

"Know-Nothing Network Music"

Chris Brown

"Now a federated, decentralized system of free association ... is the appropriate form of social organization for an advanced technological society, in which human beings do *not* have to be forced into the position of tools, of cogs in the machine. There is no longer any social necessity for human beings to be treated as mechanical elements in the productive process; that can be overcome and we must overcome it by a society of freedom and free association, in which the creative urge that I consider intrinsic to human nature will in fact be able to realize itself in whatever way it will."

--- Noam Chomsky, *The Chomsky-Foucault Debate On Human Nature*,
Chomsky and Foucault, The New Press, New York, 2006, pp 38-39.

This conference is, to my knowledge, the first national or international meeting specifically focused on the practice of computer network music. When I first started making music using computer networks over twenty years ago, my collaborators and I within what became "The Hub" assumed that this technique of sharing data within a band of computer musicians was so obvious an idea for exploring the potential of this new medium that it would quickly become widespread. In fact, it's only recently become common enough that a couple of dozen practitioners from Europe and America can come together to discuss the differences and similarities of their practices.

In this paper I briefly describe the kinds of ensemble interactivity I have used in playing network music over the years. The most significant aspect of this work for me is the new kind of ensemble relationship it achieves between its players. Ensemble relationships in music represent sociological patterns within the culture they arise from. Thus computer network music may be seen as an experimental practice that seeks, among other things, to create new forms of social interaction that include the embedded mechanical behaviors of computers used as musical instruments. The problem that arises within this practice that still needs to be resolved is to invent ensembles that embody the kind of "decentralized system of free association" that Chomsky envisioned for society. As our music, along with our society, becomes more mechanized, we need to develop a musical aesthetic that avoids forcing *musicians* to act as cogs in the machine, but will rather enable them to collaborate creatively and imaginatively in new ways, coordinating their thoughts and behaviors in real-time, which is the social purpose intrinsic to the practice of playing music together.

Emergent Phenomena in “The Hub”

The aesthetics and practice of The Hub in the 1980s and 90s centered on the idea of creating emergent phenomena from the interaction of independent, algorithmic musical instruments. Our early repertoire was based on specifications for pieces that described very simply in what distributions, with what types, how often, and in what formats data was to be shared among the players. The application of how that data was used in influencing the music made by each player was largely left up to the players themselves. We considered these specifications to be generative of the character of each piece, so that the author of the specification was credited as the composer of the piece. Thus, we prioritized network behavior over the actual sounds employed in the realization of each piece, although composers could also provide general descriptions of the type of sounds that should be used, and could veto performer choices as being incompatible with the aesthetic idea. In practice however we never made really new versions of each piece with different sounds or instruments, so each piece eventually became associated with a particular set of sounds, which gave it a character not originally specified by the composer.

The social practice of developing and rehearsing the repertoire involved the following process: 1) composer specifies the network behavior and sometimes the general sonic character; 2) each player independently develops software for their own computer music system that implements the specification, using sounds of their own choice; 3) the group meets to debug the network created by the interaction of their software; 4) once debugging is complete, sounds and behaviors of each personal instrument continue to be tweaked, but essentially, the piece is done – rehearsals generally involved only adjusting for changes that system software “upgrades” would make necessary. In fact, the group spent the majority of its time together implementing step three, and these were quite often excruciating sessions in which little or no music, or even sound was ever made! Social tensions and self-doubt could erupt, generally chastened by geeky humor, salty snacks, and beer. But once the bugs were worked out, the piece would suddenly emerge, and would usually prove to be quite robust in its behavior. Concert tours were often prepared with a single rehearsal in which everyone would simply get reacquainted with the loading and individual performance requirements of each piece.

We favored a “tight” approach to the network music ensemble: improvisation had more to do with making slight adjustments to algorithmic software that

determined all of the ongoing events of the music. Pieces that might involve “hands-off” performance techniques – in which the algorithms generated the piece by themselves without human intervention – were generally favored as an ideal. We acted democratically in the process of building a different architecture as proposed by each member of the group, but in performance we mostly interacted like cogs in a collectively designed machine. We did however avoid the uniformity of a machine that was created by a single individual, because each of our instruments were made individually by each performer.

Other Network Ensemble Pieces

The Hub disbanded in 1997, having perhaps allowed this emergent phenomena aspect of its work to have played itself out. It reformed in 2004 to perform at the Dutch Electronic Art Festival in Rotterdam, and since then has begun to concertize regularly again. In between, I made a few network pieces designed for other players.

