The InterHarmonium

:

An Investigation into Networked Musical Applications and Brainwaves

Andrew Brouse

Music Technology, Faculty of Music

McGill University, Montréal

August, 2001

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements of the degree of
Master of Arts in Music Technology.

ABSTRACT

This work surveys currently available methods for measuring human brainwaves in order to generate music and technologies for real-time transmission of audio and music over the Internet. The end goal is to produce a performable music system which sends live human brainwaves over the Internet to produce sounding music at another, physically separated location.

RESUMÉ

Ce travail traite des méthodes pour mesurer des ondes de cerveau humain afin de générer de la musique et des technologies pour diffuser la musique en temps réel sur des réseaux électroniques. Le but est de créer un système pour envoyer les ondes de cerveau par Internet vers un autre ordinateur physiquement séparé pour produire de la musique.

It is not a garment I cast off this day,
but a skin that I tear with my own hands.

Nor is it a thought I leave behind me,
but a heart made sweet with hunger and with thirst.[1]

Kahlil Gibran

[1] Gibran. p.4

ACKNOWLEDGMENTS

The author wishes to acknowledge the assistance of the following persons:

Bruce Pennycook

Alexandre Burton

Ian Manns

Robert Zatorre

Jean Gotman

Richard McKenzie

Nina Czegledy

Maxime Rioux

Galerie Oboro

and finally, for incredible patience, my advisor,

Philippe Depalle

table of contents

introduction

This work is about brainwaves, networks and the harmonic series. I am investigating sending brainwaves over the Internet to make music using the harmonic series. I will attempt to expound the historical, technical and artistic bases of my research into these areas.

I will detail some of my investigations into the technologies and historical practices which can and have enabled real-time transmission of musical data over electrical networks. I will also look at some of the history and aesthetics of the use of human brainwaves in the creation of musical works of art.

The end goal of this work was to produce a performable music system which would enable myself or others to capture human brainwaves and transport the data in real-time over the Internet to another location where the brainwave data could be reconstituted and used to synthesize music. My initial plans were much more ambitious than I could practically realize. By documenting the, at times difficult and frustrating, process of developing this system I will hopefully aid others who wish to pursue such a path.

introduction : motivation

In developing hardware and software inventions in Music Technology, my goal is first and foremost creative and compositional. What interests me most about certain new technologies is the possibilities which they open up to composers and artists. It is exactly this which has made me become interested in the principle technologies used in the InterHarmonium. Gordon Mumma's attitude towards music and technology nicely sums up my own approach to the subject:

> My 'end-product' is more than a package of electronic hardware, it is a performance of music . . . some differences exist between the design and human-engineering of electronic music studio equipment and that of live-performance equipment. In the studio the composer doesn't really work in real-time. He works on magnetic tape, without an audience, and can use his studio-time for 'reworking'. In the live performance an audience is waiting to be entertained, astonished, amused, abused, or whatever, and there is no time for 'reworking'. My decisions about electronic procedures, circuitry, and configurations are strongly influenced by the requirements of my profession of a music maker. This is one reason why I consider that my designing and building of circuits is really 'composing'. I am simply employing electronic technology in the achievement of my art. [2]

My musical goal is to be able to have multiple network connected clients able to send and receive real-time musical control and data signals in order to create a wide-area musical performance system. (Certainly, other possible usages of such a system could include remote data acquisition and analysis). This should

[2] Nyman. p.77

not be considered in any way as an attempt to replicate a musical ensemble playing together in the same room - the speed of light and tolerable latencies for ensemble performance essentially guarantee that this will never happen.[3] Pursuing that end is a fruitless goal in my opinion. What I am interested in, instead, is a system whereby all the realities of the network (latency, packet loss etc.) are accepted as part of the system and are in fact a generative part of its operational philosophy.

In the creation of such a system, I do believe that I am taking part in, and contributing to, the tradition of experimental musics as summed up so succinctly by John Cage:

> Those involved with the composition of experimental music find ways and means to remove themselves from the activities of the sounds they make. Some employ chance operations, derived from sources as ancient as the Chinese Book of Changes, or as modern as the tables of random numbers used also by physicists in research. Or, analogous to the Rorschach tests of psychology, the interpretation of imperfections in the paper upon which one is writing may provide a music free from one's memory and imagination. [4]

In using certain technologies to realize my work, I choose to exploit the limitations of those technologies and rather that trying to make them behave in a certain deterministic way, I choose to accept the way that they behave and to include the indeterminacy and incertitude of, for example, the Internet and

---

[3] Crowcroft. p.118
[4] Cage. p.10

brainwaves, as part of my composition. Thus, in using certain Internet protocols such as the "unreliable" User Datagram Protocol (UDP) for data transport as opposed to the "guaranteed delivery" Transmission Control Protocol (TCP), I must accept that some information can and will be lost. To what extent this happens may have only to do with how busy the network is on that day, or whether a certain router is down, or perhaps just a stray electron at a network interface might cause some data to be lost.

What particularly entices me about the Internet is that it is a human-created model: a macrocosm/microcosm of our own consciousness, our society, and indeed, our own brain. It is not any accident that the ideas which have taken form in the current Internet had been proposed many years before its inception by visionaries such as Vannevar Bush[5], Paul Baran[6] and J.C.R. Licklider[7]. I enjoy the idea that there is an autonomous intelligence about the network which we cannot control - this was its design. Rather than trying to impose restrictive controls and demands on the situation in order to realize a final intent, I prefer to accept the realities of the Internet "as-it-is". Any packet losses or network hiccups are the product of very complex interactions between human-designed software and hardware in interaction with the realities of the physical universe. That changes in the moment-to-moment state of a brainwave performer are likewise susceptible to various unpredictable psychological and physiological factors is also attractive to me. This is, in a sense, the incursion of true nature upon the "controlled nature" of human beings. Nature is always more complex, chaotic and unpredictable than the will

[5] http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm
[6] http://www.rand.org/publications/RM/RM3420
[7] http://www.ibiblio.org/pioneers/licklider.html

of human beings would have it. Nature - as Morse Peckham would say - is nonetheless, our nature.[8]

We are surrounded by many kinds of networks. There are neural, electrical, pulmonary, respiratory, alimentary, lymphatic and other networks inside our own bodies. Networks of waterways connect watersheds to streams, rivers to lakes and finally to the seas and oceans. Over 2000 years ago, the ancient Romans constructed elaborate networks of aqueducts to supply water to their homes. Rail, road and air networks connect our cities and towns to each other. There are electrical networks created by human beings which have been in existence for almost 150 years; the electrical power grid and the telephone system are both pervasive. There are human networks of interconnected, interdependent individuals in our societies. More recently, we have come to be familiar with computer networks and particularly that global computer network known as the Internet.

The Internet allows a kind of instantaneous collaboration over great distances which was not previously easily possible. We have seen that this can be very empowering and productive for human endeavour. At the same time, it brings with it another kind of distancing, a removal, a virtualization, of the human experience. The virtualizing qualities of the Internet - shared to a large extent with other means of "mechanical reproduction" [9] as noted by Walter Benjamin (primarily in relation to film), could imply a "loss" or "death" [10] as Roland Barthes refers to in *Camera Lucida* (in relationship to photography).

---

[8] Peckham. pp.308-315
[9] Benjamin. pp. 217-240
[10] Barthes. pp. 92-94

When we listen to a CD or view an 'audiovisual', 'multimedia' presentation in front of a computer monitor or video projector, we are witnessing a very limited reproduction of an original experience or performance. Many obvious factors such as smell, touch, taste are missing as well as many much more subtle nuances of visual, auditory and motor perception. One could make the case that there are even more ineffable modes of human perception which are lost in the process of virtualization which we would have difficulty to express in words.

My objective in designing a system for internet music performance is to take that which is invisible or ineffable and make it visible and palpable. Thus, in a sense, I am building a system which attempts to circumvent the virtualization inherent in internet transmission. My current notion is to extend the work in brainwave control of music by sending those brainwaves directly over the Internet to interact with remote or local music generation systems. Technically, what this involves is to take analogue brainwave signals as input, discretize and analyse these signals according to user-selectable parameters and then interact with other user-agents running a similar system at other arbitrary networked locations. One could certainly extrapolate from brainwaves as a control source to using any other continuously variable analogue phenomenon for that function.

I am particularly interested in the alpha wave frequency band of the human brainwave spectrum. This is largely due to the specific physiological/somatic states in which alpha rhythms are most strongly associated. In order to produce strong alpha waves, the subject must be in a deeply meditative, non-

visual state. No conscious attempt to control the alpha waves in other physiological states will succeed.

Ultimately, I want to make music that I want to listen to. I have long been fascinated by the Harmonic Series and music which is derived from the it. I have created installations, performances and compositions which exploited various characteristics of the Harmonic Series. This stems in part from my interest in the ideas of Hermann Helmholtz, Harry Partch, LaMonte Young, Terry Riley, James Tenney and others who have investigated tuning systems based upon the Harmonic Series.

Additionally, I want to explore an effect known as the "Missing Fundamental". The Missing Fundamental is a psychoacoustic phenomenon whereby the human ear perceives a fundamental frequency - the first harmonic of a tone - when that frequency is not in fact present.[11] This often occurs on inexpensive speakers such as those found in a car radio where the ear recognizes the harmonic overtone structure of a fundamental tone which cannot in fact be reproduced by that speaker. The attendant human psychoacoustic mechanism perceptually 're-creates' that missing fundamental.

I would like to use this notion to create a situation in which the missing fundamental is in fact the alpha wave centre frequency (8-13 Hz). This frequency is not directly perceivable by the human hearing mechanism but, I posit, could be made psychoacoustically apparent by its overtone structure. Due to the very close proximity of partials, this overtone structure can produce

[11] Zatorre. pp. 566-572

very rich and dense tonal clusters which can then be modulated and selectively adjusted to produce complex soundscapes.

To accurately determine the pitch of a very low sound it is necessary to observe that sound for a long period of time. The longer we listen to a tone the more accurately we can perceive or measure its pitch and the more accurately we can harmonize another interval to that tone and thus the accuracy of any tuning system depends on how long the pitches are sustained. In his experiments into just-intoned musical instruments, Hermann Helmholtz chose the Harmonium as the ideal instrument for experimenting with tuning systems for exactly this reason.

> Among musical instruments, the harmonium, on account of its uniformly sustained sound, the piercing character of its quality of tone, and its tolerably distinct combinational tones, is particularly sensitive to inaccuracies of intonation. And as its vibrators also admit of a delicate and durable tuning, it appeared to me peculiarly suitable for experiments on a more perfect system of tones.[12]

This phenomenon was also noted by Tony Conrad, John Cale, Marian Zazeela and LaMonte Young in their early 1960's experiments with long sustained tones (the use of which were initially inspired by sustained notes in the serial music of Webern).[13] They found that as tones were sustained for long periods of time it necessitated a readjustment of the tuning system because it became apparent how out-of-tune the equal-tempered pitches were. Concert singers and

---

[12] Helmholtz. p.316
[13] http://www.allmusic.com/cg/amg.dll?p=amg&sql=Bsxknikn6bb39~C

violinists usually adjust their temperament or sometimes use vibrato to mask this out-of-tuneness especially when playing with a fixed equal-tempered instrument such as a piano. Conversely, this out-of-tuneness is not nearly as noticeable when the notes are not sustained for long periods of time and thus those kinds of music in which equal-temperament is used place less emphasis on sustained tones and tend to have faster moving melodic lines.

