Contents

The Otherside
Web-based Collaborative Sound Synthesis System

Ilias Anagnostopoulos

September 5, 2008

Abstract

The Otherside [1] is an advanced open-source multimedia system, designed to run as a server application. It is essentially a sound synthesis system that allows for the creative collaboration of a number of people or machines over computer networks. Unlike most similar attempts to create network-based audio applications, the Otherside does not require any specialized audio software or massive amounts of computer processing power. The audio processing is done on the server side and directed to listeners using a simple Internet Radio protocol[2]. Any listener can then choose to participate in the sound creation process by using a simple web-browser-based chatroom where they can type in commands to control parameters of the sound or chat with other users of the Otherside, thus giving more of a community feel to the process. It also allows more advanced users to utilise their own advanced systems for network control by following the simple steps on connecting custom devices and software to the Otherside using well-known protocols such as OSC [3], IRC [4], HTTP [5] and raw network connections.

1 Introduction

1.1 Creative Collaboration

The Otherside Server provides a platform for the creative collaboration of any number of people, regardless of their understanding of music, computer programming or networking. It is platform-independent, meaning that it will run properly on any operating system and any computer architecture. This makes it easily accessible to a large number of people who can work together and even communicate, during the course of their session with Otherside.

[1] The Otherside Server[Anagnostopoulos, 2008a]
[2] Streaming Audio
[3] Open Sound Control
[4] Internet Relay Chat
[5] Hyper-Text Transfer Protocol

People can create music together, from randomly changing values and creating random sonic events to carefully crafting their own software that will interface with the Otherside like an additional musical instrument. It is an entrance to the world of computer music for people who have no such previous experience. It can also serve as an advanced experimentation platform for users who are already familiar with the protocols and principles involved. It is open and accessible to several different protocols and adheres to the open-source mentality[6], meaning that users can get involved to the extent of becoming developers of the Otherside system.

1.1.1 Computer Network Collaboration

Computer Networks are nowadays dominating the world through the internet. However, most of the well-established network applications are very simple and are not particularly demanding in processing power and network bandwidth. Digital Signal Processing is a complicated task that only recently became a "household" application on personal computers. Careful planning and consideration is required to overcome some of the difficulties posed by computer networks and their use by artistic collaboration systems.

One of these problems is that network connections and especially the internet often involve extremely long distances between two users, thus inevitably data can not be transferred seamlessly. There is an evident latency issue that is being treated differently by such project developers. Jeorg Stelkens discusses this in a paper about his own network collaboration system[7], proposing the integration of latency in the creative process, as a unique characteristic of the computer network as a medium. The Otherside does not attempt to overcome or to cover-up latency, unlike most projects mentioned by Stelkens. It merely accepts latency as a fact and uses the amount of users simultaneously transmitting data to create a sonic situation where events can either be left to chance, creating aleatory music[8], or forcing a user to take the amount of latency into consideration when crafting a sonic event in order to be able to predict the outcome.

1.2 Related Creative Development

1.2.1 NetPD

An inspiring project, leading to the initial thoughts for creating the Otherside was NetPD[9]. NetPD is based on Pure-Data[10], but acts as a meeting point rather than a sound generation engine. It allows users of Pure-Data to collaborate by sharing control data online, while each one creates sound individually in any

[6] Open-Source Software make their source code available to the public, encouraging people to study it and learn how it works

[7] peerSynth[Stelkens, 2003]

[8] Also know as chance music[Encyclopaedia Britannica, 2008]

[9] NetPD, Pure-Data based networking collaboration project[Haefeli, 2008].

[10] Pure-Data, data-flow programming language[Puckette et al., 2008].

way they want. The idea of collaboration is common between NetPD and the Otherside, but it seemed like an unnecessary limit, to keep it only between users of Pure-Data, who would have to be comfortable with data-flow programming and networking. The Otherside attempts to break this boundary by giving the chance to users to become involved as much as they please. NetPD also features a chat facility within Pure-Data, for communication between participants. Otherside is based on IRC, a well-known chat interface, but goes much further than just using it for communication. It is worth noting that it is technically possible to connect NetPD and the Otherside and create a collaboration between the two projects.

1.2.2 Pure-Data and Data-Flow Programming

Pure-Data was created by Miller Puckette, as an open-source project, similar to his Max[11] program. It is a data-flow programming language geared towards audio/visual output and control. Pure-Data is a direct descendant of Max, written in 1988[12] by Miller Puckette while he was in IRCAM[13]. David Zicarelli together with Puckette later wrote MSP as an addition to Max, thus creating Max/MSP, enabling audio-rate digital singal processing. Max was only capable of control rate processing, as computers in 1988 were not fast enough to handle audio-rate signal processing. Nowadays data-flow programming is used widely for audio applications, as it is a very powerful and versatile way of interfacing with various control protocols. Computers nowadays have enough processing power to support heavy digital signal processing applications and networking speeds are high enough to be used creatively in interactive applications. This makes data-flow programming languages an interesting option for users who want to explore the field of complex control in sound generation systems.

