Cooperating Computer Music Languages

Brad Garton, Columbia University Computer Music Center (garton@columbia.edu)

Abstract: New interconnect protocols and technologies allow computer musicians to draw upon a range of different computer-music programming environments. This paper/demonstration will show a new working methodology for integrating several contemporary music languages (RTcmix, ChucK, Csound, Max/MSP).

Let me begin this paper by telling what it isn't. My proposal for the 2007 Spark Festival of Electronic Music and Art was intended as more of a demonstration of a recent working methodology for 'doing' computer music that I have found congenial ("*Hey! Look everybody! This is fun!*") than a descriptive paper. A fixed text can't quite communicate a demo directly, so I've had to think about what I can write. I should also confess that this isn't a technical paper; it doesn't describe new software or hardware, it doesn't present a bold new signal-processing algorithm, it doesn't discuss new transformational techniques as imbedded in a language design, none of that stuff.

Instead, I plan to discuss a set of computer music software applications that are already widely-used, and describe some off-the-shelf techniques for allowing these tools to communicate with each other. For many contemporary computer musicians, this won't be earth-shattering news. Nevertheless, I would also like to use this paper as an excuse to exhort users and developers of music tools to keep in mind the expanded range of possibilities that relatively transparent and direct communication of audio/control data can provide.

The Olden Days

I recall an advertisement from the late 1980's featuring a vision of the ultimate music-technology utopia that depicted a MIDI cable plugged directly into a hapless computer-musician's brain, the implication being that ideas could flow "transparently" from conception to realization. Contrary to this fantasy, however, was the realization by many of us working in music software development that the design of our tools had a profound effect on the creative process. The conduit between ideation and instantiation was anything but "transparent".

Just one example to show what I mean: Suppose I am creating a new software synthesis application, and I decide to have the user specify the pitch parameter as a direct frequency (Hz). I can almost guarantee that the music produced with this application will have noticeable features and an aesthetic very different from music produced using a western equal-tempered pitch specification system. Of course, given the universal Turing machine nature of present-day computers, it should be possible to emulate different tools, holding up the possibility of transparency through correct design decisions. In the simple pitch-specification example the designer could have allowed for multiple pitch formats, or the user could have found some kind of appropriate conversion utility, but the point is that the structure of the tool used can exert a powerful influence on the creative imagination. It is impossible to capture all potential approaches through judicious design decisions.

In the earlier days of digital music, this effect was quite evident. Because of the difficulties associated with the then-young field of computer music, composers using music technology would often adopt only one or two applications for producing pieces. Redesigning software for an idiosyncratic compositional approach required effort far in excess of the expected results. Few people wanted to do this.

At past festivals and conferences devoted to the presentation of digitally-created music, it was almost trivial to hear which application was used for a specific work. Pieces often were centered around a single signal-processing or synthesis technique: the "LPC" works of Paul Lansky, the "FM" pieces of John Chowning, the "granular" music of Barry Truax. Interconnection between different synthesis languages was possible, but the lack of common basic standards made this operation cumbersome at best. Even soundfile formats were often incompatible between different computer music environments.

Happy Days

Technology has advanced, and the state of the computer music field has changed. Today there exists a much greater range of different approaches for producing digital music, and recent developments have made it much easier to interconnect these diverse applications. Music software packages can now be linked in two main ways:

– *connection through audio*. Soundfile formats have become reasonably stable (wav, aiff, aifc, etc.) and software libraries for reading and writing various formats are readily available. A soundfile created by one application can be processed through another without much effort. More intriguing, though, is the direct connection of digital audio between software packages using interapplication communications protocols such as JACK, Soundflower, ReWire or the direct imbedding of one application within another using a plugin architecture such as VST, AU, or LADSPA.

– *connection through data*. To a certain extent, this capability has existed generally since the advent of the MIDI protocol in the mid-1980's. MIDI has been abundantly criticized for shortcomings in its design. New music network protocols such as OSC and (again) the direct imbedding of applications now allow a much richer range of data to be exchanged.