In my system, the alpha wave frequency is averaged over a very long time such that the fundamental frequencies will not change rapidly but the instantaneous surface detail will shift and pulse while the underlying harmonic structure will remain and only change after a significant period of time. This has relevance to some traditional Indian and Chinese tuning systems and indeed to the music of LaMonte Young and a large current of contemporary works which use some form of just intonation. In most of these cases a strong emphasis is placed on a constant lower fundamental tone whilst the surface details shift and pulse. This also has concordance with the operation of alpha phenomenon is human beings in that alpha states cannot be easily achieved and usually take several minutes at least to reach after a period of quiet meditation/concentration.

history

In 1999, I arranged a performance of Alvin Lucier's seminal composition "Music for Solo Performer" (1965) at Pollack Hall in McGill University's Faculty of Music. I have been familiar with and interested in the music of Alvin Lucier for quite some time. Lucier's music usually exploits some very simple, basic acoustical, electromagnetic or other natural physical phenomenon in a novel, intuitive and poetic manner. It is the poetry of his work which attracts me. He has always been a composer unafraid to use technology in his compositions but never lets it become the composition.

history : brainwaves in music and art

In "Music for Solo Performer", the first musical work to use brainwaves as a generative source, Lucier specifies the use of the alpha brainwave spectrum to generate sounds by directly linking these subsonic (8-13 Hz)[14] waves to transducers which in turn actuate percussion instruments to generate the sounding music.[15] (see Appendix A)

In order to perform this piece, I needed a device to measure human brainwaves. After some searching, I found a number of commercial devices available for doing this. I found most of these devices to be either too expensive or very limited in functionality. Additionally, most of these devices were geared to the 'hobbyist biofeedback' user rather than being scientific

[14] Fisch. p.149
[15] Lucier. p.69

instruments. In principle, a circuit to measure alpha brain waves is not extremely difficult to build. It is effectively a very high-gain, high-quality differential amplifier. What this does is measure the differences in potential at various places on the human scalp and enormously amplifies them while rejecting as much as possible all the other electromagnetic interference which is about. The difficulty in building such a system is to ensure it has very low noise and high rejection of extraneous signals.

As I was searching for the right device, Bruce Pennycook suggested I should get in touch with Robert Zatorre at the Montreal Neurological Institute (MNI) to see if he could be of any assistance. Dr. Zatorre put me in touch with Dr. Jean Gotman, also of the MNI, who, after some discussion, suggested that a disused analogue EEG machine might be made available for my experiments. The MNI finally agreed to give me an old but still functional Grass model 8 analogue EEG machine which was being phased-out of service at that time. After learning to use this machine I began to experiment with using it to make music. At a key point in my learning and experimentation, Ian Manns, a Ph.D. candidate in Neurology at MNI, helped me with some details of setup and usage of the EEG machine. As for the actual percussion instruments required by the score of "Music for Solo Performer", I turned to a collection of instruments developed by Montreal musician and artist, Maxime Rioux. Maxime had for several years been developing a system for 'automatic' music performance which he calls the "Automates Ki":

> Les automates sont mûs par de hautes ou de basses fréquences inaudibles.
> Ces fréquences font vibrer des membranes de petits haut-parleurs sur

> lesquelles sont fixées des baguettes, ressorts, fils et autres objets contondants qui vont se percuter sur des instruments acoustiques tels; tambours, cymbales, cordes, clochettes, boîtes de conserves, etc.[16]

These 'automates' are various motley percussion instruments which are actuated by inaudible low-frequency pulsations created by analogue synthesizers and various other analogue electronic devices. It was the similarity between Maxime's actual performance system and the requirements of the Lucier score that led me to ask Maxime to participate in my performance of "Music for Solo Performer".

history : networked audio

One could trace the history of networked audio as far back as Thaddeus Cahill's Telharmonium which allowed music to be generated and played over telephone lines at around the turn of the last century (c.1900). In fact Edward Farrar's experimental attempts at sending musical tones over telegraph lines beginning in 1851 marks probably the first attempt to send music over an electrical network.[17]

The history of broadcast of real-time media over Internet Protocol (IP), however, is relatively recent. This phenomenon is only about ten years old. The ideas have been around somewhat longer but the remote possibility of actually implementing working broadcast systems is much more recent.

---

[16] http://homepage.mac.com/automateski/technique.html
[17] Weidenaar. p.1

The capacity for real-time delivery of audio/visual media over internet protocol was first implemented by the university and research communities in the early 1990s. Centres such as Lawrence Berkeley Labs and the University College of London produced reference tools for real-time audio/video/whiteboard conferencing over the nascent Mbone. These tools were all open-source, UNIX-based applications intended for use amongst international researchers and not conceived for the lay person.

The term "broadcast" has an entirely different meaning in internetworked computers than it does in the traditional radio/television sense. I will use the term broadcast generally to indicate the traditional meaning of the word. Broadcast in the very specific sense of Internet Protocols means to send some information (packets) to all hosts on the network without discretion. When we 'broadcast' audio on the Internet this is certainly not what we are doing.

For transmission of real-time media over Internet Protocol (IP) we do not in fact "broadcast" but we use either the "multicast" protocol or a "multiple unicast" format. Multicast is in fact closest to what we traditionally think of as "broadcast" and it is in fact the preferred method for real-time media distribution on the Internet. Multicast sends data to a certain (class D) internet address and hosts who wish to tune in request a stream from that address (oftentimes based on session directories which can be made available). Multiple unicast involves sending one unique stream of data per client requesting it. The immediate situation which is evident here is that the bandwidth used increases arithmetically in proportion to the number of clients receiving a multiple unicast transmission whereas it remains relatively static for any number of clients in a

multicast session. Unfortunately multicast is not practicable over the majority of networks due to the fact that it is not enabled on the majority of internet routers.

technology

During the planning of this project, I researched various technologies which could potentially allow me to realise this project. The key parameters for selection of which technology to use were: appropriateness, cost, reliability and availability. Following is a summary of my inquiries into enabling technologies for brainwave measurement, internet transport protocols and musical transport protocols.

technology : brainwave measurement

I investigated devices and techniques for measuring brainwaves. The process is in fact quite simple, proper methodology is however crucial in order to get a meaningful result. Human brainwaves can be measured by putting electrodes at key points on the scalp and measuring the microvoltages which appear there. To minimise extraneous noise and interference a ground electrode is normally placed on one earlobe of the corresponding cranial hemisphere. Differential voltages are measured between different points on the human skull. Electrodes are normally placed on the frontal, temporal, parietal and anterior lobes. An electrolytic paste is normally applied to facilitate good electrical and physical contact. Traditionally, brainwaves have been measured using highly sensitive and precise analogue electronic devices which recorded the results onto a paper trace which could then be later analysed by a neurologist. With the advent of sophisticated digital signal processing techniques this method has been largely phased out. Neurologists today will more likely employ some form of digital capture of data which can be analysed on a digital computer workstation. This

is in fact what I am doing but at a later stage in the process.

The principles of Digital EEG capture and analysis of continuous brainwave data are very similar to those used in digital audio reproduction and analysis, the primary differences being the means of data capture. analogue signal levels, and the sampling rates and sample bit depths which are commonly used.[18] Historically, one of the key limiting factors in sampling rates for Analogue EEG was the highest frequency at which the writer pens could physically oscillate which is around 70 Hz. Standard sampling rates used in digital EEG systems are between 200-400 Hz as most of the frequencies of interest lie below 100 Hz. A 400 Hz sampling rate thus gives us a comfortable Nyquist limit of 200 Hz. Below which we should be able to faithfully reproduce sampled waveforms.

In order to make music with brainwaves, I have investigated many software and hardware products which enable one to capture and analyse brainwave data. The only software which I found which was designed to allow for the kind of interaction which I was looking for is that which comes with the IBVA[19] system - a $2000 US hardware/software combination which nonetheless could not be considered 'professional' or 'scientific'. This software does perform a real-time FFT analysis of the incoming brain data and can then send out control messages via MIDI.

One major limiting factor of this genre of data acquisition device is that the sampling rate, sensitivity etc. are pre-determined by the hardware and software

---

[18] Fisch. pp. 127-153

[19] http://www.ibva.com

configuration which is not modifiable. The software allows a certain amount of flexibility in mapping the FFTs to MIDI but one is still limited to working within the kinds of data which MIDI can easily represent and to the preprogrammed software and hardware configuration which does the translation.

technology : internet protocols

IP (Internet Protocol)

Internet Protocol is the unifying layer of the internet; all devices on the internet must communicate via Internet Protocol. It is a set of rules and conventions for addressing packet messages between different endpoints on the network. The Internet today uses Internet Protocol version 4 (IPv4) which cannot easily guarantee a fixed amount of bandwidth over a given period of time, that is to say: it is difficult to guarantee a Quality of Service (QoS). It also has an address space which is limited to a theoretical maximum of 2^{32} possible hosts. These issues will be resolved to some extent with the imminent upgrade to IPv6.

It is interesting to compare IP to the IEEE 1394 (Firewire) protocol in which 80% of the bandwidth is reserved and guaranteed for isochronous (real-time) data and 20% is reserved for asynchronous (best-effort) transfer. With a protocol such as this it is possible to guarantee bandwidth and QoS (which are reasons why it is so popular for digital video interconnections).

TCP (Transmission Control Protocol)

Internet Protocol is a digital packet-switched protocol unlike the old analogue telephone system which is circuit-switched. Packet switching is more efficient but has problems such as congestion, redundancy and out-of-order packets. Current packet-switched networks and their protocols have been engineered primarily for robustness, reliability and data-integrity. This is most strongly manifested in modern TCP networking stacks. Much as these characteristics are honourable aims, they tend to work against the delivery of real-time media over the Internet. Many popular internet application layer protocols such as http, ftp, telnet, etc. need reliability and are thus built on TCP and use asynchronous data transfer and best-effort service. That is, the data packets are delivered as quickly and reliably as possible without concern for the temporal relationships, packet order or inter-packet jitter.

In delivering real-time data such as audio, we are concerned with time-sensitive media, which demands isochronous data transfer. Data packets must be received in-order and in the same relative rate and timeframe in which they were sent. It turns out that the existing TCP/IP protocol stack does not serve us well in this instance due to it's built-in error correction, flow-control, and (ironically) packet-ordering. TCP will continue to re-request a dropped packet until it is received. This will throw off the timebase of isochronous operations: by the time we get the missing data it is already too late. TCP also does not lend itself well to send-to-many operations. What is needed are more lightweight protocols. UDP is usually chosen because it is lightweight, connectionless and does not impose any unnecessary functions between the application and the network.

UDP (User Datagram Protocol)

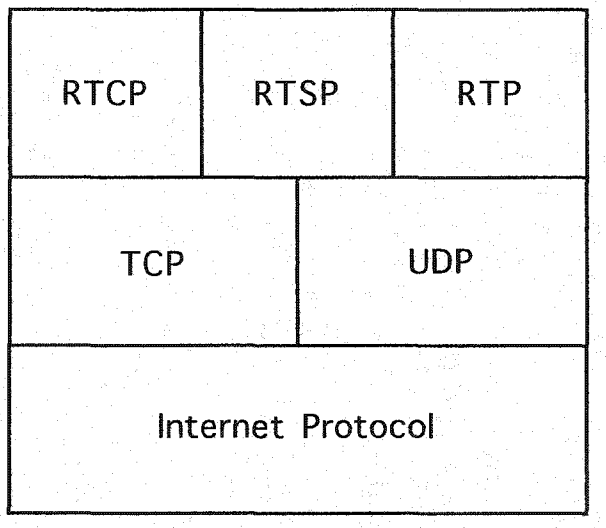
UDP is a simple connectionless packet oriented datagram (unlike TCP which is stream-oriented), each output produces exactly one UDP datagram which in turn produces exactly one IP datagram. UDP adds source and destination port numbers, a checksum and time-to-live (TTL) to the standard IP datagram. There is no reliability provided but there is a checksum provision. Higher level protocols which use UDP for transport: include daytime, dns, nfs, rtp, audp and otudp. UDP has a much smaller packet header than TCP, is connectionless and provides no error checking or correction. UDP/IP is a fast, lean protocol and is well suited to the delivery of real-time media which is why most current schemes use it. UDP does not, however, offer any services which might be of help in the delivery of real-time media.

