IXI SOFTWARE: OPEN CONTROLLERS FOR OPEN SOURCE AUDIO SOFTWARE

Thor Magnusson
Creative Systems Lab
Department of Informatics
University of Sussex
Brighton, United Kingdom
T.Magnusson@sussex.ac.uk

ABSTRACT

Sound has been liberated. The 20th century freed music from various compositional constraints and ideologies, and during the last decade we have witnessed a transformation in the way sound is organized in terms of composition, production, distribution and consumption. This paper is concerned with the production part of this complex structure and will describe some of the ideas, experiments and conclusions of *ixi software* over the last years. ixi has been building open interfaces to be used with open source audio programming environments. We will go through which cultural and technological changes have affected our work and describe the situation of audio programming as it appears to us today, as inventors of virtual screen-based instruments.

Keywords
Musical interfaces, virtual instruments, screen based intelligent instruments, open source, audio programming, OSC, open protocols, free sound.

1. INTRODUCTION

Much has been written and speculated about free music and open source recently by people of all ranks ranging from enthusiastic hackers to conservative music industrialists. In perpendicular to this discussion, I would like to explore the state of free sound (free as in free jazz but not necessarily free beer) after the advent of open source programming languages for audio and the ever more popular OSC (Open Sound Control) protocol.[1]

Over the last few years, *ixi software* has been experimenting with creating prototypes for screen-based instruments to be used with open source programming languages. We wanted to bridge the gap between physical instruments and controllers and the abstract programming environments which people use in their work. The idea was that we could represent sound or sound processing with graphical objects placed in a two dimensional space on the screen. Such representation is helpful when organizing sound and working with the "hidden" and highly logical processes of the computer. Initially we created instruments that were closed applications, which musicians could use to import their own samples into and work with according to their own logic. Each instrument had its own mode of interaction design and explored different organizational structures. This paradigm worked for a while, but users continuously asked for further features and with the advent of the open source languages like Pure Data and SuperCollider[2], it became clear to us that we should change the way we build our software. Instead of a self-contained instrument, we began creating open controllers that would send out OSC messages to the chosen audio programming language. This way, the ixi software became a simple and intuitive controller defining the events happening in a more integrated and complex sound engine that is hidden from view in performance, but open for change in the stage of composition.

2. OPEN AND FREE ENVIRONMENTS

The world of open source is an incredibly powerful force that is not only changing the current media economy and power balances but also more subtle and intangible elements such as aesthetical values. People choose to work with open source software for many reasons – usually practical, economical and political reasons – and the most obvious characteristic of open source communities is the spirit of helpfulness, culture of sharing and the feeling of belonging to a community of like-minded people. There is always help to be found, an openness to suggestions for change in the software, various (usually) democratic methods for choosing which features become accepted, and collaborative efforts made to solve conceptual or software engineering problems.

For all artists the choice of a tool is also a choice of a style. They are limited by the instrument of choice but the question for the computer musician becomes: *who* is defining those limitations on my expression? Typically the musician has a certain "sound" or style, but is that defined

[1] http://www.cnmat.berkeley.edu/OpenSoundControl/

[2] See: www.puredata.org and http://supercollider.sourceforge.net

by the software that he or she happens to use? Would the style change significantly if they were to use another software? As the honest answer to that question is likely to be a positive one, many musicians today are moving over to music software that is more open and transparent than the commercial software packages, themselves defining their own work processes and the scope of their instruments. People want to be in control and set their own limits when working in the creative fields.