1.2.3 Linux

The Linux operating system was developed by Linus Torvalds in the early 90's and is a very good example of the open-source mentality and free software. It was based on Unix, an older computer operating system that was extremely stable and advanced. Linux is now being actively developed by the open-source community, which amounts to millions of developers from around the world. It is now trusted by some as being superior to other operating systems, especially for server applications and embedded systems. A Linux operating system[14] is the backbone of the Otherside, allowing all the computer software to interface properly and serve the appropriate data through a network.

[11] Max, MSP, Jitter, by Miller Puckette [Puckette, 2008].

[12] According to Max Matthews in the foreword section of Miller Puckette's book[Matthews, 2007]

[13] Institut de Recherche et Coordination Acoustique/Musique, Paris, France

[14] Ubuntu Linux[Canonical, Inc., 2008]

2 Technical Description

2.1 Operating System

The Otherside is based on open-source principles, being built on the Ubuntu Linux 8.04 Server Edition[15] operating system. The operating system was chosen as it has proved to be very stable as a server system while being directly compatible with all the audio software[16] and techniques that were to be used. It is also an operating system that is very open to customization, something that was necessary to ensure high performance under the load of audio processing and server applications.

The preparation of the computer system on which the Otherside was tested started with the installation of the base Ubuntu 8.04 Server system, replacing the Linux Server Kernel with the Linux 2.6.24 RT[17] Kernel, which allows audio processing to be done with Real-Time priority. Since audio processing is one of the main functions of the Otherside, real-time implementation was an important step towards high performance. To enable the audio to function on the Ubuntu Server operating system the installation of ALSA [18] was also required.

2.2 Web interface

To achieve an extreme level of accessibility by users who have no interest in turning their personal computer into an audio workstation, all functions of the Otherside are accessible from a simple website interface.

The starting point is a website that is powered by the Apache 2 HTTP server[19], the most widely used HTTP server on the internet. This links to a Streaming Audio Server[20], an IRC Chat Interface, a help document simply explaining the basics of the Otherside to the users and an email link for communication with the administrator of the system. Apache 2 was chosen due to the fact that all other HTTP servers under investigation[21] were simply not as advanced and reliable, something which is also evident by the widespread use of the Apache on the internet.

The Streaming Audio Server is using the Icecast 2 Server[22] to provide a compressed audio stream produced by the Otherside to an audience with an internet connection and a media player[23] on their computer. The Icecast 2 Server

[15] A very stable operating system geared towards Server applications

[16] The Linux sound devices, although not installed by default on Ubuntu Server systems, are available from the supported repositories.

[17] An alternative Kernel, with Real-Time capabilities

[18] Advanced Linux Sound Architecture[Kysela et al., 2008]

[19] Apache 2 is a product of the Apache Software Foundation[The Apache Software Foundation, 2008]

[20] Internet Radio

[21] Examples of other HTTP Servers that were considered instead of Apache include Mathopd, micro-httpd, mini-httpd and nanoweb

[22] A product of the XIPH Open Source Community[The XIPH Open Source Community, 2008]

[23] Any software program that can reproduce digital media files

receives a compressed audio stream, utilising MPEG Layer 3[24] compression, forwarding it to a number of listeners over a computer network. This is based on the established Internet Radio Station practice of using an encoder that streams audio to a Streaming Audio Server to be forwarded to any number of listeners. The difference here is that instead of encoding sound files, the encoder is directly encoding and streaming the computer-generated sound of the synthesis engine that powers the Otherside. The link from the main website redirects a user to the website interface of the Icecast 2 Server, which contains details about the audio stream and a link for listening. Icecast 2 was chosen because it is a very well-made open-source Streaming Server with a friendly user interface.

The control interface was a challenge as there were too many functions that needed to be implemented in a simple and accessible interface. The choice of an IRC Chat interface as the control interface was made as the nature of IRC already implemented a lot of the functions that were needed, without the need of reinventing the wheel. Several different alternatives for the IRC Server software were tried, but most of the common servers[25] proved to be unsuitable for the kind of use that was about to be adopted. The IRC protocol[26] includes several functions geared towards huge IRC Server networks, which were a burden on CPU power and configuration time, therefore clearly not wanted in the Otherside. The server software which was finally used was the IRC-Net IRC 2 Server, as it was very simple to configure, reliable and somewhat modular in design, allowing the administrator to engage or disengage any modules needed for the task. By default, it comes with the bare minimum necessary to function, which is exactly what the Otherside requires.

Usually, an IRC Network has ways of registering nick-names[27] and channels/chat-rooms[28]. This is to allow frequent users to always use their own nickname while prohibiting anybody else from using it, while there is also a service for leaving messages to registered users that are currently offline[29]. These are collectively known as IRC Services. The decision against using any IRC Services Provider package came naturally at this stage, as the IRC Server used with the Otherside is merely a control interface, therefore there is no need for registering a nick-name or a chat-room.