RTP (Real-time Transport Protocol)

The Real Time Transport Protocol (RTP) was proposed by the IETF Audio-Video working group in 1996 (rfc 1889 and 1980). This provides a means for end-to-end delivery of media payloads in real-time applications. Real Time Control Protocol (RTCP) provides for feedback on performance and quality of data transmitted via RTP. Real-time Transport Protocol is a simple payload format for various real-time media streams such as audio and video. It normally rides on top of UDP and does not provide any increased reliability over UDP but does identify media formats, sample rates and provides packet timestamps for ordering purposes.

RTSP (Real Time Streaming Protocol)

A newer protocol developed jointly by Henning Schultzrinne of Columbia University, Real Networks and Netscape is the Real Time Streaming Protocol (RTSP). This protocol acts as a remote control for communication between clients and remote media servers to control such functions as Play, Pause and Record. RTSP usually uses RTP for payload transport but does not generally carry payloads itself. RTSP itself is generally running on TCP. The RTSP and RTCP connections are for control information and contain relatively little data so there is not a large performance loss in basing these on the TCP layer. As RTP is concerned with the actual transport of media payloads in real time and is moving a lot of data, it is based on the more efficient UDP protocol. RTSP uses a syntax similar to HTTP but unlike HTTP web servers the RTSP server must maintain state in its relationship to each client.



a simplified RTSP protocol stack

The overall functioning of an RTSP system works something like this:

1. A client requests a media stream via RTSP
2. The server and client negotiate via RTSP
3. The server delivers the real-time media via RTP.
4. The client and server communicate via RTCP in order to monitor and improve the quality of media delivery according to bandwidth restrictions and lost packets.
5. The client and server communicate via RTSP in order to stop or pause the stream and to teardown the session.

technology : music protocols

MIDI

The MIDI standard, originally proposed and implemented in the early 1980s, has proven to be durable and satisfactory as a protocol for musical event control largely due to its inexpensive, robust and open implementation. For several reasons, however, I feel that MIDI is not useful or interesting for me to work with in the context of InterHarmonium and I feel that there are already better, more flexible, alternatives currently available.

The MIDI data transmission rate is certainly sufficient for passing brainwave data but is particularly limiting in the possible ways of working with and expressing that data. The MIDI transmission rate of 31.25 kbits/s would offer me a theoretical 1953 samples/s at 16 bits resolution which would give a Nyquist frequency of approximately 976 Hz. This is clearly sufficient for the bandwidth of brainwave data and could in fact easily support 8 channels of

brainwave data. Why then, would I not use MIDI for this purpose? It is primarily a question of flexibility and extensibility. I want to expand or enlarge this system at a future time and MIDI, being a legacy standard as it is, imposes many arbitrary limits which may not be an obstruction at this point in time but could clearly become a problem later on.

MIDI was initially conceived to be inexpensive to implement in a wide variety of consumer hardware devices, specifically electronic keyboards, it used an inexpensive digital serial logic chips which was popular at that time - the UART controller. The limited data rate of 31.25 kbits/s forced the designers of the MIDI protocol to use various "kludges" to pack as much information as possible into that limited bandwidth. These design compromises today make the standard seem inelegant, rigid and confusing. In theory a MIDI signal could be sent over a standard 56 kbit modem. In practice, due to the unpredictable nature of the public Internet, it is not likely this would be very tractable.

ZIPI

There have certainly been other attempts to conceive a better protocol than MIDI for control of musical events; even at the inception of MIDI there were harsh criticisms of its shortcomings. The ZIPI protocol[20] was for a time developed by Zeta Music Systems, CNMAT and other collaborators but was later dropped apparently due to legal complications and for other technical reasons.

---

[20] Selfridge-Field. p.610

After the problems encountered with ZIPI, some of the same people who worked on ZIPI have contributed to OpenSoundControl (OSC). OSC was openly specified and sourced in the hopes that it might be more widely adopted.

OpenSoundControl

OpenSoundControl was conceived as a replacement for MIDI and as a possible cure for some of the problems of this ageing yet pervasively useful protocol.

> OpenSound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology. Entities within a sound synthesis or processing system are addressed individually by an open-ended URL-style symbolic naming scheme that includes a powerful pattern matching language to specify multiple recipients of a single message. We provide high resolution time tags and a mechanism for specifying groups of messages whose effects are to occur simultaneously. There is also a mechanism for dynamically querying an OpenSoundControl system to find out its capabilities and documentation of its features. [21]

OpenSoundControl is a protocol for musical specification and control which was designed to be very flexible and independent of the underlying transport layer. OSC uses a URL-like syntax and supports standard UNIX features such as file name globbing. The primary implementation of OSC uses the UDP protocol for transport. As mentioned, UDP is a lightweight and fast protocol but does

[21] http://cnmat.CNMAT.Berkeley.EDU/OSC/

not have the reliability which the TCP protocol has. When I asked Matt Wright - the primary author of OSC - about the use of UDP as a transport for OSC, he said they had not encountered any problems with lost or dropped packets in their tests. Granted, these tests primarily took place on the network of the UC Berkeley campus or over the relatively reliable inter-university networks between centres like CNMAT (UC Berkeley) and CCRMA (Stanford). Certainly one would find that across more heterogeneous and complicated network spans there would be some packet loss and jitter.

AUDP (Audio UDP)

AUDP is a protocol developed for efficient transport of digital audio over a Local Area Network. It was developed at IRCAM and integrated into the jMax music application environment. It is based directly on the UDP layer and does not use the RTP layer at all. It was determined that RTP added too much overhead to the packets and did not offer any significant features for this particular application.

Comparison : RTSP, RTP, OSC, AUDP

At this point I would like to make a comparison between the RTSP and RTP protocols and the OSC and AUDP protocols. RTSP and RTP were both developed under the aegis of the IETF (Internet Engineering Task Force) as general-purpose control and transport methods for real-time delivery of audiovisual sessions over the internet. OSC was developed specifically as a musical control protocol for the academic computer music community. AUDP

was likewise developed as a real-time transport protocol for digital audio at IRCAM for the specific needs of the academic computer music community.

RTSP and OSC

RTSP and OSC are both similar in that they concern themselves primarily with sending control information to remote hosts on an IP network. Whereas OSC is primarily intended for use on a LAN, RTSP was designed to be reliable over the public internet. OSC is primarily delivered over UDP whereas RTSP is primarily delivered over TCP. RTSP was primarily conceived, using a quasi-HTML syntax, for session control between media servers and clients over the internet. OSC was conceived as a very open, flexible system for transmitting musical (or other) control data over an IP/Ethernet LAN. While RTSP is more robust for one specific purpose, OSC is more flexible and lightweight.

RTP and AUDP are also both similar in that they concern themselves primarily with transporting real-time media streams to remote hosts on an IP network. Whereas AUDP is primarily intended for use on a LAN (between hosts running the jMax software application), RTP was designed to be reliable over the public internet. Both protocols utilise UDP but RTP includes more provisions for error correction and recovery whereas AUDP is more lightweight and assumes it is running on a LAN where packet loss is not likely to be a problem. RTP includes the RTCP (Real Time Control Protocol) which can monitor and adjust stream settings to improve media delivery to clients whereas AUDP includes no such provision. AUDP was conceived as a flexible, open system for transmitting

only sampled digital audio for low-latency musical applications over a LAN whilst RTP was conceived as a general-purpose protocol for delivery of real-time media streams to clients on the public Internet.

As it turns out in my project, I am using UDP, TCP and OSC. I would have been interested to experiment with AUDP but I had only recently become aware of it and attempts by the system administrator in the Music Faculty to install jMax with AUDP on our research Linux machine have been fruitless.

project

In the following section I will outline the trial and error processes which led to the crucial decisions about how to implement the actual software project. This was an often arduous process somewhat like navigating a hostile foreign terrain with no signposts or guides to point the way.

project : experiments

I began experimenting with networked music systems when I discovered the 'W' protocol and object in the Max environment. 'W' was an unsupported client-server protocol written by David Zicarelli and included with Max 3.5. When the appropriate server was available to act as a mirror one could send Max messages to other connected clients on a network such as the Internet. I successfully ran a patch which connected from the McGill University Undergraduate Music Computer Lab (UCL) to a server running special software and then reflected back commands and MIDI notes to the 8 other computers in the UCL. The W server software[22] was running on a Silicon Graphics machine at l'Université de Montréal (having been compiled and installed there by Alexandre Burton). I was seeing a latency of 50-100ms based on my own perceptual impressions at the time. This is already problematic for real-time musical performance. Considering that this represents a round trip just across Mont Royal on the RISQ (Reseau d'Information Scientifique du Québec) network it is a surprisingly high latency. I pinged their web server host to see what the ping times were like and got a somewhat better result:

[22] http://www.synthesisters.com/download/wproj.sit

```
[brouse@improv ~]$ ping www.umontreal.ca [...]
21 packets transmitted, 21 packets received, 0% packet loss
round-trip min/avg/max = 1.6/3.1/11.6 ms],
```

It is nonetheless possible to imagine musicians playing together in such a situation as long as they were comfortable working with the built-in delay times. I have managed to compile the W server and Max external myself from the source and had it running on "`jasper.music.mcgill.ca`" - a Macintosh G3 running the Mac OS X Server 1.2 operating system. On this system I found the w server software was - likely due to a software bug - consuming 95% of the available cpu power.

Name	User	Status	%CPU
wserver	root	Running	95.2
WindowServer	root	Running	2.4
ProcessViewer	root	Running	0.8
rotatelogs	root	Running	0.0
apache	www	Running	0.0

wserver hogging the CPU

It was an interesting experiment but as the "W" object is not supported in the Max environment and the server software only seems to run well on SGI this option was not interesting in the long term. There was once a tcp Max object created by David Rokeby capable of sending messages over tcp/ip but which is neither freely available nor supported. Quite recently, however, Norman Jaffe has released a suite of TCP/IP objects which are freely available and seem to be updated regularly.

I had for a while also been aware of the OpenSoundControl (OSC) protocol and the means to implement it over various networks in the Max environment:

otudp (Open Transport UDP). These were developed at CNMAT (the Center for New Media and Audio Technologies at the University of California, Berkeley). Ultimately, I found OSC to be much more robust and flexible than the 'w' protocol.

In my tests using OSC, I found that it worked flawlessly over the McGill Music LAN where every machine has a direct point-to-point path to every other machine via IP over a homogeneous Ethernet link layer. I did not encounter any packet loss at all although OSC is using the 'unreliable' UDP to send its packets. This is consistent with the observations of Matt Wright in using OSC at CNMAT over the Berkeley LAN. As OSC was conceived to communicate musical control signals under such conditions it seems to be doing its job properly. OSC was not, as such, conceived for use over the heterogeneous public internet where the only constant is the use of Internet Protocol and packets pass over many heterogeneous transports including routers, firewalls and various underlying link layers (Ethernet, FDDI, SONET, POTS, ATM etc.).

It was in fact somewhat surprising to me how well OSC performed over the McGill LAN. I did not experience any packet loss at all in inter-machine communications. It also performed very well in a loopback configuration sending to the same host (localhost : IP 127.0.0.1). In fact, as a means of sending control signals between different applications this easily outperformed MIDI on the same or different machines. It also had much better performance and reliability than the InterApplication (IAC) bus provided via OMS (Open Music System - MIDI software for MacOS from Opcode Systems). As MIDI has a fixed, relatively slow, data rate this is not too surprising. UDP/IP over fast

ethernet on the other hand can benefit from a theoretical maximum of 100 Mbits/s bandwidth with very little overhead.

In experiments from my home computer on an ADSL connection (Sympatico) to McGill I found I could not establish even my remote IP address using my reflector patch at McGill. In checking the status window later on the McGill machine I found that there had been corruption of the data sent over UDP. The path from my home to the McGill Music LAN takes at least 12 hops through a variety of routers which inevitably contributes to this packet loss (especially as many routers are not very friendly to UDP traffic). In order to enable two-way communication between arbitrary internet hosts I need a reliable way to establish at least the remote IP address of given hosts. It became apparent that for this purpose I needed to communicate with the remote machine via TCP/IP. I decided to use the said tcp objects made available by Norman Jaffe.[23]

I was thus able to implement my own mini-protocol which establishes those things which must be reliably established (such as the IP addresses of participating machines) via TCP/IP and then sends that musical data which depends on timely delivery over UDP/IP by way of OSC. It is interesting to note that this is similar to how the RTSP and RTP protocols interact. RTSP negotiates streaming media sessions over TCP and then tells RTP to transport the actual audiovisual media over UDP for efficiency. It then - via TCP - enables error detection, correction and monitoring of Quality of Service (QoS).

