. We can do it individually, say if we would like to access and manipulate only one object from our 100 instances:

```
soda.set("soda-42")->volume(1)->play();
```

Since the *set()* function returns a soda object, different parameters can be modified at the same time by making chains of the parameters, like

```
soda.set("soda-42")->volume(1)->pan(0.2)
->shift(2)->play();
```

Of course, we can set all of the objects together at once, based on existing values in an array or any data type. It's also possible to just loop through them and create different random pitches for them using another for loop:

```
for(int i=0; i< 100; i++) {
       soda.set("soda-" +
       ofToString(i))
       ->volume(ofRandom(1))->play();
}
```

There is another useful feature of the lib that can be applied for gaming and VR development. When a *pan()* function is called on an object, the type of the sound rendering is depending on the number of function arguments. If it has only one, the panning is done in a traditional, stereo left/right manner. However, by passing three parameters, the position of the sound can be controlled in three dimensions, by passing the azimuth, elevation and distance to the desired sound origin. This binaural sound positioning is using the *[earplug~]* external [7] for the math and modelling, and it can be really effective when developing games or 360° animation scenes.

5. Conclusion

SodaLib is a contribution for creative practitioners who are interested in making all type of realtime applications that involve sound generation without the need to know sound programming and/or advanced musical concepts. It can be used for very quick rapid prototyping, but since it is completely embeddable, it can also be used to target different professional digital publishing platforms [8]. Since data driven sound will be more common in the era of connected devices and embedded technology, the relevance for creating such frameworks is evident.

It can also help in teaching the concepts of interactive sound, visual sound instruments [9] and different techniques of sonification. On one hand, attention of newcomers from other creative disciplines and creative coding frameworks can be aroused to the directions of Pure Data and lower level DSP techniques, on the other hand, experienced pd-ers can learn how to build applications, publish their works in forms of different digital media, apart from the performance stage, installations or regular music production activities [10].

Last, but not least, a long-term, interesting topic is of course the emerging field of the theory and practice of sonification. While visualisation has its own literacy and well defined concepts with practical building blocks, sonification lacks most of these. With such tools, it would be easier to maintain discussions on the cognitive, psychoacoustical aspects of sounds that are representing data: which types of sonic structures are describing specific datasets more effectively?

6. Acknowledgements

Thank you for all the developers of Pure Data & LibPd, including Peter Brinkmann, Dan Wilcox and others, dynamic patching ideas of IOhannes Zmoelnig, the RJDJ crew and many others for their effort in the making, Gábor Papp, Bence Samu for helping in and discussing tons of interesting concepts, Réka Majsai, Adam Somlai-Fischer, Bill Manaris for having such inspiring discussions around and beyond the topic.

References

[1] https://github.com/stc/ofxSodaLib (implementation for OpenFrameworks)

[2] Brinkmann, Peter et al: Embedding Pure Data with LibPd. PdCon11: https://www.uni-weimar.de/medien/wiki/images/Embedding_Pure_Data_with_libpd.pdf

[3] Hermann, Thomas et al: Sonification Handbook. Logos Publishing Hous, Berlin, 2011 http://sonification.de/handbook/

[4] Sonic Instruments Workshop, ICAD, ISEA, 2015 http://www.binaura.net/stc/ws/isea/

[5] Fry, Ben: Visualizing Data. O'Reilly, 2007

[6] http://openframeworks.cc/

[7] Earplug External, a binaural filter based on KEMAR impulse measurement for Pd https://puredata.info/downloads/earplug

[8] Brinkmann, Peter: Making Musical Apps. O'Reilly, 2012

[9] Nagy, Agoston: Visual Sound Instruments (Thesis), MOME, 2014

[10] Steiner, Hans Christof: Build your own instrument with Pd. In: BangBook, pd-graz, 2006 https://puredata.info/groups/pd-graz/label/book/bangbook.pdf