Free Software and Computer Music: Beyond Linux

Introduction

The Linux revolution is a remarkable achievement of the Free Software movement. In about ten years, it swept across the UNIX scene and populated machines everywhere in the world. From mobile phones and PDAs to big iron and clusters, we see the Linux kernel powering computers. In its wake, we see the spread of knowledge and information, enabled by the concept of Free Software. As we move from one Linux system to another, not only life is made easier by the availability of the same (or very similar) tools, but, more importantly, these are systems we can learn more about because nothing about them is hidden. In other words, there is no holding us back because of proprietary issues.

Linux, however, is but a component of the great jigsaw that makes up the big Free Software picture. Firstly, as Richard Stallman reminds us, what we call Linux is actually the GNU/Linux operating system. Funnily enough, despite being the most visible part of the system, GNU software is often not given its due. A remarkable achievement on its own, as a collection of software that has on the whole replaced what we know as UNIX, the GNU project is present almost everywhere you look. And far beyond Linux, too: my first encounter with GNU was using a version of Emacs (called Micro-Emacs) in an Atari ST machine.

In addition to GNU/Linux, a number of other Free OSs exist, such as BSD. From the GNU project, there is the exciting prospect of the Hurd kernel, which, when it finally arrives, will probably send a chill down most developers' and hackers' spines. These all give anyone looking to use Free Software good choice, at perhaps the price of having to learn something about it, before actually using it. Beyond these, there is a wealth of Free (and Open Source) Software for other non-free OSs. A walk through Sourceforge just demonstrates how much the concept has developed. Overall, using Free Software is starting to become part of the lives of many people, and hopefully, as we will see in the course of this talk, it will take over the world.

Music Systems: Free Software on non-Free Systems

Music systems developers have, historically, pledged allegiance to the idea of Open Source, if not Free, software. Computer Music and Free Software are forever joined at the hip. The development of the MUSIC family of music programming language, perhaps the most important genealogy of computer music programs, has effectively been Free Software, probably even before the concept was ever originated. It is worth

our while tracing the development of this software to demonstrate how it actually benefited from the concept of Free Software.

The MUSIC *N* family of software originated at the Bell Telephone Laboratories, from the work of Max Mathews, who wrote in MUSIC I in 1957 and MUSIC II in 1958. These programs had been written for the IBM 704, a relatively primitive machine, and were themselves very basic, producing direct-synthesis triangle wave signals. They nevertheless, were the first programs to generate digital audio as we know it today. Samples were dumped to a magnetic tape that could be put through a primitive digital-to-analogue converter and the sound was recorded onto analogue tape for further performances. An earlier attempt actually happened in Australia with the CSIRAC computer, which had a digitally-controlled 'hooter', which was programmed to play melodies. This was perhaps the first digitally-controlled sound device, but not digital audio as we know it.

The real breakthrough came with MUSIC III, in 1960, for the IBM 7094. This programme introduced many of the programming concepts we see in today's computer music languages: the unit generator, a synthesis 'building' block and the table-lookup oscillator, the workhorse of digital synthesis. MUSIC III paved the way for the development of computer music and was a formidable achievement. A number of composers/researchers were attracted by Mathews' ideas and joined in the work: Tenney, Howe, Winham and Randall, among others.

In 1962, MUSIC IV was created, providing the basic model for a whole series of derivatives. Written mostly in assembler for the 7094, this was then passed on from Bell Labs to Princeton University, where a new version, IV B, was created for the same computer model. Other centres became interested in the software, so FORTRAN-coded versions appeared, MUSIC IVF and IVBF. It is unclear as to what degree all of these programs were free software, but at least their source code seemed to be open, as they were distributed among a number of research centres and spawned a whole series of variations and a 'family tree' of software.

Direct 'descendants' included:

- MUSIC 6 and MUSIC 10, from Stanford University
- MUSIC 5, from Bell Labs (from which customised versions were developed and used in other places around the world)
- MUSIC 360, MUSIC 11 and CSOUND, from the MIT
- MUSIC 7 from Queens College
- CMUSIC from San Diego

From this family tree, we have also indirect descendants in the form of widely used systems such as RTCmix, CLM, etc., and adopted children, such as Pure Data and Supercollider. None of this development would have happened if proprietary constraints had been put in this work, from the beginning. Generations of computer music researchers and composers have learned from the availability of this software in binary and source code forms, developing their skills using and improving them. It is interesting that, although intrinsically Free, most of these software systems were build on top of proprietary operating systems, until the availability of Free OSs (effectively with Linux and GNU software).