Open source programming environments for audio such as Pure Data or SuperCollider are good examples of free and aesthetically open platforms. Both languages are relatively low level concerning musical concepts, but high level in relation to the mathematics and DSP that are required to synthesize or manipulate audio signals. In fact, you would have to search long and well to find any traditional musical concepts in Pure Data or Super-Collider. There are objects that translate MIDI values to cycles-per-second or decibels to linear amplitude, but these are also electronic music concepts. These languages have been built on the premises of the functional needs of the task at hand – synthesizing or processing audio – and not on a cultural heritage like the western musical tradition. It is the hardware of the computer, lower level languages such as C or C++ and the programmers' ideas of programming styles and environments that define the structure of these composition tools, not normative ideas about what music is or should be. [2, 3]

Composing music using Pure Data or SuperCollider is clearly exciting. One starts with a white page: a text file in SuperCollider and an empty white patch file in Pure Data. Only your own imagination sets the limits of expression. There is no timeline there, no preprogrammed reverb or delay effects, no sounds or samples or instruments. You compose your own music or instruments using the objects available and if you find you cannot express something, you can either write the object or unit generator yourself or ask for it to be done on the appropriate mailing lists. The act of musical composition and instrument making has become the same activity – manifested through programming.

3. IXI CONTROLLERS

Using open source programming languages for music points the attention of the musician to a field that ranges from the level of signal processing and the microstructure of a composition (where you work with the texture of the sound) to the macrostructure of the music, i.e. how musical events happen in time. The sound programming environments under discussion are open and flexible for almost all known synthesis and compositional techniques we use today, but the feature that we have found missing is the power to develop interesting and visually pleasing graphical user interfaces as part of the audio work.[3] We are interested in the study of actions – the performance – of a musician when he or she is using an acoustic or digital instrument in a studio or live performance. Our reason to build those interfaces is not the visual design, but the *mode of interaction* which they embody. Each of the ixi applications has a different interaction structure, which allows the musician to do certain things – such as automation or adaptation – that might not be easily performed by using a different controller.

With the OSC protocol, we are now able to create controllers that are open in their application. These screen-based virtual controllers are pattern generators whose output can be mapped to all levels of musical composition from the micro- to the macro-level. On the contrary to most MIDI controllers, there need not necessarily be any musical reference to the things that are happening at the interface level of the controller. We could have graphical objects such as agents, wheels, boxes, pickers, etc. that represent the events happening, and these objects are not necessarily taken from the world of music. In fact we would rather not use such visual metaphors, as they contain too much history and aesthetic implications. For us, the beauty of OSC is precisely the fact that it is an open standard, and does not incorporate a certain musical tradition like MIDI does.[4]

3.1 Case Studies

3.1.1 SpinDrum – SpinOSC

SpinDrum is a good example of the paradigm shift (if we are dramatic enough to call it so) we underwent in *ixi software* from instruments to controllers.

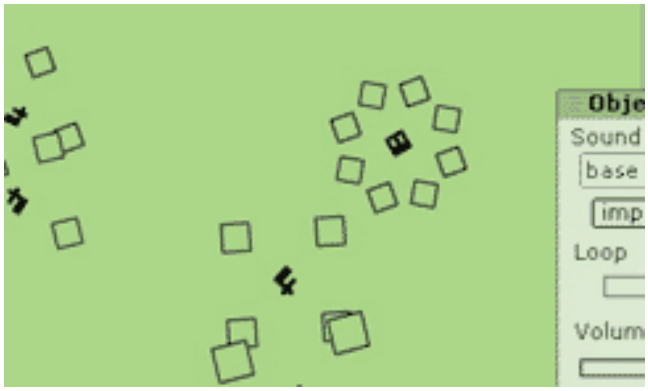

Figure 1. Screenshot of SpinDrum.

The initial prototype of SpinDrum was built as a standalone

[3] As it is not the objective of Pure Data or Supercollider to do drawing-based graphical programming (although there are interesting video and OpenGL libraries for Pure Data, they are not very practical when building complex interfaces) we have had to come up with different techniques to make this technology possible.

