Why Snd is in My Toolbox

Juan Reyes
Maginvent.ORG
juanig@Maginvent.ORG

December 19, 2011

Abstract

This paper presents reasons to see *Snd* as a good composition tool, with features not often provided in other sound editors and commercial music software. Historical evolution of computer music and research through the years, unfold ideas and elements behind *Snd's* design and prove its functional goals. A time-line of accomplishments at CCRMA are furthermore plentiful justification as to why *Snd's* development has been steered the way it has. Here various aspects of *Snd's* evolution are described, in addition as to why it is a complete music programming language, bringing out the most of algorithmic composition and parametric manipulation to computer aided composition.

1 Introduction

Choices in digital music processors and software synthesizers abound these days. Having used some of them, including those with sequential interaction (not so real time), and concurrent interaction (real time and instantaneous feedback), from a composer's standpoint, each category serves its own purpose. Sequential interaction software synthesis packages are now narrowed to a handful namely Music-N style packages like *Csound*[21] and *CLM-4*[13]. Concurrent interactive systems are more spread nowadays, and include real time packages like Pd[10], and SuperCollider[6]. In addition to the mentioned categories, studio and performance situations have also a plethora of sound play-lists and waveform editing applications, ranging single file processing to multitrack packages for overlaying and mixing sound files at once to stereo and multi-phonic mixing. Mixing applications like *Ardour* [3], also add the benefit of real time digital signal processing by means of plug-ins and extensions to the actual main application.

Advantages or disadvantages to real time and not real time systems can be narrowed to goals, ideas and objectives arising on each challenge posed to a composer while working a particular piece. In an attempt to categorize a range of usages, of course, there are performance applications which require some type of input and an instantaneous feedback device. There are also "studio devices" which appropriate sequential interaction, whereby composers use a system to manually edit sound waveforms, and experiment, working to algorithmically generate note lists. In Electroacoustic Music composition, studio work implies production of "Tape Music" pieces usually presented on arrays of loudspeakers in halls lacking live performance. The evolution of computer music systems brings now an option of a choice.

Snd [18], developed through the years by Bill Shottstaedtat CCRMA, follows a studio and sequential interaction scheme, where composers generate note lists by means of algorithms and function calls by using almost all known sound synthesis methods. However, *Snd* provides the capability of manually editing and processing of a waveform by providing usual cut-and-paste, and copy, click mouse functions, as well as time domain envelope functions for editing sound file's amplitude and other parameters. *Snd* also features sound synthesis options and note-list generation which can be accomplished by programming high level languages like Scheme, Ruby, and Forth. For historical and compatibility reasons, this paper focuses on using Scheme within *Snd* . Note lists are here referred as a sequence of time ordered statements. By convention, each of such statements specify one complete sound event, with the name of an instrument followed by the onset time and duration of the event that the instrument is to perform. In more elaborate statements, a series of fields containing parametric information specify details for the event, complementing basic note-lists.

Snd's instruments are user-defined and consist of a growing set of signal processing elements that are configured to perform specific signal processing functions like oscillators, which produce a variety of periodic signals, modifiers, which deal with various techniques to modify a signal. furthermore delays provide means for reverberation and other procedures. Interconnections of all processing elements is done at a software level, using Scheme high level features. Instrument calls on note lists behave just as regular function calls on ordinary Scheme programming.

2 Snd Evolution Feedback

Most of *Snd* development and its state of the art can be traced to the evolution of Computer Music and all of its branches since 1960. *Snd* inherits from knowledge built at CCRMA from the days the center was located at the Stanford Artificial Intelligence Laboratory (SAIL), up to now, running on portable computer machines as well as in major operating systems including Linux [18]. Since September 1996, *Snd* has been Open Source, with many people including musicians and engineers, contributing contributing to its development geared primarily by its author Bill Shottstaedt. *Snd* can be thought of as last steps taken in the development of Computer Music systems, regardless of being a non real time sequential interactive system.

2.1 SAIL: Beginnings of Computer Music Research at Stanford

To understand *Snd's* features, design and behavior, it is worth pointing the origins of computer music research at Stanford. In the 1960's, John Chowning with the help of David Poole, were able to implement Max Mathews' *Music-IV* program on an IBM 7090, and a DEC PDP-1 computers at SAIL [1]. The system at the time was a time sharing computer with a development environment to write software tools for music analysis and synthesis. This gave way to *"MUSIC"* written for SAIL, FAIL compilers, and FORTRAN by David Poole, Leland Smith, and Tovar in the late '60s. In 1972, Leland Smith and Tovar started development of the original "SCORE" program adding support for generating score-files in *Music-IV* . Following in the mid 70's, "MUS10" added Algol constructs to Poole's original "Music" synthesis program [12].