In the early nineties, many of these big computer music packages were available as source code for ftp download, sometimes covered by confusing licenses (such as Csound and the MIT License, more about this later), sometimes without any license whatsoever. As a postgraduate student in the University of Nottingham in 1993, I had heard of these and was willing to spend some of my spare time doing something with them. I managed to get, from cecilia.media.mit.edu, the source code for Csound, which after some makefile fiddling, I built on a MIPS machine running AT&T Unix. The trouble was, the machine did not have any means of playing audio (in fact I did not even know where it physically was located). My use of Csound was restricted to reading the printed out messages and checking for clipping errors, etc., then running the same Csound code on a (much, much slower) Atari ST machine. Then we got some SUN workstations with audio, so I re-built Csound on them and I was able to use it there. All of this not knowing that, down in Bath, a certain Prof. ffitch was building Csound on absolutely any platform he could get his hands on. With my success with building Csound, I tried also building CMix and Common Music, with mixed results. A lot was learned in the process and all as a result of these packages being available as Free software (even if licenses were unclear).

The Composer's Desktop Project

An interesting initiative that perhaps did not qualify as Free or Open Source (although these terms were not in current use then and perhaps would never describe it very well) was seen in the early days of the Composer's Desktop Project. Developed in a number of Universities in the UK, CDP tried to put together an affordable system for Computer Music. In the early 90s, most of the source code was available for its users to adapt, modify, experiment with. The whole principle of a co-operative of composers developing for an affordable platform was very reminiscent of what goes on in much larger scale within the Linux community now.

The 'affordable' platform around 1990 was the Atari ST. One of its attractions was that it had a built-in MIDI interface. The CDP group was, however, thinking much beyond MIDI and, in fact, that became of little importance in the system. The

problem was how to get audio in and out of the machine. One of the more electronics-minded members, David Malham, developed the so-called SoundStreamer, which was an in-between bit of hardware that connected the Atari games port to a Sony PCM encoder/decoder. This was actually designed for use with BetaMax machines for digital audio recording (one of the first such commercial systems). So in terms of hardware, the system was capable of playing and recording audio, as well as writing and reading to BetaMax tapes.

With a grant from the Gulbenkian Foundation and support from a some UK universities, software was written and ported to the system. This included Cmusic, Csound, the 'Groucho' manipulation package, a graphical additive synthesis programme, a graphical desktop, a command-line interpreter and developer's tools such as Emacs and make. Another important bit of the system was the sound-filing system, complete with an API, which enabled large files to be written to disk, something that the Atari OS did not provide. In later years, the system moved out to a PC-Windows platform and became more proprietary, losing its initial co-operative status.

I have myself benefited enormously from using the CDP platform. It was quite irritating in the beginning to spend three days for a particular processing run to finish and see some mistakes in it, but it was a great educational experience (in fact, as far as Csound was concerned, I had the Unix -testing work-around mentioned before, so things were too bad). Since Music Departments generally could not afford expensive Unix machines with audio hardware, the Atari CDP box was a way into computer music for people like me. Having bits of development software, source code and time to look into these things was a good didactic combination.

Csound

This account of the earlier days of CDP perhaps demonstrate how Free Software not only benefits the community of users, but also the software itself. The larger its user base and developer community, the more it will thrive in the 'software pool'. Another good illustration of this principle is Csound: although for many years based on 70s code design concepts and using an assembly-like orchestra language, it grew to become one of the most used music programming languages. This was largely due to its availability, perhaps as much as to do with the existence of documentation, performance and extendibility. Csound was one of the first music software to become fully available for ftp download when it was released in 1986. As discussed above, many individuals around the world, myself included, picked it up to mess around with it.

Thanks largely to John ffitch's work on organising the source code into the 'canonical Csound', the software became the largest system around (in terms of its unit generator collection). This was the effort of a large development community, which is evident when we look at the credit list, which has over thirty names on it. As with this type of effort, people have come and gone, and the development has occurred in bursts and splits. A number of 'flavours' of Csound were spawned, somewhat repeating the MUSIC *N* family tree. Some of these survived, some died. But the development nevertheless went forward and things got changed and improved as needed.