An IRC Server also has the ability to connect to other IRC Servers, forming IRC Networks. This was one of the most impressive features of the IRC protocol, as it means that there is room for a great amount of evolution for the Otherside, by connecting to other Othersides and forming a huge Otherside Network in a very simple and efficient way. This is also an easy way to get extensive load off of one server's central processing unit, by interfacing several computers to run the Otherside System and sharing the load of processing in a clustered-computing

[24] Commonly refered to as mp3 due to the ".mp3" extension after the filename on such files

[25] Servers that were considered include the Undernet IRC Daemon, the Ratbox IRC Daemon and the Hybrid IRC Daemon

[26] The IRC protocol refers to the specifications discussed in RFC1459[Oikarinen and Reed, 1993] and its successor, RFC2813[Kalt, 2000]

[27] Known as NICKSERV

[28] Known as CHANSERV

[29] Known as MEMOSERV

fashion.

However, an IRC Server is only the back-end, requiring a Client program[30] for user interaction. Most Client programs assume a certain level of understanding from the user about the IRC Protocol. Investigation on this matter led to the use of CGI:IRC, a very smart way of creating a web-browser interface for the IRC Server, thus eliminating the requirement for an IRC Client program to be installed on the client computer. CGI:IRC runs on the server machine and creates a CGI-based web-page that can run on the majority of web-browsers, allowing the user to connect to and use the IRC Server from this web-based client.

This IRC interface allows a number of users to connect to a chat-room and type in messages. The interaction between the users and the server was achieved by using an IRC Infobot, a computer program that sits on an IRC Server, posing as a human user, that replies to certain messages with predefined actions. These messages usually contain information regarding the Server. This idea let to the creation of an Infobot-like program that would listen for specific messages, but instead of replying back to the user that sent the message, it would respond by directing them to the Sound Synthesis Engine. After some investigation, it became evident that there was nothing similar readily available. Thus came the decision to create an original control-bot using a computer programming language. I investigated C/C++ and Python[31], deciding against C/C++ as it was lacking some of the easy-to-use functions of Python for string formatting[32]. By studying some existing Infobots[33] I was able to see how they worked and obtain ideas for implementing in my own code. The source code was loosely based on the XoR bot skeleton as it had some interesting string formatting examples. I ended up altering almost all the functions to make them more suited to the specific needs of the project. The SimpleOSC Library[34] for the Python programming language enabled the reformatting and redirection of messages to the Sound Synthesis Engine, using the extremely versatile OSC protocol[35]. The program also uses the Sockets Library for TCP/UDP network connections to connect to the IRC Server and to redirect all undefined messages sent by users to a module on the Sound Synthesis Engine that treats it as raw ASCII[36] data. Finally, some of the existing Infobot funtions of the XoRbot were retained almost intact, especially regarding the IRC Protocol channel functions[37] and also to

[30] Common IRC client programs include mIRC for the Windows operating system and bitchX for POSIX systems

[31] C/C++[Ritchie, 1993] and Python[Python Software Foundation, 2008] are widely used low-level programming languages

[32] In computer programming, string formatting refers to the formatting of input text with the aim of using either only a part of it, or bringing it to a desireable format for subsequent use

[33] The XoR bot skeleton[ArchGh0ul, 2007] and the Redland Bot[Beckett, 2008]

[34] OSC Library for Python[Holth et al., 2008]

[35] Open Sound Control[Wright, 2002]

[36] American Standard Code for Information Interchange

[37] Although these are not needed at present, they may be necessary if there are any new IRC Server-side modules activated

include a short help message and the basis for future MIDI[38] implementation.

The users that log on to the chat-room see a user with the nick-name âĂIJOtherbotâĂİ, which is the bot that interfaces the IRC Server with the Sound Synthesis Engine. The only thing they need to do to control the Sound Synthesis Engine is send messages to the bot as if they were chatting to a friend, from their web-browser window.

2.3 Sound Synthesis Engine

The Sound Synthesis Engine is based on a version of the Pure-Data data-flow programming software, called PD-extended. It is modular in design and consists of a Polyphonic Additive Synthesis[39] Module, a Wavetable Synthesis[40] Module, a Single Sideband Modulation[41] Module, a Ring Modulation[42] Module, a Comb Filter[43] Module, a Stereo Delay Line[44] Module and a Speech Synthesis Module. The Speech Synthesis Module is not based on PD-extended. Instead, it is based on the Festival Speech Synthesis System[45], developed by the Centre for Speech Technology Research of the University of Edinburgh.