[23] http://www.opendragon.com/Pages/MaxObjects.shtml

project : design

The principle idea and challenge when I set out to assemble this project was to build a music system which could accept real-time control signals from human brainwaves and send them to a client computer which would then synthesize sound from the received (brainwave) control signals.

At the outset my objectives were much more ambitious than I could practically achieve. In the end, I felt it was more important to build a simpler but usable and robust networked music system than to attempt something very complicated which might or might not work well.

In deciding what software packages to use to implement this project, I looked at Max/MSP, SuperCollider, pd, jMax, QuickTime, OpenSoundControl, CAST, as well as the possibility of directly coding something in CodeWarrior, ProjectBuilder or RealBasic.

Important in this choice was that the chosen tools be flexible, fast to prototype, powerful, not too expensive to use and able to be diffused via a free runtime or the ability to make a standalone application. I would have preferred to develop my project under Mac OS X for reasons of stability and network performance but it seems that in particular audio and MIDI music application developers are having the hardest time at transitioning to OS X. I suspect that this is largely due to the fact that the audio and MIDI APIs for OS X are still not published or finalized (even though has been a shipping product for several months). It turns out that the applications which I need are almost all running under Mac

OS 9.

I would prefer to make the end-user agent software able to run on a multitude of computer platforms (Windows, Linux, etc.) The only feasible way of doing this would be to write in Java, or possibly use a development environment like RealBasic which takes care of most of the cross-platform issues (compiles for OS 9, OS X, and Windows from the same source project). RealBasic does support MIDI, QuickTime audio and streaming natively but would be very difficult to include any sophisticated algorithmic functions and I/O flexibility. CodeWarrior and ProjectBuilder are obviously both very powerful and sophisticated development environments but are not amenable to experimentation and prototyping. SuperCollider is a very powerful, sophisticated, and efficient environment. It is very flexible, has very high-quality sound output and supports OSC. It is however Macintosh-only and does not allow the creation of binary applications but does have a free runtime version. The language is powerful but somewhat cryptic and not well-documented. The cryptic nature of the language was confirmed when - while I was attending the SuperCollider Nightschool at CNMAT in July, 2000 - Alberto da Campo, one of the most active and knowledgeable SuperCollider users, confirmed that he himself only knew about 30% of the language. Thus, it seems that the only person who really knows SuperCollider is its creator, James McCartney.

I finally chose to do the majority of my work in Max/MSP largely for the speed and flexibility in implementation of complex projects which that environment offers. Other key advantages are the large toolkit of available signal analysis

and processing tools which are already available in this environment, and the possibility to create a stand-alone application. Something developed in Max/MSP could nonetheless be reasonably easily adapted to another framework or written directly in C. Max/MSP is mature and supports all of the things which I need: control, MIDI, audio DSP and OpenSoundControl. There is also a large user base which tends to be very helpful and knowledgeable. Less desirable about Max is that it is not very efficient at real-time audio processing (but getting better), and that the audio and signal processing quality is not always as good as I would wish.

My initial decision in developing this software was to use the latest versions of my chosen software tool: Max/MSP (versions 4/2 respectively). David Rokeby once described as a "deal with the Devil" his rewriting and recompiling software he was using for his installation "The Giver of Names" at Galerie Oboro right up until a few hours before the installation was set to open to the public. I had initially made such a decision to adopt new 'bleeding-edge' technologies for the implementation of my thesis project. Apart from the pure adrenaline rush of living dangerously, there were salient reasons why this choice was made. The new versions of Max/MSP would have allowed me certain advantages which I imagined would make the final product more useful and powerful: the new audio driver model allows much greater flexibility in audio signal routing and the ability to resample a waveform at a different sampling rates are two key enhancements which would have been very useful in my system.

New technologies always offer compelling reasons for their adoption but the

fact remains that software rarely behaves 'as advertised' and it is one thing to take risks using something with which one is intimately familiar and quite another when one cannot be entirely sure how the software will behave.

At the end of the day, I want a system which will work and do so reliably. After a significant amount of frustration with bugs and inconsistencies in the aforementioned software products, I decided to take a more conservative tack and to use software tools with which I am familiar and which I know will behave in a predictable manner. To this extent, I chose to implement my project primarily using the Max/MSP 3.6.2 software version which has been stable for over one year.

In order to implement my system I determined that I needed to support a certain number of specific functions/capabilities:

- There will be a 'server' software which will measure the brainwaves, perform some basic analysis, maintain a connection with the 'client' software and send out the brainwave data to the client in real-time
- There will be a client software which will initiate a connection with the server
- The client and server will negotiate a reliable 2-directional communications link between two networked hosts via TCP
- The client and server then negotiate a less reliable but more high-performance real-time data connection over UDP

I am using an analogue electroencephalogram (EEG) machine to amplify and filter the raw brainwaves from their original amplitude of around 5-10 μV up to a range of 500-1000 mV where they can be brought directly into the Max/MSP

software via a standard audio analogue-to-digital converter. Human brainwaves typically contain useful spectral information in the range from 0-128 Hz with the majority of the energy lying below 40 Hz. This spectral range is subdivided into several smaller bands each of which have particular significance to the trained neurologist. The frequency spectra are generally considered to break down as follows: delta, 1-4 Hz.; theta, 4-8 Hz.; alpha, 8-13 Hz.; and beta 13+ Hz.[24] The band with which I have mostly concerned myself is the Alpha wave band comprising those frequencies between 8-13 Hz .[25] I am particularly interested in this band because of the specific physiological/somatic states in which alpha rhythms are most strongly produced. In order to produce strong alpha waves, the subject must be in a deeply meditative, non-visual state.

Due to practical and economic factors, I decided to implement the capture of analogue brainwave data using the Grass Model 8 EEG machine which I have in my possession. I now know how to operate the machine effectively and it provides high quality capture amplifiers and filters. This machine can be configured to capture many sorts of varying voltage sources including: electroencephalogram (EEG), electrocardiogram (EKG), electromyogram (EMG), and polygraph (lie detector)[26] It could also be used to measure other varying potentials such as Galvanic Skin Response (GSR), Eye Movement Potentials (EOG)[27] or even electrical potentials in living plants or other organisms. The major downside to using the Grass EEG is that it is rather bulky and not easily portable. It also does require a somewhat careful setup each time it is used. I feel these issues were outweighed in this case by the precision,

[24] Fisch. p. 167
[25] Fisch. p.124
[26] Grass. pp. 16.2-16.3
[27] Eaton. p.4

flexibility and cost to me ($0) of the instrument.

I have implemented the software such as to allow different methods of data capture. I am assuming that whatever kind of analogue data is being captured it can be brought in using a standard audio A/D converter. This could be the built-in audio ports on a Power Macintosh computer, another sound card or an external breakout box such as the MOTU 2408 via a host PCI interface or 828 via FireWire (IEEE 1394) interface. In my case I have used all of these methods at various times. Audio input is either via the Sound Manager library built into the Mac OS system software or via the Steinberg ASIO (Audio Streaming Input/Output) software. Ultimately, any sound input which can be used by Max/MSP is sufficient for my purposes. For portability, it is nice to be able to use the built-in Macintosh sound inputs which can, albeit not entirely faithfully, capture the required range of frequencies but will only allow for two channels of input. Seeing as an EEG is normally done with 6 or more channels of sensor input this already presents a limitation. Portability is also rather moot until I have a more portable analogue EEG analogue capture device. I did investigate building my own custom electronics for EEG capture but concluded that, for the meantime, this would entail too great an expenditure of time and money. Newer versions of the MSP software can allow the A-D capture device to be reconfigured dynamically in software so this would permit me to be flexible in my future implementations.

I concluded that the high-precision electronics of the Grass analogue EEG (DC-coupled) combined with good quality audio A-D converters (which are AC-coupled but can capture the frequencies I am interested in) and the Max/MSP

software was a good combination for economic, technical, artistic and compositional reasons. It would have been possible to build reasonably good quality analogue amplifiers and filters myself instead of using the Grass EEG machine but in this case the EEG machine was entirely free, of very high quality, quite functional, and came with a history of having actually been used in the treatment of patients. I also found it to be rather aesthetically pleasing.

I have decided to use TCP to establish contact between remote hosts and for sending of control information and other crucial communications and to use OSC/UDP to send real-time data. The use of UDP for real-time data entails some potential data loss but allows greater efficiency in communication. I have used the TCP objects from Norm Jaffe as previously mentioned which have solved the problem of reliable communication but have introduced some other mysterious problems of their own into the patches which seem to be related to cached buffer state and buffers not being released in a timely fashion. OSC has been employed for real-time data transfer.

I have devised my own mini-protocol in order to allow negotiation between server and client. I have attempted to model this based on my knowledge of existing protocols and to keep it as simple, flexible and self-descriptive as possible. A method is provided to establish initial contact and auto negotiation of IP addresses and ports between the client and server. When this negotiation occurs successfully, a confirmation message is returned to the client. At this stage I have only enabled a one-to-one server-client relationship. It is possible in the future I could extend this to allow multiple clients of a single server.

Up to 8 separate communication channels can be opened at one time and maintained. The 8 communication channels have been multiplexed so that they can run over the same UDP port. For a multi-client implementation all clients would communicate via a common TCP port but use unique UDP ports. The server and client have been constructed so as to be capable of operating either in a loopback (on the same computer) mode or across multiple hosts. The port assignments are currently hard-coded but could be dynamically assigned by the server as requests for connections are made in a multi-client version.

As much as possible, I am trying to implement this software (in Max) using good object-oriented design principles. I would summarize these as follows:

- object encapsulation
- privacy of local data
- message passing between objects
- multiple instantiation of objects
- hiding of complexity in objects
- use of classes and subclasses

This is done primarily by extensively using "abstractions" or sub-patches which are saved externally and then loaded (instantiated) into the main patch. This allows multiple identical copies of that object to be used. I feel this approach to Max programming is somewhat like programming in an OO style in a non OO language. (i.e. it is possible in straight C to use data structures as "quasi-objects" and gain many of the advantages of OO programming while retaining the efficiency and clean syntax of C). The other advantage of this approach is

that one can built objects which provide a specific, limited but reliable, tested and reusable functionality which can be used to build up more complex systems. This also helps enforce the "hiding of complexity" which enables good OO design. In this manner, one can have multiple copies of an abstraction which are instantiated into the "main" patch rather than use sub-patchers in the patch itself. This gives the ability to use multiple instances of the same object which will be identical and - if properly debugged - reliable.

The Max environment does not allow sub-classing as such but I feel that by following these principles it will be easier at a future time to re-implement this software prototype in a full OO development environment with all the attendant advantages of efficiency and full object oriented behaviour. All this being said and despite my best object-oriented intentions, Max is very much a procedural programming environment which ultimately enforces a procedural programming style.

I have found it necessary to incorporate various unsupported 3rd party Max external objects into my project to implement various functionalities which are not present in the default Max/MSP environment. The external objects which I have added to the Max/MSP environment are: `"OpenSoundControl"` and `"otudp"` to implement OSC and UDP functions; `"tcpClient"` and `"tcpServer"` to implement TCP; `"centroid~"` to provide a means of estimating spectral content; and `"smooth"` to provide a means of averaging integer values over variable amounts of time.