Invention#5 (1999) was based on software ported from my interactive rhythm installation called *Talking Drum*ⁱⁱ, using *udp* messages containing only control data to trigger sample-based synthesis at each of seven computer stations in Oakland, California and Karlsruhe, Germany, as part of the “net_condition” exhibition at ZKM there. A *berimbau* player in Oakland interacted with preset rhythmic patterns, generating variations that were sent to players at each site, who voiced them intuitively and freely. Their instruments were mirrored at each rendering location, so that changes in voice controls were updated and heard at each location. Individual sound events could also be played with a mouse, which would be sent to the rhythm generator machine in Oakland and affect the interactive rhythm algorithm. Each event message was time-stamped with a delay to smooth latency jitter, and with very minor differences the music at each location was the same. Although technically successful, socially the piece was completely unsatisfactory. Without visual feedback, and with a complete disjunction of the visual and social environments, there was very little sensation that the network connection really mattered to the music. Coordination of all the various software components was also overly complex at each station, so the piece was only performed once.

Cloudstreams/Bellwethers (2001) was written for a local network of six computers running SuperCollider2, for a workshop with players who had varying levels of programming experience; while the framework software was written by me, it was intended to be extended and voiced by each player. I wanted to explore

Invention#5's sound mirroring technique in a local network context, where dynamic network reconfigurations would produce changing spatialization effects. Each player always has an assigned partner rendering a version of their partner's sound, mixed with their own. Rather than sending specifications for individual sound events, clouds of sounds are stochastically rendered from the same specification data, so that two clouds are produced at two different stations in response to gestures initiated using the mouse by any one player. Any player can also reshuffle the assignment of partners, creating a new randomly generated arrangement. A second section of the piece involves each player singing or whistling into the internal mic of her computer, creating a stream of bell sounds sent to her partner after a small echo delay. The dynamic network interactions produced wonderful ensemble relations with tangible acoustic results, but since most of the software was still written by me, it sounded more uniform than I would have liked.

Finally, I made *TeleSon*, for two *reactTables*ⁱⁱⁱ for the ICMC in 2005, in which two remote instruments were tightly networked to form one single virtual instrument. This was by far the most successful distance network performance I have ever done, because the integration of mirrored visual and audio outputs at each location resulted in a highly responsive sense of ensemble. The piece was a quartet, with two musicians playing locally on each *reactTable*, one in Barcelona, Spain and the other in Linz, Austria. All networked data was generated in response to performer's gestures on their own instrument, and the *reactIVision*^{iv} software sensing these gestures from a video signal simply sent its output via Open Sound Control messages to both locations. I wrote the synthesis and algorithmic generation software in *SuperCollider3*, and excepting slight differences at each site due to network latency (around 25 msec), each player's actions were seen and heard identically at each instrument.

The Hub Redux

Since The Hub reformed in 2004, our attitude towards the role of the network in our music has changed. While the interest in creating music that involves data-sharing remains, the aesthetic and performance practice has loosened up considerably. The hiatus brought a reconsideration of what really defines the sound of the group -- what the relative roles of network structure, the individual sonic personality of each group member, and improvisation are in creating the identity of our music. Perhaps the preoccupation with network architecture is only a structural framework on which to build a music that depends equally on the musical abilities, personalities, and imaginations of its members. Certainly of

equal importance with the interest in emergent phenomena and sonified data structures has been the social interest of computer musicians in performing music together as a group, in creating music that sonifies the interpersonal as well as the technological network.

What has happened to ensemble music making in the age of live electronic music? Has the technoculture's emphasis on individualism ignored and even devalued the role of spontaneous, collaborative musical intelligence? Perhaps more than technically, the Hub pioneered a response to this need for social networking for the age of laptop musicians. Recent performances have emphasized the potential of the group to perform in different configurations, and to add guest musicians as well as multimedia artists to the group. In order to do this, more flexible and open specifications are required. Our future seems to me to depend on our ability to extend the practice of data-sharing beyond the boundaries of our own ensemble. Can we design and agree on some simple protocols, easily implemented by any musician/programmer, that will allow us to play our own musics while also sharing data in musically meaningful ways?

"Know-Nothing" Network Music

What do we and our machine instruments need to know about each other in order to play effectively with each other in a network? Do we all have to speak (use) the same language (software?) Do we all have to adhere to an agreed upon protocol? Is that protocol specific to a piece, or can we create one in which many different pieces (performances) are possible?