The Otherbot outputs three kinds of messages towards the Speech Synthesis Engine. The easiest to explain is probably the Raw ASCII Data. When a user types a message in the chatroom that is not recognised as a predefined command, the bot establishes a UDP[46] connection to port 9998 of the server and redirects the full message. UDP Port 9998 is opened by a PD-extended object listening for inbound connections, thus acting as a daemon program. The data is received as ASCII integer values. These are treated as MIDI pitch values and are directed to a polyphonic additive synthesis module, which creates the equivalent of each MIDI message received, spatially panning it according to the number of characters received in a message. The odd numbers are panned to the left while the even ones are panned to the right. Therefore a message length of three characters would have the first and third characters panned to the left while the second character is panned to the right, after being converted from characters to ASCII values representing MIDI pitch values and then synthesized to sound. The sound is then directed to a Stereo Delay Module for further processing. The spatialization in this module is evidently very simple, something which will be rectified in future versions, by replacing the current algorithm with one that does proportional panning. It currently acts as a mere demonstration of the potential of the Otherside.

[38] Musical Instrument Digital Interface, presently not implemented, but it is part of the plans for further development and compatibility

[39] Sound synthesis technique[Reid, 2000]

[40] Sound Synthesis Technique[Bristow-Johnson, 2008]

[41] Signal Processing technique, following the Hilbert Transform mathematical module[MathWorks Team, 2008]

[42] Signal Processing Technique[Lehman, 2007]

[43] Subtractive Synthesis Technique[Smith III, 2007a]

[44] Time-based Signal Processing Technique[Smith III, 2007b]

[45] Speech Synthesis software, developed at the University of Edinburgh, UK[Black et al., 1999]

[46] User Datagram Protocol

When a message typed in the chatroom matches a predefined command recognised by the Otherbot, this is redirected as OSC data by establishing a UDP connection to port 9997 of the server. UDP Port 9997 is opened by a PD-extended object designed to receive OSC data and direct it to the equivalent OSC path. These OSC paths direct data to the appropriate objects in the PD-extended patch, where they control certain functions, such as the speed at which the wavetable is read or the delay time and feedback.

The Wavetable Synthesis Module consists of two saved wavetables. One is used to define a waveform while the other is used to define the speed at which the waveform is being reproduced. When the speed is altered using the appropriate OSC command, the waveforms are being read at different speeds, so the whole process can be sped up or slowed down according to taste.

The outputs of the Wavetable Synthesis Module are directed to two Single Sideband Modulators. The first phase of the Single Sideband Modulation module directs the signal through a "hilbert" object, which is basically two 4th order filters whose output is 90 degrees out of phase, approximating the mathematical Hilbert Transform[47]. The two out-of-phase outputs are then directed to two signal multipliers that multiply a sawtooth wave and an inverted instance of the same wave, with the two filtered signals. This calculates the real and the imaginary part of a complex wave. The real part is then subtracted from the imaginary part and is directed to an output which is then directed to the Stereo Delay Module.

The Speech Synthesis module is a bit more complicated since it does not solely rely on PD-extended to complete its task. If a message in the chatroom is preceded by the "talk" command, then anything following the command is being written in a buffer text file. Once this is done, the bot calls a system function that runs Text2Wave, a program that comes with the Festival Speech Synthesis System that converts text files to audio files. The command tells Text2Wave to use the buffer text file as an input file and synthesize the contents of this file to sound, saving it as a PCM Wave file. The PD-extended module that deals with the Speech Synthesis implementation looks for this file and plays it in a loop until the buffer text file is altered. When this happens, the PCM Wave file is altered as well, so the next time the loop is started it starts playing the new file. The Speech Synthesis module sound output is then fed to a Ring Modulator. The Ring Modulator module multiplies the sound with a carrier wave. The frequency of the carrier wave is controllable by OSC, enabling a user to actively control how the voice is effected during playback.

The output of the Ring Modulator module is directed to the Comb Filter module. This creates four instances of the input signal and combines them, applying slight delays to each instance. The level of each of the four instances is changed sequentially according to a time variable. The time variable can be altered via OSC as well. The output of the filter is directed to the Stereo Delay Module.

[47] As described in the help file for the "hilbert" object in PD

The Stereo Delay Module is a delay line with a feedback loop[48]. It was based on an earlier design of the author that also had a Graphical User Interface. This was used on the "Weird Synthesizer for Weird People"[49] project and was based on a design by Jason Plumb[50].

2.4 Encoding

The master signal output of the PD-extended patch is not directed to a "dac"[51] object, since we are not interested in directing the sound to the sound-card of the computer it runs on[52]. Instead, it is sent directly to the encoder, an "mp3cast" object that is part of the "Unauthorized" library[53]. This encodes the sound to the desired format. In this case, it uses the "LAME" library[54] to encode the audio into an MPEG Layer 3 stream and send it to the Icecast 2 Server, directly from PD-extended.