[4] Arguably MIDI is not very practical for certain musical cultures whose music is more microtonal or polyrhythmic.

application where the user would import samples into the software and then work with them according to the logic of the graphical interface. The instrument consists of rotating wheels, with variable number of pedals, which trigger an assigned sound file whenever they reach the top position (12 o'clock). The speed of their rotation is controllable. Therefore, manipulating and changing the rotation properties of the wheels can result in exciting polyrhythmic structures. The wheels' size represent the amplitude of the sounds and the vertical and horizontal location represent the pitch and panning respectively. The user can easily add, move or delete wheels making the instrument quite spontaneous and easy to use in a live performance.

This was all well and good, but the strength and openness of OSC could easily be applied to the SpinDrum application and therefore we rewrote it to become an OSC controller. The SpinOSC has more or less the same interactive functionality as the SpinDrum, but now the output is not sound but OSC control messages that are constantly sent from the application to the receiving sound engine which can be built in any language or environment that supports OSC. We have had reports from people who have used it successfully with Max/MSP, Reaktor, Pure Data and SuperCollider.

3.1.2 Picker

Picker was the first OSC application we made. It was based on the notion that for creating generative music, regular or "real world" numbers are more interesting than purely random numbers for generating sounds and controlling musical patterns. Not only should those "real world" numbers be understandable and intuitive, but also easily controlled. The idea is that having an interface with a picker that picks up the RGB colors and its own location and sends this information to a receiving OSC sound engine is a useful tool. As the color background and the picker can be animated, interesting patterns can be picked up and used in a musical composition or performance.

Figure 2. Screenshot of Picker.

In Picker, the user can import video files and/or connect a web camera and blend those video streams together in various ways. Bitmaps can also be used as filters, layering up to four layers of bitmaps on top of each other with adjustable blend. The users can design their own graphical patterns – as animation or bitmaps – which are used to control the sounds. There are four pickers in the application and they can be taught how to follow a route or just move around the screen randomly

There isn't any one way of using this application. We have seen people use it to control visuals rather than sound, others have created drone music, scale transitions, or perform filter and effect manipulation. It is just an open controller that can be used in any situation where such number streams and the easy control of them can be useful.

4. TECHNICAL SETUP

As mentioned above, we chose to use OSC because of its generality, its support in almost every programming language and its speed.[5] OSC is much faster than MIDI and using the UDP protocol, transmitting information between computers on a network becomes an easy task. Another important factor is that one can send symbols, strings, integers, floating points and binary files which makes it more powerful and clearer than MIDI which only deals with integer numbers.

All mapping is much easier with OSC than with MIDI. A typical OSC message has address structure like a Unix file-system or the URL file system people are used to on the Web. As the OSC is built around a client-server architecture, the client will send an OSC message with an address which correlates to the address space on the server. [4] In our case, the ixi application is the client and a patch written in SuperCollider or Pure Data is the server.[6] [1] As the ixi controller has a certain logic to it, which is predefined, the OSC information it outputs is hardcoded, i.e. the user cannot change the names or variable scope of what he or she is receiving. But that is not a problem as this information can be translated and mapped appropriately on the server side.

We can take Picker as an example. The OSC information that is being sent out from picker looks like this: [/picker1, red, green, blue, x, y] where "red", "green", etc. are variables of color and location picked up by the picker. This message is then bundled with the other picker messages and sent through OSC to the server, which decodes the bundle and routes the picker information to

[5] MIDI can send around 4000 bytes per second whereas OSC, using a 10Mb/s Ethernet network can send around 1.250.000 bytes per second.

[6] It becomes a bit more complicated with Supercollider as the SC Synth is a server on its own and the SClang is a client of the SC Server. The language and the server communicate with OSC. We usually need to talk to the SClang, so it serves both as an OSC server for the ixi apps and as a client to the SC Server.

their desired mapped functionality. Thus one picker can control pitch where the next controls filtering or all control their own synthesis elements. There are infinite ways of building a sound engine that receives the picker information. The Picker is a limited controller, it cannot be changed and it is not modular, but the way it can be used has no limits.[7]

Currently we are mainly using Python and Java to write the ixi software,[8] the main reason being that we want to work in a language where the act of programming is more like sketching or prototyping, i.e. a quick way of mocking up an example of a working idea which can be explored and tested out with a sound engine. It took us long time to come to this solution, but it made sense for us not to code for any specific programming environment (like SuperCollider, Pure Data or Max/MSP), but rather keep the applications open for use in all platforms and for all environments. Using Python and Java, it also becomes much easier to port our applications for the Mac OS X, Linux and Windows platforms.