Standard sampling rates in modern digital EEG systems are between 200-400

Hz [28] This reflects the fact that most of the frequencies of interest lie below 100 Hz. A 400 Hz sampling rate then gives a comfortable Nyquist limit of 200 Hz. It turns out that sending 400 floating-point numbers/s over OSC/UDP does not seem to pose a problem (at least not on a LAN) and it seems to be well within the capabilities of OSC. The one thing which I had to do to allow this much data to pass without packet loss was to increase the default buffer and packet sizes of the OpenSoundControl and otudp objects significantly. (OSC to its maximum=32000 bytes default=1024, otudp to 2048 buffers - default=20 - with a size of 512 bytes each default=1024). These values may seem excessive but were determined on an experimental basis in order to enable me to send the amount of data necessary with minimal packet loss. I have been quite liberal in allocating memory to the Max application and to the otudp and OSC objects as network performance is crucial to this software and neither OSC and otudp nor Max/MSP have ways to dynamically allocate more memory than that which is allocated at their instantiation.

There are surely more efficient means of accomplishing this (e.g. writing my own custom object or protocol) but for my purposes, OSC is general and efficient enough that for the sake of simplicity I have decided to use it. OSC, in fact, entails very little overhead on top of straight UDP, it merely serves to format the packets so that they can be efficiently sent by otudp. OSC is being, albeit slowly, adopted by different software vendors as an alternative - if not replacement - for MIDI.

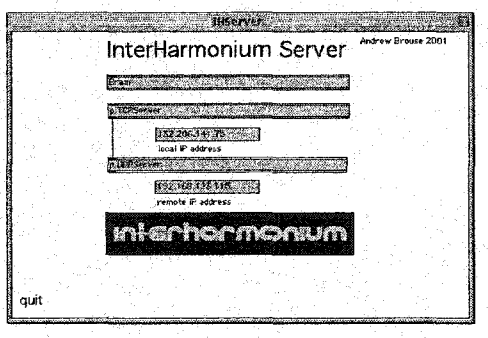
---

[28] Fisch. p.129

project : implementation

For the InterHarmonium there are two main software components - a server and a client called respectively InterHarmonium Server (IHServer) and InterHarmonium Client (IHClient). These have both been built as Max/MSP patches which can be run using the MaxMSPPlay runtime application. They have also been compiled into standalone applications which do not require any additional software to run.

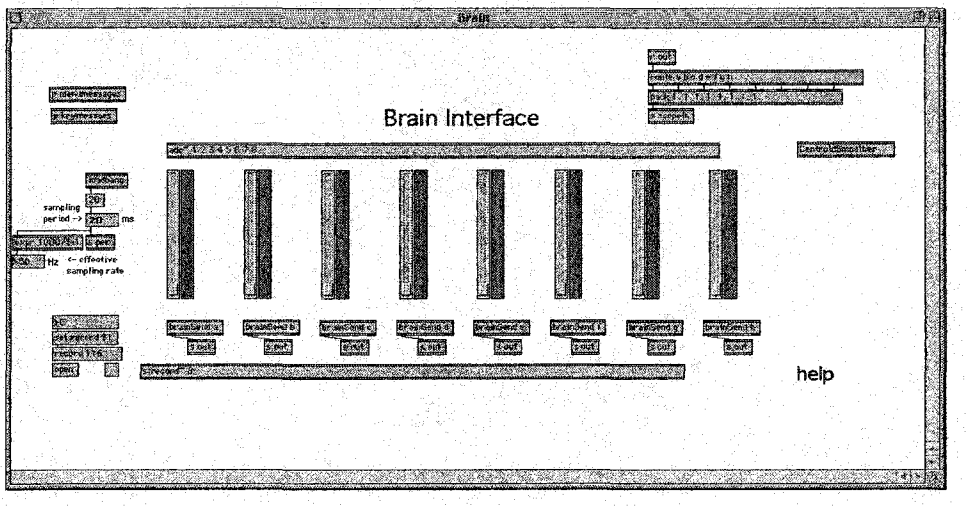
InterHarmonium Server



The server consists of three key parts: a data acquisition "Brain" module which gathers data from up to 8 analogue inputs, downsamples it and prepares it for sending to the remote client, a TCP server which negotiates a reliable connection with the remote client, and a UDP server which formats and sends real-time data to the remote client.

The "Brain" subpatcher contains all of the logic and functional units which handle capturing and formatting of the brainwave data. Inside the first level of the subpatcher there are two abstractions: brainSend and prep. Prep merely prepends the argument passed to it onto whatever message comes into the

patcher and then sends it out. This allows me to send multiple channels of brainwave data on one data transmission channel and to be able to sort them out on the other end (using route). This could be thought of as a rudimentary form of multiplexing although using sample frames would provide more data coherence. The brainSend abstraction contains in itself two other abstractions: brainIn and simBrain. BrainIn opens an analogue to digital input channel using the channel number argument which is passed to it from the parent patcher where the argument is the input channel # as recognized by Max/MSP. This object handles input of the raw EEG (audio) signal. The argument must be a valid integer channel number. This abstraction uses the `number~` object not only to display instantaneous input levels but also to resample the brainwave input signal using the interval message to set its effective sampling rate (which can be adjusted). The `number~` object sends out the instantaneous floating-point values of the signal at the given interval. The smallest interval or period which `number~` will support is 20 ms which translates into an effective sampling rate of 50 Hz and Nyquist limit of 25 Hz. Given the frequency spectrum of interest (alpha) lies between 8-13 Hz this is sufficient.



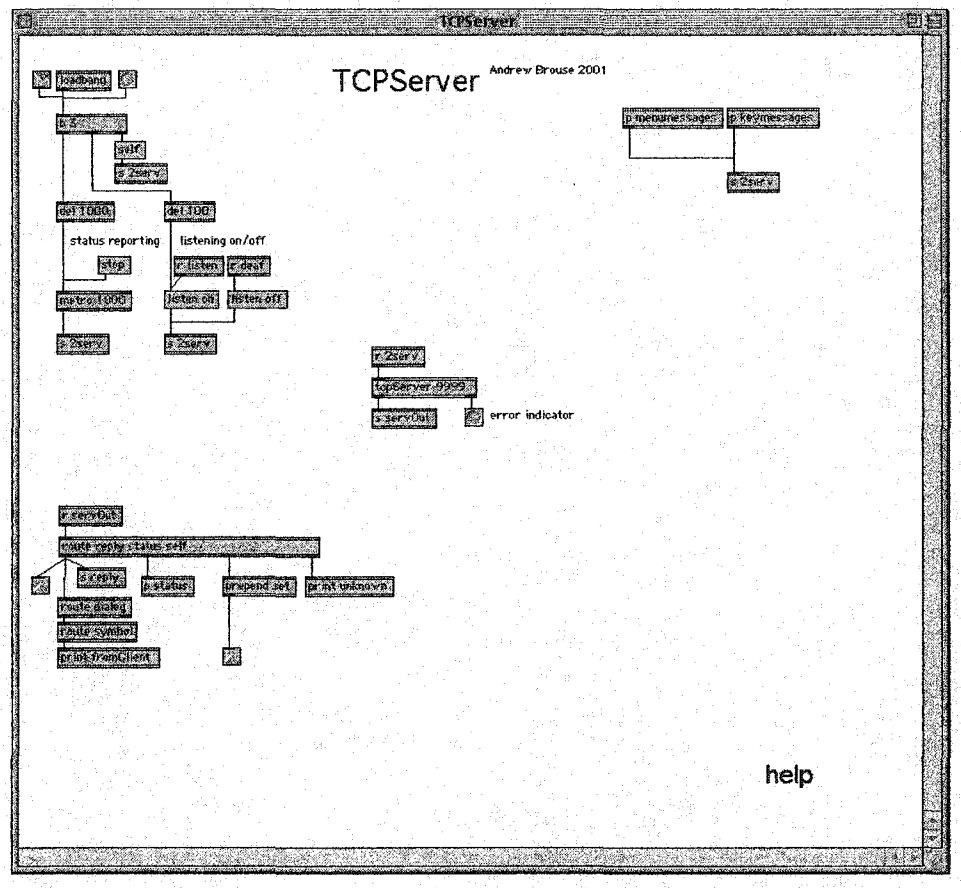
The Grass EEG machine already has very high quality analogue filters for isolating this frequency range and thus I have not implemented a low-pass or band-pass filter in software. This would be very easy to add at a future time if necessary.

The `simbrain` abstraction is primarily for setup and testing purposes. Connecting a human subject to an EEG machine is time-consuming and can be somewhat inconvenient for the subject and technician. The `simbrain` abstraction thus generates a pseudo-random signal in the same range as the alpha brainwaves and enables end-to-end setup and debugging of the system with no need for a human subject to be hooked up to the EEG machine. I have used the `drunk` object to simulate a random walk across the frequency range of concern. The `'z'` key on the computer keyboard will toggle between simbrain and the real brainwaves coming in via the analogue to digital converter.

I have used the `centroid~` external by Ted Apel, John Puterbaugh, and David Zicarelli to determine an averaged centroid frequency for the alpha brainwaves. By combining this with the 'smooth' object from David Rokeby, I can obtain varying degrees of smoothness or averaging of the centroid signal which I then use to determine a fundamental frequency of the signal to pass on to the `HarmonicGenerator`.

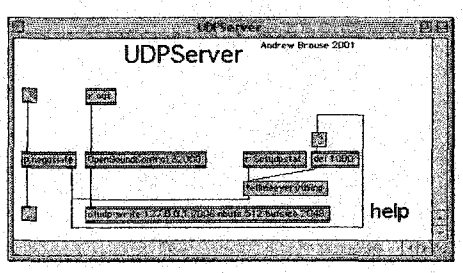
The `TCPServer` listens on an available port (9999 in this case) for incoming connection requests. When a request is received, the external object `tcpServer` negotiates with the client on the other end to establish a reliable connection. Once a connection is established with the remote client, data can be sent using

the 'send' command. This is easily done by prepending 'send' to whatever message needs to be sent. Integer and floating-point numbers do not require the send command as they are automatically sent. Apart from negotiating IP number and port number with the remote host and maintaining the connection, the `TCPServer` is also used to send short dialog messages, the centroid frequency of Alpha waves and a few other key control messages. Incoming messages from the client are sorted by using 'route' and are then dealt with as necessary. The 'self' command to the `tcpServer` object returns the IP address of the local machine. This allows me to test the configuration in a 'loopback' mode: with the server and client both operating on the same machine.



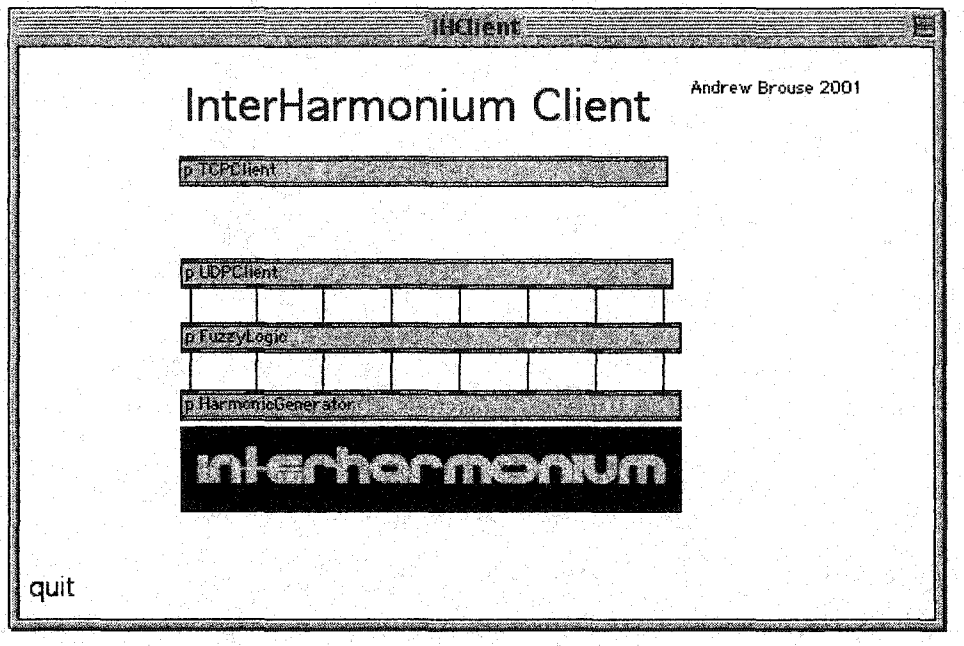
The `Negotiate` sub-patcher takes host and port messages received from the TCP server and formats them so that the UDP server's functional core, `otudp`, can understand them. `otudp` only understands messages of the format (`HOST IP_ADDRESS PORT_NUMBER`) which must be sent as one message. Negotiate is effectively the glue between the `TCPServer` and the `UDPServer`.