Several different ways of doing the conversion and streaming to the server had been investigated, prior to deciding on the final solution. Methods that were investigated included compiling Darkice[55] with LAME library support to enable it to encode using the MPEG Layer 3 format. However, Darkice required an input by a program that was compatible with it. An early experiment was to use "dac" object in PD-extended, together with Jack[56], a signal routing program for ALSA. Darkice had to be recompiled with support for Jack. The output of the "dac" object was then disconnected from the sound-card input and was directed to the input of Darkice instead. However, this set-up required two additional programs to be compiled, installed and running on the server, Darkice and Jack Daemon. It also required the Digital-to-Analogue conversion to be done within PD-extended. It was clearly not an efficient way of doing this and especially Darkice proved to be a program that heavily used the CPU.

It is also interesting to note that the whole Sound Synthesis Engine and Encoding section is totally server-based and does not have a Graphical User Interface. PD-extended is started from a bash shell[57] with real-time priority and no GUI loaded. Since the encoder is part of the PD-extended patch, it does not require any other external programs to be started.

[48] This means that the output is fed back to the input to create multiple delays

[49] Pure-Data based software synthesizer[Anagnostopoulos, 2008b]

[50] Jason Plumb maintains a website called Noisybox, hosting his own creative projects[Plumb, 2007]

[51] Digital to Analogue Converter, the most common object to be found as the last part of a data-flow programming audio patch

[52] The audio signals remain digital on the server-side, since the sound is meant to be reproduced by the client-side application

[53] Pure-Data library by Yves Degoyon[Degoyon, 2008]

[54] Mp3 encoder[Cheng et al., 1988]

[55] An audio streamer, written by Akos Maroy

[56] Jack is a virtual patchbay for signal routing

[57] Bourne Again SHell

2.5 "Plug and Play" implementation

The Otherside Server is designed to be a totally âĂIJhands-freeâĂİ system. The BIOS[58] of the machine running the Otherside has been configured to boot-up after power-loss. This means that as soon as the machine is plugged into mains power, it boots up. As it is designed around the philosophy of a Linux Server, it is head-less. This means that no monitor, keyboard or mouse is connected to the computer. In order for it to be shut down in case it needs to be moved or any other such case, it can be safely done by a simple press of the power button. The BIOS is configured to have a four second delay on the power button for a hard power-off. But an ACPI[59] daemon running on the computer ensures that the momentary push of the power button is recorded as an event. When this event is recorded, a safe and immediate shut-down script is called. The computer can then safely be unplugged from the power. It will automatically boot up again the next time it is plugged in.

All the necessary programs and server applications are started automatically upon boot-up using appropriate scripts. The "watchdog" application ensures everything remains up and running as long as the server is turned on. As soon as the machine is plugged in, the server will be up and running within a matter of minutes, with no further action necessary.

2.6 Administration

The administration of the Otherside Server is rarely required. However, in case the necessity for updates arises, there are a few simple additions to the Otherside System that make it an easy task to maintain.

An Open-SSH[60] Server is running on the machine, thus enabling secure remote administration over the internet. The SSH protocol allows encrypted TCP/IP[61] communication between two machines over a computer network.

The Subversion[62] client is installed on the machine, allowing for fast and simple version control of the applications. Together with âĂIJapt-getâĂİ[63], this makes a core for remote updating of the software as and when needed. The âĂIJapt-getâĂİ program installs, uninstalls or updates any packages on the Ubuntu repositories. These include newer Kernel versions, newer versions of libraries, and certain software updates such as the Apache 2 HTTP Server and the IRC 2 Server. Subversion allows the retrieval of any changes made to the software that has been developed and stored in an external Subversion Server, not related to the official Linux distribution. In case a bug is discovered that was not there in the last known stable version, Subversion allows an administrator to easily return to a previous version of the software. All this can be done through

[58] Basic Input/Output System[IBM, 1983]
[59] Advanced Configuration and Power Interface[Hockin, 2007]
[60] Secure SHell[OpenBSD, 2008]
[61] Transmission Control Protocol/Internet Protocol
[62] Version Control System[Collins-Sussman et al., 2004]
[63] The default package manager system in Ubuntu Linux

the SSH protocol, from any computer in any part of the world as long as there is an active internet connection.

With the software that is currently being used, updating can be done while the server is running, minimising the down-time to the time needed for a complete reboot. This is under five minutes, including the time needed to safely shut-down and restart. This can even be done remotely, with no need to have physical access to the actual hardware.

2.7 Expansiveness

The possibilities for expansion are virtually endless, owing to the modular design principles the Otherside System is based on. It does not solely depend on the developer, but also the community of the users and their imagination. The Otherside is open to user-side additions and expansions.