[R Dobson in 1999 compared the fates of Csound and cmusic: "I suppose one
significant difference between cmusic and Csound, which
might explain the relative lack of 'modern' opcodes in the former, is
that while cmusic has largely remained the property and product of F.R.
Moore, Csound has reaped the benefit of a large, skilful, enthusiastic
and mostly unrestrained net-wide user and development group. This is
partly thanks to the prodigious work of John ffitch just about keeping
everyone and everything co-ordinated, but partly also due to the fact
that it is still essentially standard C, such that anybody can 'have a
go' (including myself), on just about any platform (including the Atari
ST) without have to climb a fierce learning curve of object-orientation,
STL, multi-threading, yacc, lex, et al."]

The licensing issue was a question that took a while to be solved. Csound was originally released under the MIT License, which in the view of many people in the community was problematic and confusing. Csound was then moved to the GPL and immediately after, moved again to the LGPL, which was seen as less restrictive. One of the major issues was the fact that many of us did not really understand the fuss about licenses. For some, like me, this was due to ignorance, and for others, like John ffitch, it was because they came from an era where the concept of Free Software did not exist. As he said to me, "in those days the question was not there, everyone shared their software. In fact, when we managed to get a software to work we were so happy that we wanted to give it to everybody."

With Csound, it was not only the case that the source code for the sound compiler was available, but also that Csound code itself was freely distributed. Catalogues of csound instruments were around for people to use, learn and modify. From Richard Boulanger's toots to the famous Amsterdam Catalogue, a wealth of code was being made available for the community. All these things were incredibly useful for people learning computer music, composition, signal processing, etc.. For this community, such resources are often more important than C source code. That is also the case elsewhere: all the PD patches, SuperCollider code, etc, are fantastic resources available to any computer musician who wants to learn.

The OLPC project

This in fact, brings us to a recent project that is agitating some parts of the Free Software community, the One Laptop per Child project, also known as OLPC. OLPC is a non-profit organisation set up by MIT Media Lab staff to provide a 100-dollar laptop as an educational resource for children in, mostly, developing countries (although not restricted to them), which include China, India, Brazil, Egypt, Nigeria, Thailand and Argentina. The specification for the laptop is almost finalised and the software that will run in it will be all Free Software. The kernel will be the latest Linux and Red Hat is one the project sponsors, so the distribution will be Fedora-like.

It is not clear as to what will exactly be in the final system, but mostly likely a Gnome desktop, probably not a fully-fledged one, will be there. The graphics toolkit then will most likely be GTK and the scripting language, python. In terms of hardware, the laptop will have a 500MHz AMD processor, 7-inch screen, 0.5 Gbyte of flash memory, 128mb of RAM, wireless ethernet and, of course, audio IO, based on the AC97 design. The audio CODEC will be supplied by Analog Devices, the AD 1888.

We got involved in this because Barry Vercoe has joined in to work on the music and audio part of the project and invited a group of us to look into it. The idea is that Csound 5 will be powering it, as a sound server, doing all the necessary tasks for synthesis, processing and 'plumbing' of audio. We would then be using scripted user interfaces for any music/audio software that would go into the system. This is where Python comes in, and pygtk for the graphics. Given the limited storage and memory space, it is likely that we will have a cut-down version of Csound 5, not the 1000-plus opcodes, but something that will have the basic things we need.

The idea for the software in the OLPC project is that it will be something that would enable constructionist learning (or 'learning by making things'). So we want to provide tools for children to learn not only about music but also about the sound universe around and beyond them. We want the laptop to become an instrument that can be played, used for composing, used for messing with sound on its own or in a group setting, as an 'orchestra' of laptops, interconnected by wireless networking. Since the idea of the project is to 'blanket cover' areas with laptops, so that every child has one, classes can become music groups, where the musical performance experience is shared.

The interfaces to the system will be crucial part. The important thing is to keep everything intuitive, but also open the door for further exploration. So it is likely that most users will not go to the level of a csound orchestra, but some might. For these,

some doors will be left open. In designing the graphic interfaces for different tools, it will be important to keep in mind that the target public will be of tremendous cultural and ethnic diversity. So it is important to provide culturally-neutral interfaces, something that might prove to be quite hard.