The `UDPServer` handles the transport of real-time brainwave data. Once a destination IP address and port have been specified, the UDP server just sends whatever data is passed to it along to the destination address. No verification of receipt is performed. The `OpenSoundControl` and `otudp` objects work together and thus, to pass any data from the Max environment through `otudp`, it is necessary to use `OpenSoundControl`. To a large extent I am not really using all the features of the OSC protocol as such but am using the object as a conduit for Max messages. The otudp status menu command will provide a dump of all current otudp vital statistics to the Max window. As already mentioned, OpenSoundControl has been given the largest buffer size which it will accept (32000 bytes) and otudp has been given a very generous number of buffers (2048) and buffer size (512 bytes). These values were arrived at by trial and error and seem to deliver the best performance and reliability. I have probably allocated more memory than is required but since these objects have no way to dynamically re-allocate memory, I feel this is the best course of action.



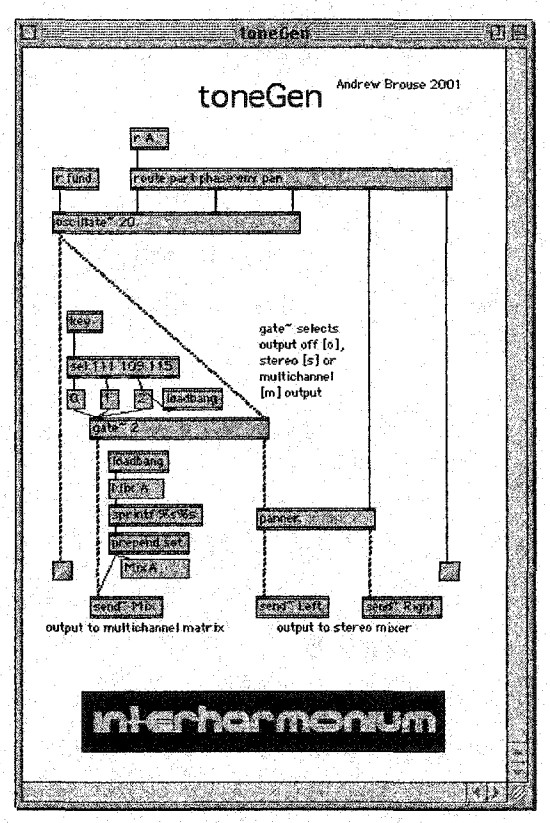
InterHarmonium Client

The InterHarmonium client software consists of four primary parts: the `TCPClient` abstraction which takes care of negotiating TCP network connections, `UDPClient` which handles receiving of data via UDP, `FuzzyLogic` which provide basic routing, algorithmic and mapping functionality and the `HarmonicGenerator` abstraction which takes the data from `FuzzyLogic`, parses it, and employs that data in turn to generate sound using its additive Fourier synthesizer.



This `HarmonicGenerator` abstraction is based on a similar idea to an application which I made in 1998 also called "HarmonicGenerator" which performed additive Fourier synthesis of pure tones. The purpose of this patch is to generate any arbitrary harmonic in relationship to a given fundamental frequency with specified phase, amplitude and panning settings. All tones are

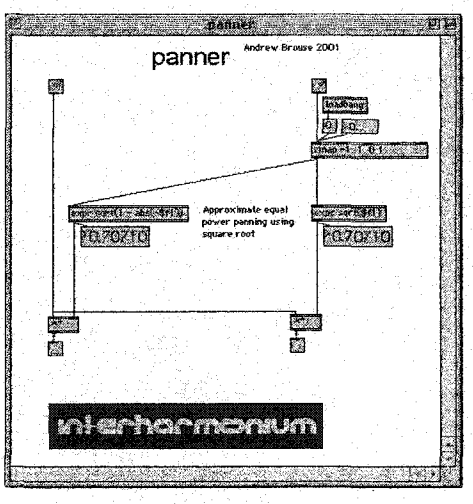
generated by the `toneGen` abstraction. The fundamental frequency is received by `r fund` in each of the tone generators as a floating point number. The `sender` abstraction takes care of formatting messages to be sent to each individual toneGen object. In this particular case, I have used 25 toneGen objects to provide a maximum of 25 distinct partials. This alone consumes approximately 45% of the available cpu power on my 333 MHz Macintosh G3 PowerBook and thus I would be very reluctant to provide any more tone generators. It is certainly possible to generate a very rich and complex sound with just 25 oscillators.



The `toneGen` abstraction takes two argument: the first being its symbolic identifier (in this case I have used upper-case letters) and the second being its default initial harmonic number (frequency ratio to the fundamental). It is

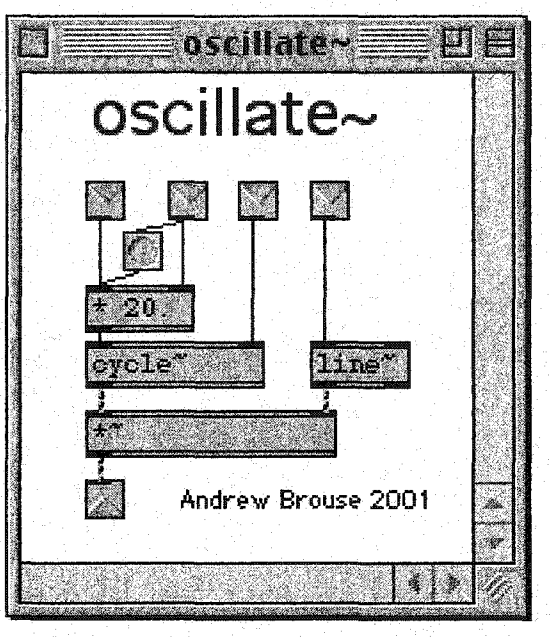
important to note that this should be specified as a floating-point number (i.e. 12.0 rather than 12). If it is initially specified as an integer, then only integer multipliers and frequencies will be generated. All toneGen objects receive the fundamental frequency via the `r fund` object. Each specific toneGen receives its partial (harmonic) number, phase, amplitude envelope and panning instructions which are sent to the same name as its identifier with part, phase, env and pan prepended respectively to allow routing of messages. This information is then passed to the `oscillate~` abstraction which actually generates the tones. A `gate~` object is used with a keyboard selector to allow for signal output routing to be either off [o], to stereo [s], or discretely output to a multichannel matrix mixer [m] which is not currently implemented.

A simple `panner` abstraction takes a floating-point input between -1.0 and +1.0 to represent the range from hard left to hard right panning. I have used a simple square-root expression to provide an approximation of equal power panning.



The `oscillate~` abstraction is a simple object which takes care of setting and updating the frequency of the harmonic in relationship to the fundamental, adjusting phase (0-1.0), and the amplitude envelope (0-1.0).

Output is conventionally routed to the stereo outputs of dac~. The tilde [~] key can be used to toggle audio on and off.



project : operation

InterHarmonium Server (IHServer)

When the IHServer starts up it initiates a series of actions in order to enable it to respond to client requests:

1. bringing in the sampled brainwaves and downsampling them
2. routing and preparing the brainwave data
3. opening a TCP port on the host computer and waiting for a connection
4. negotiating with a host when a connection is requested
5. responding to client requests to open a specified UDP port for sending data
6. sending out the raw brainwave samples
7. accepting other requests from the client (such as disconnect)
8. being able to dialogue with the client

As mentioned, the functionality of the Server is divided into three principal components: TCPServer, UDPServer and Brain with a negotiation/formatting component which links the TCPServer and the UDPServer.

The TCPServer is responsible for maintaining a reliable contact with the client and for sending all control communications as well as permitting dialog between the server and client. It is essential to have one reliable - if inefficient - channel of communication in order that vital information such as the client's IP address and UDP port number can be passed between server and client.

When the application starts, it sends a loadbang which first turns verbose mode

off, and then - after a 100 ms delay, turns listen on. After a 1 second delay, a metro object send a bang every one second to the tcpServer object. This effectively causes this object to poll its current status. (Bang causes a status output). The status output of the server is passed - via route - to the changes object which determines if the status has changed. If the status changes it will output that change at the polling interval. When the bound status message is received - indicating that the tcpServer object is bound to the specified port of the Macintosh IP stack - a bang is sent to turn listen on. This will ensure that the server will be put into listen mode regardless of how long it takes to bind to the TCP/IP stack. The active server listening port can be set as desired but this will cause any connected client to be disconnected. The server must always be started before the client. Menu commands are provided for seeing the status of the TCPServer and for switching verbose mode on and off.

The negotiate sub-patch accepts host and port numbers to be passed to the UDPServer. These numbers will have been sent by the client in order to initiate a UDP connection. In each case the token of "host" or "port" is appended to the beginning of each number in order to identify each message. Route is used at the receiving end to separate these messages and to strip off the token in order that the data might be used as appropriate.

Route strips off the 'host' and 'port' tokens and simultaneously routes the data to the appropriate branch in the processing chain. Both the IP address and port number are then set. When the IP address is set this is sent to an append object which contains the port number. This constructed address and port number then has 'host' appended to the beginning and we now have a message which

can be understood by `otudp`. Thus the port number must be set before the IP address because when the IP address is set the entire formatted message is sent out immediately.

The `UDPServer` is responsible for preparing and sending out the raw brainwave data as received from the Brain module to the specified Internet address. Because UDP is a connectionless protocol this module will just continue to send data to the specified network object without regard for whether it is being received or not. The key desired characteristic of this component is that it be as efficient as possible in delivering the real-time data.

Tagged data is sent to the `OpenSoundControl` object which arranges the Max data into an acceptable binary format and then passes this formatted data on to `otudp` which is configured as a 'writer' for sending data. `Otudp` has been configured by default to send to the loopback address (127.0.0.1) which allows one to run the server and client on the same machine for testing purposes. The client, however, will autodetect its own IP address and attempt to negotiate that address with the server when it starts up.

The Brain module handles input of sample brainwave data via the selected audio input, downsamples this to a range appropriate for brainwave data (50 Hz sampling rate) and handles tagging and routing of the data. It also provides a submodule, simBrain, which give a pseudo-random signal in the same range as an alpha brainwave which can be used during setup and testing when a live subject is not available for setting up the system.

Brain is configured to allow eight channels of analogue audio (brainwave) input which corresponds with many currently available analogue-to-digital converter interfaces. In this case I have used the MOTU 828 FireWire (IEEE 1394) interface for much of my testing.

Once the server has been started then the client can be set up to connect to it. You must know the internet address (IP address) of the server in order to use the client. The TCP port of the server/client has been hard-coded as port 9999 but could be dynamically allocated. The UDP port can be set by the client during the negotiation process.

InterHarmonium Client (IHClient)

In the 'main' patch of the InterHarmonium Client software, functionality is divided up into four principle sub-patchers: `TCPClient`, `UDPClient`, `FuzzyLogic` and `HarmonicGenerator`. Each of these sub-patchers encapsulates certain functions and passes messages and data along to the other functional units as necessary. This enables me to much more easily replace any one of the functional units at any time in the future. A 'quit' button is provided within the patcher as well as via a menu command.

The `TCPClient` patch establishes and maintains reliable network data connections with the InterHarmonium Server software. It is built around the "tcpClient" Max object. Note the difference in capitalization to distinguish the patcher from the object.