How much freedom does a network piece allow to each individual performer? Is it possible to design an "open" system, in which each performer is free to send any other player any type of message, without any requirements on how it should be used? Within this approach, each player can publish whatever data (s) he likes to whomever (s)he wants in the network, and the receivers are free to use or ignore that data in whatever way they like.

But there would be no point in using a network if this data were not somehow used to create new types of musical behavior. So this approach assumes a practice in which every player in the network is also a programmer, because pieces in which all the code is written by one player tend towards uniformity. Uniformity is the bane of the computer medium, manifesting itself in the dominance of repetition, replication, and homogeneous textures in its musical results; networks may be used to combat this trend, instead of simply creating

extended means of control. Yet there is nothing inherently fascinating about programmers performing: networking is interesting because the programmer-players make their own idiosyncratic and flexible software instruments, and those instruments are responsive to other instruments in a network.

“Know-Nothing” network music ideally uses rich specifications, loosely applied. It presumes no necessary behavior in response to data made available within the network, but assumes that all performers will have a vested interest in using it. It accepts the model that each computer music instrument is also a listener, capable and interested in knowing what structures are being used by the other players in the ensemble, and designed to take advantage of that knowledge in flexible and varied ways. It models itself on the idea of group improvisation by human musicians, adapting that practice to the special capabilities and limitations of machine musicians.

“Know-Nothing” Protocol

So, this is an invitation. Anyone interested should begin by downloading Ross Bencina’s wonderful OSCgroups^v software, which makes connecting an arbitrary number of computer players in an ensemble simple, requiring only that everyone send messages using the Open Sound Control protocol, and that the IP address of the machine that runs the OscGroupServer application be known to all players. Within this system, all messages are sent to all members of the group, who use their own protocols to filter for relevant information. Login to the server using a Unix command-line after navigating to the OscGroupClient directory entering something like the following:

```
./OscGroupClient music.mills.edu 22242 22243 22244 57120 myName myPwd  
knowNoGrp knowNoPwd
```

...where ‘music.mills.edu’ is replaced by the url or IP address of the machine running OscGroupServer, and the fourth port number is replaced by the port number used to communicate with your software application. The terminal application will regularly display the names of all other players logged into the group.

Next, we should agree on a few very basic musical parameters that we can use to describe to our network partners the music we are playing. Let’s also presume

that the default data format describing these categories is a floating-point number between 0 and 1. Here are some parameters to start with:

density, 0 = silence, 1 = continuous sound, and values in between describe the proportion of sound to silence, or granulation;

amp = silence to maximum;

freq = low to high, logarithmic scale mapping 10 octaves from 20 Hz -> 10.24 kHz;

timbre = 0 is simple, 1 is complex – this measure might combine measurements of harmonicity and spectral density;

periodicity, 0 = random, 1 = evenly pulsed.

OSC messages reporting the status of these parameters in your musical output should be made entirely of strings, formatted with a field beginning with your name, followed by a field for the receiver's name, which defaults to 'all', then includes the parameter and data fields, like this:

```
/myName/all/density/0.5
```

Next, write procedures to automatically update the ensemble with new values that describe your musical output as it changes. You should also design data structures to store, display and, last but not least (!), use the data received from other players in the group to influence your own music. The ability to dynamically choose which data is selected and how it is used in the generation of your music will obviously be critical in making the network interaction meaningful. Clearly, this requires adding network input data inputs to your own instruments and algorithms, as well as analysis functions of your music output. But any amount of implementation is enough to begin. Keeping things simple makes it easier to hear the influences.

While one approach would be to write a very simple new piece of music software that maximizes these network functions, it is preferable to the "Know-Nothing" approach that simple ways of implementing networking are added to already existing, more complex and personal instruments. Rather than worrying about perfecting the network interaction, let's accept fuzzy implementations that gradually gain their focus, so that we don't need to abandon the music we've already developed for ourselves in order to play in a network with each other.

ⁱ <http://crossfade.walkerart.org/>.

ⁱⁱ "Talking Drum: A Local Area Network Music Installation", Leonardo Music Journal, vol. 9, MIT Press, 2000.

ⁱⁱⁱ <http://mtg.upf.edu/reactable/>

^{iv} <http://mtg.upf.edu/reactable/?software>

^v <http://www.audiomulch.com/~rossb/code/oscgroups/>