3 Conclusion

The Otherside is unique in several ways, compared to other examples of network collaboration systems. First and foremost, from a technical point of view, all processing is done on the server side and the sound is transfered to the listeners and users as an audio stream. This means that all users are listening to the same sonic output. In contrast, NetPD [64] only provides control data, which is meant to control a sound synthesis engine running on the client-side. This means that each user will be creating a totally different sound in a totally different way. A common problem associated with computer network collaboration systems is the difficulty is synchronisation of the audio due to network latency, especially in a situation where the collaboration might be between several users in the same room. If each of the users is producing audio independently, it is technically impossible[65] to ever synchronize the output. However, in the Otherside system, since all users are working towards a common sonic piece that is being created on the server side, only one computer needs to act as the audio reproduction client in each such setting. Therefore, in the collaboration situation where users are in the same room, the audio only needs to be streamed to one computer, coming out of one pair of loudspeakers. There are no synchronization problems, since all the users are doing is transmitting control data to the server, which creates one sonic artwork by synthesizing transmitted data in the order of arrival.

In the case of a network of Othersides, the servers all share the same user control data as IRC servers share all data between them when networked. However, each Otherside Server runs its own sound synthesis engine so audio data does not need to be transmitted between servers. All servers will create the exact same version of the audio stream, since they have received the same control data, and stream it independently. This is extremely useful as users can stream the audio off their local server, while being able to perceive the input of a user

[64] The NetPD Project[Haefeli, 2008]

[65] Due to the different clock frequencies and network latency values

connected to a different Otherside Server. This function makes the possibility for a true global collaboration very realistic.

3.1 Social Impact

There is an evident social side to the Otherside system, since the use of an IRC protocol allows users to easily communicate as they would in normal chat-rooms. They are given the opportunity to start their own chat-rooms and form âĂIJteamsâĂİ of users that cooperate for a desired sonic experiment. This can easily grow into a network of Otherside Servers in different parts of the world, who could collaborate in creating sound or developing the Otherside system. The IRC protocol is currently not being used at its full potential for chat services. It can easily form into a full IRC chat interface, with services enabling users to have a personalized identifier/nickname and build permanent subject-specific chatrooms. The Otherside presents the world with the opportunity to turn the idea of computer network collaboration platforms into a global social network.

The prototype Otherside Server is currently up and running at the University of Sheffield Sound Studios, Sheffield UK.

http://usss-otherside.shef.ac.uk

References

[Anagnostopoulos, 2008a] Anagnostopoulos, I. (2008a). Otherside server. http://usss-otherside.shef.ac.uk/, Accessed on 01/09/2008.

[Anagnostopoulos, 2008b] Anagnostopoulos, I. (2008b). Weird synthesizer for weird people. http://iforgotthem.tripod.com/ift.htm, Accessed on 01/09/2008.

[ArchGh0ul, 2007] ArchGh0ul (2007). Python irc bot skeleton. http://www.rohitab.com/discuss/index.php?showtopic=24081, Accessed on 01/09/2008.

[Beckett, 2008] Beckett, D. (2008). Julie, formerly known as redlandbot. http://crschmidt.net/julie/, Accessed on 01/09/2008.

[Black et al., 1999] Black, A. W., Caley, R., and Taylor, P. (1999). The festival speech synthesis system. http://fife.speech.cs.cmu.edu/festival/manual-1.4.1/festival-1.4.1.ps.gz, Accessed on 02/09/2008.

[Bristow-Johnson, 2008] Bristow-Johnson, R. (2008). Wavetable synthesis 101, a fundamental perspective. http://www.musicdsp.org/files/Wavetable-101.pdf, Accessed on 02/09/2008.

[Canonical, Inc., 2008] Canonical, Inc. (2008). Ubuntu linux. http://www.ubuntu.com/, Accessed on 01/09/2008.

[Cheng et al., 1988] Cheng, M., Taylor, M., et al. (1988). Lame ain't an mp3 encoder. http://lame.sourceforge.net/index.php, Accessed on 02/09/2008.

[Collins-Sussman et al., 2004] Collins-Sussman, B., Fitzpatrick, B., and Pilato, C. (2004). *Version Control with Subversion*. Sebastopol, California: O'Reily Media, Inc., first edition.

[Degoyon, 2008] Degoyon, Y. (2008). Unauthorized pure data. http://ydegoyon.free.fr/software.html, Accessed on 02/09/2008.

[Encyclopaedia Britannica, 2008] Encyclopaedia Britannica (2008). Aleatory music. http://www.britannica.com/EBchecked/topic/13676/aleatory-music, Accessed on 02/09/2008.

[Haefeli, 2008] Haefeli, R. (2008). Netpd. http://www.netpd.org/, Accessed on 01/09/2008.

[Hockin, 2007] Hockin, T. (2007). Acpi daemon. http://acpid.sourceforge.net/, Accessed on 01/09/2008.

[Holth et al., 2008] Holth, D., McChesney, C., and the ixi software group (2008). Simpleosc 0.2.5. http://www.ixi-software.net/content/download/simpleosc0.2.5.zip, Accessed on 01/09/2008.

[IBM, 1983] IBM (1983). *IBM Personal Computer Technical Reference manual*, first, revised march 1983 edition. page iii.