While the SAIL time sharing computer proved to be useful in few analysis and synthesis applications, it was not so good for generating compositional level musical works and acoustic research. In 1978, CCRMA acquired the first on-line computer system, designed by David Poole and Peter Samson of System Concepts in San Francisco. This digital synthesizer was affectionately known in the community as the "*Samson Box*". The principal purpose of its design was to address high computational bandwidth required for real time high quality digital audio signal processing[5]. The *Samson Box*was an elegant implementation of nearly all known, desirable unit generators in hardware form. Additive, Subtractive, and non-linear FM synthesis and wave-shaping were all supported [19]. For over a decade much music was composed on the *Samson Box*.

The music compiler in the *Samson Box*also made use of computer languages such as SAIL(based on Algol), and FAIL (its FORTRAN counterpart), mainly because of the Artificial Intelligence Laboratory. These compilers were tailored to accept to accept note lists as input, and generated output as time ordered sequential streams while controlling all processing elements that produced musical output. Making use of these compilers was not trivial for many users, no matter background. In response to this, in the early to mid 80's came *Pla* and other applications known as *DpySnd*, *Mixer* and *Edpla*, all written by Bill Shottstaedt. *Mixer* was a mixing scripter, *EdPla* a GUI-oriented view onto *Pla*. *DpySnd* was a sound editor and the predecessor of *Snd* . A lot of experimentation went into every aspect of these applications. Worth mentioning is development of a graphic user interface that at the time included the notion of a "window system" to tackle interaction with sound parameters [12]. *Pla* was a composing language with syntax borrowed from SAIL but it featured music oriented components filling compositional needs of users [15].

2.2 DpySnd and Pla: Windows to the Samson Box

Many of the structural components of *DpySnd* and *Pla* made it and are distilled in *Snd* . The idea of having a general purpose programming language like the Algol part of SAIL still persists as the Scheme part of *Snd* . This takes advantage of incorporating a vast body of practical knowledge and men-hours into languages[17], and furthermore, code ensures sustainability and continuity of programming thoughts and ideas that can be reused just by grasping intricacies of a programming language.

Pla consisted of a variety of "musical macros" including pitch names, rhythmic values, motives, transposition values, among others. These macros (or functions), translated musical values to numerical values understood by the musical compiler, but its purpose was to assist composers on creating note lists for their instruments[16]. Improvements over *MUS10* included "cyclic lists" that allowed streams of expressions to variables of same type. For example a cycle of pitches [GBD], as an argument to a macro meant musically generating an arpeggio. Square brackets translate as constituting a cyclic stream of an expression. Nowadays this feature is used nowadays on many music languages but in particular SuperCollider. *Pla* facilitated use of envelopes at any level of processing too, giving time domain control aspects of signal fluctuations [16]. In *Snd* as in almost all software synthesis packages, the use of envelopes is crucial for generating expression in compositional contexts.

2.3 Computer Music meets Personal Computers

After more than ten years the *Samson Box*era was overshadowed by a transition to personal computers, making its pass to the NeXT computer in the late 1980's. These machines were the first computers with a DSP signal processing chip dedicated to sound and music. In 1989, and for the sole purpose of developing a music workstation, NeXT Inc. hired Stanford graduate Julius O. Smith and composer David Jaffe to develop the NeXT Music Kit. This software was a system for building music, sound, signal processing and MIDI applications for the NeXT . The Music Kit unified MIDI and the *Music-IV* paradigm, thus combining real time live interaction (MIDI), with a variety of synthesis methods [19].

At dawn of personal computers NeXT pioneered the idea of a workstation using previous desktop metaphors of the *Xerox Star* and *Macintosh* computers. In addition to this concepts, multimedia was of prime concern and one of the reasons for having a DSP chip. This notion of sound and graphics worked in tandem to give users the flexibility of a real time "WYSIWYG", what you see is what you get and hear system. Aside from many desktop applications, NeXT computers were packaged with a development kit including C and Lisp language compilers. Later it also included Objective C for development of NeXT footprint applications. Among others, several computer music applications taking advantage of this developing environment were Michael MacNabb's *Ensemble* [7], Rick Taube's *Common Music* (CM)[20] , Perry Cook's Physical Modeling applications[2], and Bill Shottstaedt's *Common Lisp Music* (CLM)[17].