I now want to describe a scenario to demonstrate how this interconnectedness can work. Using the "scrub" capability of Michael Klingbeil's wonderful SPEAR audio analysis/resynthesis program, I can generate a variety of interesting sounds. I would like to process these sounds through a filter instrument I built using Perry Cook and Ge Wang's ChucK language, but I want to control the pitches of my filter instrument using a notated score. To complicate matters further, I like the idea of triggering a random sequence of notes every 7.8 seconds using a script in RTcmix, a music language I helped to write. Finally, I plan to process everything through a few commercial reverberation/echo plugins, and track the output several times using Digital Performer to create a final piece.

How can I accomplish this? Obviously one way would be to do the whole operation in stages – write the soundfile with SPEAR, process it through ChucK (after generating appropriate control data from a musical notation program), combine everything at a later point... but this would diminish my ability to do something critically important to the result: my "scrubbing" gestures need to be tightly-coupled to the sound at the end of the chain. In other words, I want to hear what I'm doing.

Modern technology to the rescue! Through various audio-distributing schemes and software imbeddings, I am able to get all elements of the chain working together. Using Soundflower, I can route the audio output of SPEAR to the **[chuck~]** object which imbeds the ChucK language inside Max/MSP. I can control my ChucK instrument using the Max/MSP java interpreter (**[mxj]**) running Nick Didkovsky's music notation package JMSL. My RTcmix scripts can be triggered using the Max/MSP imbedded **[rtcmix~]** object, with the audio output from RTcmix and ChucK sent from Max/MSP to Digital Performer via ReWire. Digital Perfomer has a number of terrific reverberation and echo plugins, and can easily handle the mixing and tracking of the resulting sounds.

The Future

I could have complicated the above scenario further by adding a connection to some SuperCollider action via the CNMAT **[OpenSoundControl]**-family of Max/MSP objects, or incorporating an additional java component by using Phil Burke's JSyn synthesis system. The complications from these arise through the necessary re-routing of the audio outputs if I want to produce a single, coherent result. OSC would also require additional data coding and network set-up. Even though these complications are fairly minimal, the contrast with a fully-imbedded language (like the **[csound~]** object or an additional VST/AU plugin) leads me to encourage developers to make interconnectability as simple and transparent as possible. I also have to admit a bias towards imbedding in that it allows a very high degree of 'cooperation' between different environments without much need for data translation or reformatting.

Ultimately I would like to see music software that is almost completely interchangeable; one environment acting as the host or being imbedded inside another with ease. Indeed, this situation has a corollary in contemporary web-usage. Users are almost completely unaware when they visit different web pages if they are running javascript, java, a flash animation – how many have even heard of FTP?

I would love to see music software also offer this degree of interoperability. I want to urge computer musicians to explore the capabilities available through multiple music environments. Although it can't transcend the creative bias that attends any musical tool, it can at least present a diverse set of options for consideration.

References

Here is a list of some of the software and applications discussed:

ChucK – http://chuck.cs.princeton.edu/
[chuck~] – http://music.columbia.edu/~brad/chuck~/
[csound~] – http://www.davixology.com/csound~.html/
Digital Performer – http://www.motu.com/products/software/dp/
JACK – http://jackaudio.org/
JMSL – http://www.algomusic.com/jmsl/
JSyn – http://www.softsynth.com/jsyn/
Max/MSP – http://cycling74.com/products/maxmsp/
Open Sound Control (OSC) – http://www.cnmat.berkeley.edu/OpenSoundControl/
ReWire – http://www.propellerheads.se/technologies/rewire/
RTcmix/**[rtcmix~]** – http://rtcmix.org/
Soundflower – http://cycling74.com/downloads/soundflower/
SPEAR – http://klingbeil.com/spear/
SuperCollider – http://www.audiosynth.com/