`TCPClient` handles all TCP interactions on the client side and the patch is essentially a wrapper for various functionalities of the tcpClient object. The two arguments to the tcpClient object specify the IP address and port to connect to. The tcpClient object is set to a loopback (IP 127.0.0.1) on port 9999 by default. Output of the object is routed according to whether it receives a data (reply), status, or 'self' message. The `route` object is used extensively to filter messages based on their content. The "`s 2client`" object is used to send messages to the tcpClient object for the sake of visual clarity in the patch. A function was provided for setting the user's IP address behind a firewall/router which does network address translation (as the internal IP address of their computer will be different than the external IP on the internet). This function is currently disabled. The functions which are enabled are: set TCP port, set Server IP address, get own IP address, connect, disconnect, negotiate, verbose on/off, and dialog - to send short text messages to the server.

The `UDPClient` patch primarily handles the UDP connection to the server and receives real-time data from the server over that connection. It is based around the otudp and OpenSoundControl objects. The "otudp" object reads raw UDP packets from the port specified as its first argument. The number of available buffers is indicated by 'nbufs' (2048) and the buffer size is indicated by 'bufsize' (512). The UDP port is set by default to be port 7006 but can be changed if desired.

UDP packets are sent to the OpenSoundControl object which reformats the data which is then sent to route to effectively de-multiplex the packets and strip off the routing information (tokens `a-f`). The data is then sent out 8 outlets

corresponding to the 8 channels of brainwave data. The raw numbers are also then packed together again into a list which is sent to the "scope" patch to provide a visual display of the data.

Appropriate messages can be sent to `otudp` to turn off error reporting, get status or change the UDP receive port. The `OpenSoundControl` object is merely translating the UDP packets into a Max-usable form.

The `FuzzyLogicMatrixRoute` patch is a misnomer which reflects my wishes for what I initially intended it to be and what I might replace it with in the future rather than what it is right now. In actuality, this is really just 8 discrete configurable lookup tables (LUT) which allow the user to set up 'mappings' of input to output data. This mapping is programmable and repeatable via a 'save' function.

Because the input signals are floating-point and the `table` object currently only understands integers, I convert the floats first to int, perform the mapping and then convert them back into floats. This imprecise procedure itself introduces another "fuzzy" distortion into the signal. The conversion is performed thus:

1. Multiply the float by 256
2. Convert to int.
3. Use this scaled integer to do an inverse lookup on a 256 by 256 table.
4. A predetermined set of tables are read at launch time (tab1-tab8). These can be edited by the user.
5. Lookup values are then scaled back by the inverse of 256 (1/256 = 0.003906) which makes them into floating point numbers again.

The `HarmonicGenerator` patch manages allocation of sound generating patches according to data which is passed to it. It takes 8 continuously varying floating point numbers as input and generates sound from that data. As the title implies, the generated sound is based upon the Harmonic Series.

The 8 floating point inputs pass through the `MultiplyMatrix` patch which is an attempt to generate 24 continuously varying floating point values which are nonetheless all interrelated in complex ways. The actual implementation of this patch could doubtless be improved. The output of the multiply matrix is then passed through an `abs` object which ensures that the values remain positive. The resulting 24 floating point values are then used to provide values for the phase, pan and amplitude envelopes of the tone generator modules. These values are passed to the tone generators via a `sender` object which is used primarily for tidiness and encapsulation. The sender object takes an argument which corresponds with the argument of a tone generator inside the ToneGenBank patcher. The `ToneGenBank` (Tone Generator Bank) patcher contains 25 distinct `toneGen` modules which are all identical save for their two passed arguments. The first argument as mentioned corresponds with a sender patch, the second sets the initial harmonic number as a ratio to the current fundamental frequency. Thus if the second argument to `toneGen` is 20., it will generate a sine tone 20 times that of the fundamental. The audio outputs of each individual `toneGen` are summed by the `receive~` objects.

The `toneGen` patch takes care of receiving synthesis parameters as input (partial, phase, envelope and pan), synthesizing the sound according to those parameters, and then routing it to an output as desired.

The synthesis parameters are received from the "sender" patch and then routed according to the parameter being controlled. The actual sound synthesis is done by the `oscillate~` abstraction and panning is done by the `panner` abstraction. Provision is made for sending the audio output to either a panned stereo output or a multichannel matrix mixer (not yet implemented).

The `oscillate~` abstraction does simple sine tone synthesis according to given parameters. It is built as a wrapper for the `cycle~` synthesis object.

The first inlet sets the fundamental frequency which can then be multiplied by any number (set via the second inlet). The third inlet sets the relative phase and the fourth sets the amplitude which is somewhat smoothed by the use of a `line` object.

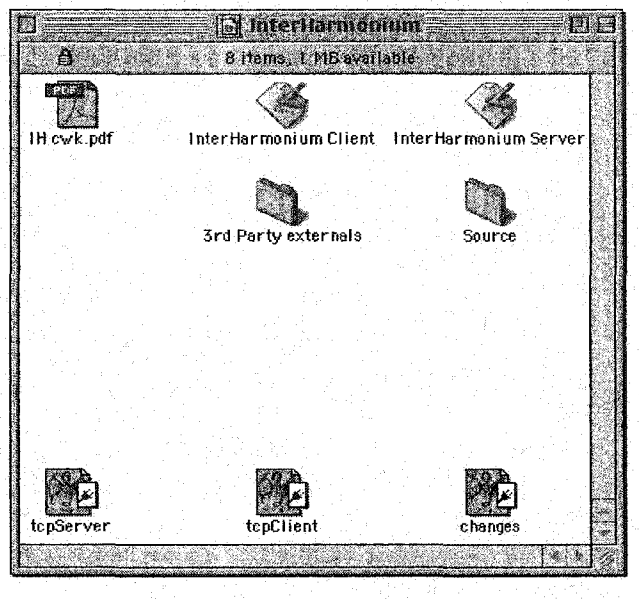
The `panner` abstraction takes an audio input signal and a floating point number (-1.0 to 1.0) and pans the audio between the two outputs according to the following schema: -1 = hard left; 0 = centre; +1 = hard right. To produce a more realistic effect an approximate equal power panning formula is used based upon the square-root of the pan value.

project : directions for use

Note: this software will only run on a recent Power Macintosh

(G3 or newer , MacOS 9.04 or newer) with a network connection.

1. If you receive the software on CD, it is recommended to copy the entire contents of the CD to a folder on your hard drive. This can be effected by dragging the CD icon from the desktop into the place you want to install it.



2. The Max external objects "tcpServer", "tcpClient", and "changes" must reside in the same folder as the InterHarmonium applications.

3. The InterHarmonium Server and Client applications are designed to work on two separate physical machines connected via an IP network but they may also run on the same machine in 'loopback' mode. (This significantly increases the resource load on this machine and at least a G3/300 MHz Power Mac is

recommended.)

4. OMS (Open Music System from Opcode) is neither necessary nor recommended.

5. Launch the InterHarmonium Server on the server machine first then launch the InterHarmonium Client on the client machine.

6. The Server should display some status messages as it launches and then finally display "listening: 9999" in the Status window.

7. If you have OMS and you are running in loopback mode, you will get an OMS error message. Click "quit" when this message comes up.

8. There is ordinarily no reason to make any changes to the default configuration of the server when it launches - all configurations can be changed from the client.

9. To run in 'loopback' mode with both Server and Client on the same machine, on the Client software:

- select 'Set Loopback Address' in the 'NetworkControl' menu.
(This should in fact already be selected by default.)

10. To run the system with sound output:

- select 'Connect' in the 'NetworkControl' menu.

- select 'Negotiate' in the 'NetworkControl' menu.

- select 'Audio On' in the 'HarmonicControl' menu, this will start the unmodulated harmonic generator

Then in the Server application:

- either connect the live brainwaves to an A->D device or select 'SimBrain On' in the 'BrainControl' menu, this will begin sending the brainwave control signals to the client and you should immediately hear modulation

- you can monitor the brain waveforms by selecting 'Scope Display' in the 'BrainControl' (Server) or 'HarmonicControl' (Client) menus.

11. Keyboard Shortcuts

(press the given key on the computer keyboard to actuate the corresponding shortcut - most apply to both the server and the client)

d - open dialog box to send short text message to the remote machine
m - output to multichannel mixing matrix (to be implemented at a future time)
o - turn sound output off
s - output to stereo left/right channels
z - toggle real/simulated brain signals

~ (tilde) - toggle audio on/off at the local machine

12. Menu functionality:

Server:

NetworkControl - adjusts network settings

Listen On

Listen Off

Set Listen Port

Disconnect

Verbose Mode

otudp Status

TCP Status

BrainControl - adjust settings related to brainwave processing

Audio On

DSP Status Window

SimBrain On

Scope Display

Patchers - opens key patchers

TCPServer

UDPServer

Brain

Client:

NetworkControl - adjusts network settings

- Set Remote IP Address
- Set Loopback Address
- Set TCP Port
- Set UDP Port
- Connect
- Disconnect
- Dialog
- Verbose Mode
- otudp Status
- TCP Status

HarmonicControl - adjust settings related to brainwaves->sound mapping

- Audio On
- DSP Status Window
- Synthesize
- Scope Display

Patchers - opens key patchers

- TCPClient
- UDPClient
- FuzzyLogic
- MatrixRoute
- HarmonicGenerator

conclusion and future directions

The two great challenges in realizing this project have been technical and artistic: how to make it work technically and have it resonate artistically. Though I have tackled both, neither has completely yielded the results I had wished for at the outset. This project has been neither a complete failure nor a resounding success. It has been a first tentative step in an exciting and not well-trodden direction. The software, as written, does what it was conceived to do; it functions more-or-less properly. When I recently performed using this software at "Bruits du Noir", I ended up significantly rewriting large sections of the audio synthesis function. I now feel I would prefer to rewrite the entire software package itself. There are many possibilities for improvements in the elegance, robustness and flexibility of this software.

What has become increasingly clear to me - as long as I create software like this to make music - is that the software must evolve to meet the needs of the music as the music itself evolves. Neither the music nor the tools upon which it relies can evolve in a vacuum; they are interdependent. The purpose of this exercise, then, has been to discover not just what kind of music I want to make and not just how to create the tools to make it but how the two thought processes - artistic and technological - interact.