2.4 CLM: Lisp Evolution of a Music Language

As a sound compiler *CLM*provides a Lisp environment extended with a large number of unit generators such like in the *Samson Box*, and much more. Its interface is presented to the user as a Lisp interpreter where various kinds of expressions can be used to generate sound files. To take advantage of greater compilation speeds with the $DSP5600x$ processor in the NeXT , *CLM*instruments were optionally compiled and run either on Lisp software, in C code, or in DSP code. However, over the years processors caught up and surpassed $DSP5600x$ processor computational speeds. Good sound file compilation and rendering speeds are achieved just by using C code version of *CLM*'s compiled instruments. *CLM*is structured into several parts but to the user, couple of them stand out, one is the instrument definition section ($definstrument$), and the other is the note list section ($with-sound$). This followed tradition of first defining and instruments and then calling it with a note list [17]. Calling musical events in *CLM* was as simple as calling other functions in the Lisp environment.

3 Snd's Outstanding Features

Snd's user's interface might not appeal to experienced users because its behavior doesn't follow expectations found on other mainstream commercial editors, and therefore it is easy to to miss its power and usefulness. By default *Snd's* GUI interaction and manipulation provides measures for dealing with pointing, clicking and dragging its windows and widgets. But the fact of the matter is that far more control is available through its "Emacs-style" key combinations on a PC QWERTY keyboard. Inspiration for this kind of design came from *DpySnd* on the DEC PDP-10 computer. But above all, *Snd* was designed to function within CCRMA's signal processing and sound processing, Common Lisp Environment and *CLM*. Indeed for a time *Snd* was considered a graphic display front-end for *CLM*[8].

Like in *CLM*, it is true that in order to use *Snd* to its fullest potential users must learn its control interface, and acquire some proficiency in Scheme or its other languages [9]. This kind of interaction brings most of *Snd* features hidden as Scheme source code which can be loaded at any time. *Snd* from its roots has evolved within the Unix domain and in the context of traditional Computer Music previously described. Unlike its commercial counterparts, hidden gems wait curious users willing to read and compile its open source files. Suffice to say, this code contains examples of every subject matter related to the fields of sound signal processing. Little insight will unfold qualities that make *Snd* not only an ordinary sound editor but also a software synthesizer, and a composition environment plus an analysis tool capable of providing solutions in the spectral domain by using any of its transforms. While sound editing and spectral analysis work interactively and in realtime, synthesis and processing take time to render results.

In past years Bill Shottstaedthas been developing his own incarnation of a Scheme compiler named *s7*. Its implementation is intended as an extension language for other applications, primarily *Snd* and *Common Music*. Scheme was chosen as language for *Snd* because of its dependency on Motif's graphics library handled by *Guile*, which is also a Scheme environment. For most part *s7* has taken over *Guile* and improving on speed, performance and reliability. Most of *CLM*'s functionality works seamlessly in *s7*, without the need of a C compiler for speed optimization such as the one still found on *CLM*-4. Benchmark tests on some of *Snd's* instruments almost rival in speed its *CLM*counterparts [14]. Aside from running *s7* and *Snd* from the interpreter in the GUI, they can be run as standalone commands on a terminal shell, or as an Emacs inferior process, to add the functionality of a text editor and compilation environment. Instructions for customizing Snd are surely part of its documentation, but examples abound in its source code and home page[18].

4 Own Reasons for Using Snd

Aside from tradition, which plays a big role for not falling into traps of reinventing the wheel, *Snd's* features provide continuity, knowledge and experienced acquired by working with *CLM* for years. Studio work for Electroacoustic composers remains an option and a road to follow because of all detail into modeling and expression when going this path. While debating real time performance and interaction benefits, in contrast to rendering and freezing music aspects, both roads require different focus. Timbral and spectral search seems more appropriate on *Music-IV* descendant systems, and for historical as well as technological reasons already described, *Snd* reigns among others. *Snd* inheritance is always a good starting step on compositional and research projects. Some might find *Snd's* documentation a bit cryptic but getting use to is not a huge task. There are explanations, examples and anecdotes that assist novice users and experienced programming composers.

Snd is in my toolbox because it provides means to model musical events and processes using features of a high level programming language. For example a Model of Jean Claude Risset's "paradoxical sounds" [11], is only feasible if implemented on an algorithm, because of its numerical complexities. Several approaches can be used to tackle this challenge, but sometimes, all approaches need to be modeled in order to select the more suitable. Another example is note-list generation by means of chaotic and dynamical systems that depend on initial conditions. Several sets of initial conditions are needed in order to obtain good results on note-lists. Translating from Matlab (or rather Octave) to Scheme, in a sonification fashion, is just a matter of porting the code. *Snd* is a good case of Open Source software because there are examples of approaching an issue, either if someone else has tried it, or suggestions on how to solve it. This is the case on quite a bit of signal processing techniques. For example inspiration on using delay lines in exotic ways, like flanging and Doppler effects, can be found on *Snd's* documentation and code. Such is the case is Fernando Lopez-Lezcano's *Dlocsig* spatial diffusion implementation[4]. Among other reasons to justify *Snd's* usage is the fact that compositional ideas can be encapsulated in *s7's* algorithms for posterity.