[Kalt, 2000] Kalt, C. (2000). *The IRC RFC-2813*. http://www.irc.org/tech_docs/ircnet/rfc2813.txt, Accessed on 01/09/2008.

[Kysela et al., 2008] Kysela, J. et al. (2008). Advanced linux sound architecture. http://www.alsa-project.org/main/index.php/Main_Page, Accessed on 01/09/2008.

[Leadbeater et al., 2008] Leadbeater, D. et al. (2008). Cgi:irc. http://cgiirc.org/, Accessed on 01/09/2008.

[Lehman, 2007] Lehman, S. (2007). Ring modulation. http://www.harmony-central.com/Effects/Articles/Ring_Modulation/, Accessed on 02/09/2008.

[MathWorks Team, 2008] MathWorks Team (2008). Single sideband modulation via the hilbert transform. http://www.mathworks.com/products/signal/demos.html?file=/products/demos/shipping/signal/hilberttransformdemo.html, Accessed on 02/09/2008.

[Matthews, 2007] Matthews, M. (2007). *The theory and techniques of electronic music*, chapter Foreword. Singapore: World Scientific. http://www-crca.ucsd.edu/ msp/techniques/latest/book-html/node5.html, Accessed on 01/09/2008.

[Oikarinen et al., 2008] Oikarinen, J. et al. (2008). Ircd-irc2 irc-net server. http://packages.ubuntu.com/hardy/net/ircd-irc2/, Accessed on 01/09/2008.

[Oikarinen and Reed, 1993] Oikarinen, J. and Reed, D. (1993). *The IRC RFC-1459*. http://www.irc.org/tech_docs/ircnet/rfc1459.txt, Accessed on 01/09/2008.

[OpenBSD, 2008] OpenBSD (2008). Open-ssh server. http://www.openssh.com/, Accessed on 01/09/2008.

[Plumb, 2007] Plumb, J. (2007). Noisybox. http://noisybox.net/computers/pd/, Accessed on 01/09/2008.

[Puckette, 2008] Puckette, M. (2008). Cycling 74. http://www.cycling74.com/products/max5/, Accessed on 01/09/2008.

[Puckette et al., 2008] Puckette, M., Zmoelnig, I., and the PD Community (2008). Pure-data. http://puredata.info/, Accessed on 01/09/2008.

[Python Software Foundation, 2008] Python Software Foundation (2008). Python programming language. http://www.python.org/, Accessed on 01/09/2008.

[Reid, 2000] Reid, G. (2000). Part 14: An introduction to additive synthesis. *Sound on Sound*. http://www.soundonsound.com/sos/jun00/articles/synthsec.htm, Accessed on 02/09/2008.

[Ritchie, 1993] Ritchie, D. M. (1993). The development of the c language. In Bergin, Jr., T. J. and Gibson, Jr., R. G., editors, *History of Programming Languages-II*. New York: ACM Press. http://cm.bell-labs.com/cm/cs/who/dmr/chist.html, Accessed on 01/09/2008.

[Smith III, 2007a] Smith III, J. O. (2007a). *Comb Filters*. Stanford University, California, may 2008 edition. http://ccrma.stanford.edu/ jos/pasp04/Comb_Filters.html, Accessed on 02/09/2008.

[Smith III, 2007b] Smith III, J. O. (2007b). *Delay Lines*. Stanford University, California, may 2008 edition. http://ccrma.stanford.edu/ jos/pasp/Delay_Lines.html, Accessed on 02/09/2008.

[Stelkens, 2003] Stelkens, J. (2003). peersynth: A p2p multi-user software synthesizer with new techniques for integrating latency in real time collaboration. In *Proceedings of the 2003 ICMC*. Singapore: ICMA.

[The Apache Software Foundation, 2008] The Apache Software Foundation (2008). Apache 2. http://httpd.apache.org/, Accessed on 01/09/2008.

[The XIPH Open Source Community, 2008] The XIPH Open Source Community (2008). Icecast 2 server. http://www.icecast.org/, Accessed on 01/09/2008.

[Wright, 2002] Wright, M. (2002). The open sound control 1.0 specification. http://opensoundcontrol.org/spec-1_0, Accessed on 01/09/2008.

Appendices

A The Otherbot IRC Bot

The Otherbot was written in the Python programming language. It is responsible for interfacing the IRC Server with the Sound Synthesis Engine. It connects to the IRC Server on startup, joins a pre-defined chatroom and waits there, keeping the connection alive. Then it responds to a set of predefined commands with predefined actions, while directing all data it does not understand to the Raw Data module. The redirection of the data is done by establishing UDP network connections to Pure-Data daemon objects.