As this work has progressed I increasingly get the feeling that it is just starting; in an extremely disquieting way, it has raised far more questions for me than it has answered. I find this at once frustrating yet also stimulating. All I can postulate is that it will continue.

glossary

ADB	Apple Desktop Bus
ADPCM	Adaptive Differential Pulse Code Modulation
ADSL	Asymmetrical Digital Subscriber Line
APCM	Adaptive Pulse Code Modulation
API	Application Programmer Interface
ASIO	Audio Streaming Input/Output (Steinberg)
ATM	Asynchronous Transfer Mode
AUDP	Audio UDP
Broadcast	IP broadcast to all hosts on the network
BSD	Berkeley Software Distribution
broadcast	traditional notion of media broadcast (Radio, TV)
CAST	CNMAT Additive Synthesis Tools
CCRMA	Center for Computer Research in Music and Acoustics
CELP	Code Excited Linear Prediction
CNMAT	Center for New Music and Audio Technologies
CODEC	COder/DECoder
CPU	Central Processing Unit
D1	professional digital uncompressed video standard
DCT	Discrete Cosine Transform
DSP	Digital Signal Processing
EEG	Electroencephalogram (graph)
EKG	Electrocardiogram
EMG	Electromyogram
EOG	Electrooculogram

FDDI	Fiber Distributed Data Interface
FFT	Fast Fourier Transform
Firewire	IEEE 1394 bus standard also known as iLink
GNU	GNU's Not Unix (Unix-like operating system)
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IP/TV	Internet Protocol Television (Cisco)
Ipv4	Internet Protocol version 4
Ipv6	Internet Protocol version 6
ISDN	Integrated Services Distribution Network
ITU	International Telecommunications Union
LAN	Local Area Network
LPC	Linear Predictive Coding
MacOS	Macintosh Operating System
Max/MSP	music software development environment
Mbone	virtual Multicast Backbone
MIDI	Musical Instrument Digital Interface
MNI	Montreal Neurological Institute
MODEM	ModulatorDEModulator
MOTU	Mark of the Unicorn
Multicast	IP multicast to group
mLAN	IEC 61883-6 protocol for Digital Audio over Firewire
NAT	Network Address Translation
OC-3	Optical Carrier Level 3 (155.52 Mbps)
OMS	Open Music System (Opcode)

OO(P)	Object-Oriented (Programming)
OSC	Open Sound Control
OSI	Open Systems Interconnection
OT	OpenTransport (streams-based MacOS IP networking stack)
otudp	Open Transport UDP object for Max
PCM	Pulse Code Modulation
POTS	Plain Old Telephone System
QoS	Quality of Service
RFC	Request for Comments
RISQ	Reseau d'Information Scientifique du Québec
RSVP	Resource Reservation Protocol
RTCP	Real Time Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SCSI	Small Computer System Interconnect
SDP	Session Description Protocol
SGI	Silicon Graphics Incorporated
SMIL	Synchronized Multimedia Interaction Language
SONET	Synchronous Optical Network
T1	1.544 Mbps digital transmission line
T3	44.736 Mbps digital transmission line
TCP	Transmission Control Protocol
TTL	Time-To-Live
UART	Universal Asynchronous Receiver/Transmitter
UCL	Undergraduate Computer Lab (McGill University)
UDP	User Datagram Protocol

Unicast	IP unicast to one other host on the network
W3C	World Wide Web Consortium
WAN	Wide Area Network
WWW	World Wide Web
ZIPI	proposed replacement for MIDI

appendix a : music for solo performer

MUSIC FOR SOLO PERFORMER (1965)[29]
for enormously amplified brain waves and percussion

The alpha rhythm of the brain has a range of from 8 to 12 Hz., and, if amplified enormously and channeled through an appropriate transducer, can be made audible. It can be blocked by visual attention with the eyes open or mental activity with the eyes closed. No part of the motor system is involved in any way. Control of the alpha consists simply of alteration of thought content - for example, a shifting back and forth from a state of visual imagery to one of relaxed resting.

Place an EEG scalp electrode on each hemisphere of the occipital, frontal, or other appropriate region of the performer's head. Attach a reference electrode to an ear, finger, or other location suitable for cutting down electrical noise. Route the signal through an appropriate amplifier and mixer to any number of amplifiers and loudspeakers directly coupled to percussion instruments, including large gongs, cymbals, tympani, metal ashcans, cardboard boxes, bass and snare drums (small loudspeakers face down on them), and to switches, sensitive to alpha, which activate one or more tape recorders upon which are stored prerecorded, sped-up alpha.

Set free and block alpha in bursts and phrases of any length, the sounds of which, as they emanate from the loudspeakers, cause the

[29] Lucier. p.69

percussion instruments to vibrate sympathetically. An assistant may channel the signal to any or all of the loudspeakers in any combination at any volume, and, from time to time, engage the switches to the tape recorders. Performances may be of any length.

Experiment with electrodes on other parts of the head in an attempt to pick up other waves of different frequencies and to create stereo effects.

Use alpha to activate radios, television sets, lights, alarms, and other audiovisual devices.

Design automated systems, with or without coded relays, with which the performer may perform the piece without the aid of an assistant.

Edmond Dewan, Technical Consultant

appendix b : 3rd Party Externals Used

Norm Jaffe

<http://www.opendragon.com/Pages/MaxObjects.shtml>

David Rokeby

<http://www.interlog.com/~drokeby>

<http://node.net/cgi-bin/ftpex/FTPex.cgi?get&IRCAM_Pub_Nov.99

/FAT/utilities/Rokeby%20Objects.sit>

CNMAT (Matt Wright)

<http://www.cnmat.berkeley.edu/osc>

Ted Apel, John Puterbaugh, and David Zicarelli.

<http://www-crca.ucsd.edu/~tapel/software.html>

appendix c : IETF Proposals and Requests for Comment

Host Extensions for IP Multicasting
<ftp://ftp.isi.edu/in-notes/rfc1112.txt>

SDP: Session Description Protocol
<ftp://ftp.isi.edu/in-notes/rfc2327.txt>

Real Time Streaming Protocol (RTSP)
<ftp://ftp.isi.edu/in-notes/rfc2326.txt>

RTP: A Transport Protocol for Real-Time Applications
<ftp://ftp.isi.edu/in-notes/rfc1889.txt>

RTP Profile for Audio and Video Conferences with Minimal Control
<ftp://ftp.isi.edu/in-notes/rfc1890.txt>

RTP Payload for Redundant Audio Data
<ftp://ftp.isi.edu/in-notes/rfc2198.txt>

Audio/Video Transport Working Group
<http://www.ietf.org/html.charters/avt-charter.html>

bibliography

Abelson, Harold and Gerald Jay Sussman with Julie Sussman. *Structure and Interpretation of Computer Programs.* Cambridge, Mass: MIT Press, 1996.

Apple Computer Inc. *Quicktime for the Web: A Hands-On Guide.* San Francisco: Morgan Kaufmann, 2000

Bargar R., Church S., Fukuda A., Grunke J., Keislar D., Moses B., Novak B., Pennycook B., Settel Z., Strawn J., Wiser P., and Woszczyk W. "Internet Capabilities", White Paper, *Journal of the Audio Engineering Society*, Vol. 47, No 4, April 1999: pp. 300-310.

Barthes, Roland. *Camera Lucida.* New York: Noonday, 1981.

Beggs, Josh and Dylan Thede. *Designing Web Audio.* Sebastopol: O'reilly, 2001.

Benade, Arthur H. *Fundamentals of Musical Acoustics,* 2nd ed. New York: Dover, 1990.

Benjamin, Walter. *Illuminations.* New York: Schocken, 1969.

Bosanquet, R.H.M. *An Elementary Treatise on Musical Intervals and Temperament* (London 1876). Utrecht: Diapason Press, 1987.

Cage, John. *Silence.* Middletown, Conn: Wesleyan University Press, 1961.

Crowcroft, Jon, Mark Handley and Ian Wakeman. *Internetworking Multimedia.* San Francisco: Morgan Kaufmann, 1999.

Donahoo, Michael and Kenneth Calvert. *The Pocket Guide to TCP/IP Sockets : C Version.* San Diego: Academic Press, 2001.

Deering, Steve. "Multicast Routing in a Datagram Network." Palo Alto: Stanford University, 1991. (Ph.D. thesis) - see "Host Extensions for IP Multicasting" RFC 1112, 1988.

Eaton, Manford. *Bio-Music: Biological Feedback Experiential Music Systems.* Kansas City: Orcus, 1971.

Fisch, Bruce J. *Fisch and Spehlmann's EEG primer : basic principles of digital and analog EEG.* Amsterdam: Elsevier, 1999.

Gibran, Kahlil. *The Prophet.* New York: Alfred A. Knopf, 1981.

Goldberg, RoseLee. *Performance: Live Art 1909 to the Present.* New York: Abrams, 1979.

Gozza, Paolo. *Number to Sound: The Musical Way to the Scientific Revolution.* Dordrecht: Kluwer Academic Publishers, 2000.

Hacker, Scot. *MP3: The Definitive Guide.* Sebastopol: O'Reilly, 2000.

Helmholtz, Hermann. *On the Sensations of Tone as a Physiological Basis for the Theory of Music.* New York: Dover, 1954.

Holtzman, Steven R. *Digital Mantras : The Languages of Abstract and Virtual Worlds.* Cambridge, Mass: MIT Press, 1996.

Jacobs, D. "MuscleMusic: experience using BodySynth." IRCAM, 1992.

Kelly, Kevin ed. *Signal: Communication Tools for the Information Age.* New York: Harmony, 1988.

Knapp and Lustad. "Bioelectric controller for computer music". *Computer Music Journal,* 14(1) pp. 42-47. 1990.

Kumar, Vinay. *Mbone: Interactive Multimedia on the Internet.* Indianapolis: New Riders, 1996.

Lu, Guojun. *Communication and Computing for Distributed Multimedia Systems.* Boston: Artech House, 1996.

Lucier, Alvin and Douglas Simon. *Chambers.* Middletown: Wesleyan University Press, 1980.

Mambretti, Joel and Andrew Schmidt. *Next Generation Internet.* Toronto: Wiley, 1999.

Maremaa, Tom and William Stewart. *Quicktime for Java: A Developer Reference.* San Francisco: Morgan Kaufmann, 1999 .

McLaughlin, Elbert H. "The coalescence of music and the internet : a hybrid solution for the use of music materials in worldwide web publication." Montreal: McGill University, Faculty of Music. (MA Thesis), 1996.

Minsky, Marvin. *The society of mind.* New York: Simon and Schuster, 1986.

Moorer, James. "Audio in the New Millennium." *Journal of the Audio Engineering Society*, Vol. 48, No 5, May 2000: pp.490-498.

Naughton, John. *A Brief History of the Future: The Origins of the Internet.* London: Phœnix, 2000.

Nuñez, Paul. *Electric Fields of the Brain: The Neurophysics of EEG.* Oxford: Oxford University Press, 1981.

Nyman, Michael. *Experimental Music.* New York: Schirmer, 1974.

Peckham, Morse. *Man's Rage for Chaos: Biology, Behavior and the Arts.* New York: Schocken, 1976.

Pope, Stephen Travis. *The Well-Tempered Object : Musical Applications of Object Oriented Software Technology.* Cambridge, Mass: MIT Press, 1991.

Qdesign Corporation. *Qdesign Music Codec 2 Manual.* Vancouver: Qdesign Corporation, 1999.

Raghavan, S.V. and Satish K. Tripathi. *Networked Multimedia Systems, Concepts, Architecture, and Design.* New Jersey: Prentice-Hall, 1998.

Roads, Curtis. *The Computer Music Tutorial.* Cambridge, Mass: MIT Press, 1996.

Rosenboom, David. *Biofeedback and the Arts: Results of Early Experiments.* Vancouver: Aesthetic Research Centre of Canada, 1976.

Schultzrinne, Henning and Jonathan Rosenberg. *Internet Telephony.* 2000. (Unpublished)

Schultzrinne, Henning. "Internet Media-on-Demand: The Real-Time Streaming Protocol." New York: Columbia University, Dept. of Computer Science, 1999. (presentation)

Schwartz, Elliot. *Electronic Music : A Listener's Guide.* New York: Praeger, 1975.

Selfridge-Field, Eleanor. *Beyond MIDI: The Handbook of Musical Codes.* Cambridge, Mass: MIT Press, 1997.

Sorenson Vision Inc. *Sorenson Broadcaster - User Guide.* Logan, Utah: Sorenson Vision Inc., 1999.

Stern, Judith and Robert Lettieri. *Quicktime Pro for Macintosh and Windows.*

Berkeley: Peachpit Press, 1999.

Stevens, W. Richard. *TCP/IP Illustrated.* Reading, Mass: Addison-Wesley, 1994.

Tenney, James. "Harmonium #3." Baltimore: Sonic Art Editions, 1990.

Tomkins, Peter and Christopher Bird. *The Secret Life of Plants.* New York: Harper & Row, 1973.

Wanderley, Marcelo and Marc Battier. *Trends in Gestural Control of Music.* (CD) Paris: IRCAM, 2000.

Weidenaar, Reynold. *Magic Music from the Telharmonium.* London: Scarecrow Press, 1995.

Wiener, Norbert. *The Human Use of Human Beings: Cybernetics and Society.* New York: Discus, 1967.

Williamson, Beau. *Developing IP Multicast Networks - Volume 1.* Indianapolis: Cisco Press, 2000.

Zatorre, R.J. (1988). "Pitch perception of complex tones and human temporal-lobe function." *Journal of the Acoustical Society of America,* 84, pp. 566-572.

Zuckerkandl, Victor. *Sound and Symbol: Music and the External World.* Princeton: Princeton University Press, 1973.