5 Final Thoughts

Although real time computer music systems are common use today, for most part computer music research seems to have given way to the music industry in many respects. Live laptop performances abound all over the world, therefore attracting and misguiding people on what computer music quests should be. Many of these performances are the result of a combination of loops and rhythmic patterns, very often justified by the origins of "musique concrete". But loops are only a single aspect of computers and musical style. Other performers get more creative by using filters and basic synthesis techniques with limited timbrical explorations because beat is king. Consequently more reflection is needed on all respects regarding live performance issues and territories. Traditional "live electronics" is more than improvisation, loops, filtering and basic synthesis methods.

Performance means musical gesture and expression, although an old issue, in no way computer techniques should get in the way and constrain expressing. In this spirit a music software application should serve as means to get in a state of musicality whereas performers and composers are able to get ideas materialized. A software application (or rather language), should lead to insights and extremes that land into fresh boundaries of expression. As stepping stones, hidden treasures in *Snd's* directories await to be rediscovered and reused creatively for new proposals that might as well still pose demands on machine's processor power. Therefore, experimentation and hacker's spirit are needed to achieve new results while modeling and testing new ideas. For many composition is not normally a realtime process [16], but instead a delicate task of coming and going, validating and discarding ideas.

References

[1] John M. Chowning. Computer music: A grand adventure and some thoughts about loudness. In *Proceedings of the International Computer Music Conference (ICMC-93, Tokyo)*, pages 2–3. International Computer Music Association, 1993.

[2] Perry Cook. *Real Sound Synthesis for Interactive Applications.* AK Peters Ltd. , Natick MA., USA, 2002.

[3] Paul Davis. Ardour: Digital audio workstation. `http://ardour.org/`, 2011.

[4] Fernando Lopez-Lezcano. dlocsig: a dynamic spatial location unit generator for clm-2. `https://ccrma.stanford.edu/~nando/clm/dlocsig/`, 2011.

[5] D. Gareth Loy. Notes on the implementation of musbox: a compiler for the system concepts digital synthesizer. *Computer Music Journal*, 5(1), 1981.

[6] James McCartney. Supercollider: a real time audio synthesis programming language. `http://www.audiosynth.com/`, 2011.

[7] Michael McNabb. Ensemble, an extensible real-time performance environment. In *Proceedings of 89th Audio Engineering Society Convention.* AES, 1990.

[8] Dave Phillips. Developing and using snd: Editing sound under linux, part one. `http://linuxdevcenter.com/lpt/a/1251`, 2001.

[9] Dave Phillips. Developing and using snd: Editing sound under linux, part two. `http://linuxdevcenter.com/lpt/a/1293`, 2001.

[10] Miller Puckette. Software by miller puckette. `http://www-crca.ucsd.edu/\~msp/software.html`, 2010.

[11] Jean-Claude Risset. Paradoxical sounds. In Max Mathews and John Pierce, editors, *Current Directions of Computer Music*, pages 149–158. MIT Press, 1989.

[12] Bill Schottstaedt. Timeline of clm/cm development. `http://ccrma-mail.stanford.edu/pipermail/cmdist/2003-May/000970.html`, 2003.

[13] Bill Schottstaedt. Clm homepage. `http://ccrma.stanford.edu/software/snd/snd/clm.html`, 2011.

[14] Bill Schottstaedt. Snd 12.5. `http://ccrma-mail.stanford.edu/pipermail/cmdist/2011-September/006369.h%tml`, 2011.

[15] Bill Shottstaedt. Pla: A composer's idea of a language. *Computer Music Journal*, 7(1), 1983.

[16] Bill Shottstaedt. Pla, a computer music language. In Max Mathews and John Pierce, editors, *Current Directions of Computer Music*, pages 215–224. MIT Press, 1989.

[17] Bill Shottstaedt. Clm: Music v meets common lisp. *Computer Music Journal*, 18(2), 1994.

[18] Bill Shottstaedt. Snd manual. `https://ccrma.stanford.edu/software/snd/snd/snd.html`, 2011. Last visited on Dec 15, 2011.

[19] Julius O. Smith. Viewpoints on the history of digital synthesis. In *Proceedings of the International Computer Music Conference (ICMC-91, Montreal)*, pages 1–10. International Computer Music Association, October 1991.

[20] Rick Taube, editor. *Notes from the Meta level.* Swets Zeitlinger Publishing, The Nederlands, 2004.

[21] Barry Vercoe. Csound homepage. `http://www.csounds.com/about`, 2011.