The networking part will also be an important component. The csound server will need to be transparent, allowing for interconnection of machines at the control and audio levels. While the control side is mostly already present (I will give some examples in my other 'Scripting Csound' talk), sending audio signals will need some extra code and fine tuning to the laptop hardware. This is actually a very exciting idea, which might move upwards from the OLPC project into other high-end Linux music performance systems.

The OLPC is a very exciting initiative that places GNU, Linux and Free Software at the forefront of the spread of technology. This has in fact created interesting backlash reactions from the non-Free software world. Bill Gates described the 100-dollar laptop as a gadget, but later seemed to announce similar plans of his own. The Intel Company has announced, in Brazil, their model for a 400-dollar laptop of similar specification to OLPC. It might appear strange that Intel has chose to announce it in Brazil, but not when we learn that that country is one of most interested partners in the OLPC project. If the project comes to full fruition, we might have half a billion (or more) machines in the world running Linux, something that the non-Free software industry might find difficult to swallow.

Difficulties and Learning Curve issues

The point I made about users needing an intuitive interface to play with is a perennial issue of music software design. With Free Software, it is often said that developers do not do a good job in facilitating the use of their programs. While this might be true in many cases, it raises the question of whether computer music software should actually be easy to use. At this point, it would be interesting to insert a much quoted anecdote by Richard Moore (the creator of Cmusic). It goes like this:

> *Computer music as a field has been likened to a building with a sign*
> *on it saying "Best Eats in Town".*
> *Many people go into this building expecting to find an elegant*
> *restaurant with a parchment menu, formidable wine list, and*
> *pleasant, efficient, even charming service. What they find instead, to*
> *their surprise, is a shiny, enormous, extremely modern kitchen, with*
> *abundant supplies of every kind of foodstuff in voluminous,*
> *refrigerated storage.*

Indeed, the "Best Eats in Town" are available here, but only to those willing to learn to cook!

The trend in computer music today, in all fields, from composition to performance, seems to be to provide easy-to-use interfaces, be it in hardware (controllers, etc) or in software. When this is placed against the backgrounds of Western musical tradition and Science/Technology, where learning something, such as an instrument or a discipline, is a slow and arduous task, we seem to find a contradicting situation here. A violin did not seem to undergo its development in order to make it easier to play, in fact, most likely the opposite happened. Yet somehow when faced with a computer music software we want immediate results, even without knowing anything about what we are doing. We do not seem to care about learning a craft.

We want controllers that do not require us to learn and practice them and instruments that can be played without effort. We required this and we have got most of it: easy-to-use graphic sequencers and synthesisers, with preset beat-patterns and crystal-clear sound. We have it to the point on which it is sometimes impossible to make anything that does not sound good. But of course, what we have got is an amorphous mass of sameness. When we finally realise that microwave dinners are not what we want, cooking is what is required, and a good, well-stocked, kitchen.

Free Software is the kitchen. Some of it comes with great recipes too, which can be learned. And it is way ahead of proprietary software in this area, mainly because it does not really need to provide pre-cooked wares to be able to stand out in an over-crowded market place. So most of the effort is placed at right task for this, which is furnishing the kitchen. This simple realisation is enough to persuade anyone who is serious about their craft, computer music, to look at what the Free Software community has to offer. The learning curve might be steeper, but it pays off in the long run. The journey is as important as reaching the destination.

Conclusion

As we saw, Free software is an open door, regardless of it being or not being on top of a free OS, or even cohabiting with other non-Free programs. In fact, it opens other doors for further experiences with Free software. Many users and developers (myself included) moved on to experiment with and use Free OSs as an extension of other encounters and experiences with Free Software in proprietary environments. We hope that future generations will have a closer and more immediate contact with Free Software, and perhaps the OLPC project might have a part to play on this.

Developing for Linux is important. But beyond it, there is a whole world of software, so it would be worthwhile extending this to say: developing Free Software for as many platforms as you can is even more important. More Free software means more open doors available for anyone wanting to get out, leading eventually to a bigger world beyond Free Software: Free Music, Free Art, Free Learning, Free Ideas...