5. CONCLUSION

The codes of the western musical tradition underwent many transformations in the 20th century and many would argue that the remains are only ruins. That is a positive situation in our view, as from chaos structure arises. We find that the commercial software of today does not reflect this situation well and argue that the open source programming environments and the open protocols of OSC serve well as an important addition to the general environments that contemporary musicians and composers need. These environments have been taken into use by a wide range of people from academics to DJs who use them in their own way. Each culture brings something new to the environment technically (externals, classes, code libraries) and aesthetically (as new uses or ideas can spread with the tools used to perform them) thus cross-influencing the work of each other. As post-modernist theory illustrates, the old distinction between "high" and "low" culture has been ever disappearing. This becomes even clearer in practices where the distinct cultures are using the same tools for their cultural productions, and that is a recent change with few historical cultural examples.

Software such as SuperCollider and Pure Data are fantastic compositional tools and workshops for the creation of musical instruments. What we find lacking when working with those environments is the possibility to represent musical patterns visually and create intuitive spontaneous instruments in forms of graphical user interfaces that allow for quick reactions in a live situation. We decided to use OSC in our instruments as we were interested in our software being able to function on most coding platforms and for the generality of OSC itself. Such instruments can supplement physical instruments and interfaces, and they can also contain functionality, which the physical instruments or controllers don't have, such as: memory, automatic repetition, hypercontrol[9], machine learning, adaptive qualities, artificial life, genetic algorithms and artificial intelligence. There is a vast space open where the screen based instrument with its non-tactile but visual feedback can be explored for the benefit of musical performance and composition.

6. ACKNOWLEDGEMENTS

ixi software is a collaboration between Enrike Hurtado Mendieta, Thor Magnusson, David Bausola and many musicians and beta testers who give us valuable feedback and ideas for further development. Many thanks to Chris Thornton for inspiring talks and I would also like to thank The Media Centre in Huddersfield, UK and Buchsenhausen in Innsbruck, Austria for a nice residency programmes which we have had the pleasure of experiencing.

7. REFERENCES

[1] Thompson, John, “Network Communication with SuperCollider 3 Synthesis Servers via OSC: Musical Implications of High-bandwidth Networks”. February 2005, weblocation: http://www.uweb.ucsb.edu/~johnt/Java_SC3.html

[2] McCartney, James. “Rethinking the Computer Music Language: SuperCollider” in *Computer Music Journal,* 26:4, pp. 61-68, Winter 2002. MIT Press 2002.

[3] Puckette, Miller. “Using PD as Score Language” in *Proceedings ICMC 2002*. Pp. 184-187.

[4] Wright, Matt. “OpenSound Control: State of the Art 2003.” Published in the NIME 2003 Proceedings. http://www.cnmat.berkeley.edu/Research/NIME2003/NIME03_Wright.pdf

[7] The idea of modularity has been with us from the beginning, i.e. to create software where the user sets up the functionality of the application and the interactivity, but that is a project for later time as we’d rather spend our time exploring basic interactive structures first.

[8] We use the OSC library by Daniel Holth for Python (http://wiretap.stetson.edu) and the Java implementation by Chandrasekhar Ramakrishnan (http://www.mat.ucsb.edu/~c.ramakr/illposed/javaosc.html). We also use Processing in workshop situations.

[9] By “hypercontrol” I mean structured, branched mapping from a higher level control to a web of lower level control-elements.