A.1 Python Source Code

```
#!/usr/bin/python

    # OtherBot
    # IRC Bot for the man on the go...
    # Loosely based on XoRbot
    # Written by Jesus H. <ppsycho@mailcity.com>
    # <http://otherside.servebeer.com>
    # <http://iforgotthem.tripod.com>
    # Thanks to Dr. Dave Moore for the suggestions

    # Libraries
    import socket
    import string
    import sys
    import os
    from array import array
    import string
    import osc

    # Configuration
    TRUE=1
    FALSE=0

    OSCPORT=9998 #for OSC
```

```
UDPPORT=9997 #for RAW DATA

SERVER="192.168.1.68"
PORT=6667
BOTNICK="OtherBot"
THECHAN="#OtherSide"

PASS="pouti0"
AUTHED=FALSE

OWNER="jesus"
rBUFF=""

BUFR = 'touch /home/access/buffer.txt'
SPX = 'text2wave -f 44100 -o /home/access/buffer.wav /home/access/buffer.txt'

# Functions
def parseARGS(args):
args = args.split('"')
args = args[1:] #remove the user info...
args.remove(' ')
args.remove('')
#print args
return args

def parse_talk(args):
args = args.split(':')
return args[2][5:]

def parse_raw(args):
args = args.split(':')
return args[2]

def parse_speak(args):
args = args.split(':')
return args[2][6:]

def parse_cmd_args(cmd, args):
args = args.split(':')
return args[2][len(cmd)+1:].split(' ')

def linetostring(line):
s = ""
for v in line:
s += v # + " " (no space needed now)
return s
```

```
    # Initialize connections
    connection=socket.socket( ) #for TCP
    connection.connect((SERVER, PORT))
    connection.send("NICK %s\r\n" % BOTNICK) #IRC stuff
    connection.send("USER %s %s bla :%s\r\n" % (BOTNICK, SERVER, BOT-
NICK))
    connection.send("JOIN :%s\r\n" % THECHAN)
    osc.init() #for OSC
    udpsock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #for
UDP
    udpsock.connect((SERVER, UDPPORT))
    os.system(BUFR)

    # It's a dirty job, BOT somebody's gotta do it... (commands we're listening
for)
    while 1:
        try:
        rBUFF=rBUFF+connection.recv(512)
        temp=string.split(rBUFF, "\r\n")
        rBUFF=temp.pop( )
        except:
    connection.close()
    sys.exit(1)

        for line in temp:
            line=string.rstrip(line)
    rawLine = line
            line=string.split(line)
    WHO=line[0].split('!')
    WHO=WHO[0].strip('1:')
    if len(line) > 3:
    WHERE = line[2]
    try:
    if WHERE <> BOTNICK:
    WHERE=WHERE
    else:
    WHERE=WHO
    THECMD = line[3][1:]

    if THECMD == '+exit' and AUTHED == TRUE and WHO == OWNER:
    connection.send("PRIVMSG %s :%c4,0 EXITING \r\n" % (WHERE,3))
    connection.send("QUIT \r\n")
    connection.close()
    break
    sys.exit(0)
    elif THECMD == '+slap' and AUTHED == TRUE and WHO == OWNER:
```

```
    args = parse_cmd_args('+slap',rawLine)
    connection.send("PRIVMSG %s :%s slaps %s around a bit with a large trout
:D\r\n" % (WHERE, args[0], args[1]))
    elif THECMD == '+speak' and AUTHED == TRUE and WHO == OWNER:
    speak = parse_speak(rawLine)
    args = parse_cmd_args('+speak',rawLine)
    connection.send("PRIVMSG %s :%s\r\n" % (args[0], speak[len(args[0])+1:]))
    elif THECMD == '+nick' and AUTHED == TRUE and WHO == OWNER:
    args = parse_cmd_args('+nick',rawLine)
    connection.send("NICK :%s\r\n" % (args[0]))
    elif THECMD == '+join' and AUTHED == TRUE and WHO == OWNER:
    args = parse_cmd_args('+join',rawLine)
    connection.send("JOIN :%s\r\n" % (args[0]))
    elif THECMD == '+part' and AUTHED == TRUE and WHO == OWNER:
    args = parse_cmd_args('+part',rawLine)
    connection.send("PART :%s\r\n" % (args[0]))
    elif THECMD == '+login':
    args = parse_cmd_args('+login',rawLine)
    # print args (was here for debug output)
    if args[0]==PASS and AUTHED == FALSE:
    AUTHED = TRUE
    connection.send("PRIVMSG %s :%s LOGGED IN \r\n" % (WHERE, WHO))
    print WHO + " is authed"
    elif THECMD == 'osc':
    args = parse_cmd_args('osc',rawLine)
    osc.sendMsg(args[0], args[1:], SERVER, OSCPORT)
    elif THECMD == 'talk':
    msg = parse_talk(rawLine)
    f = open('/home/access/buffer.txt', 'w')
    f.write(msg)
    f.close()
    os.system(SPX)
    elif THECMD == 'midi':
    connection.send("PRIVMSG %s :No Midi implementation in this version,
%s \r\n" % (WHERE, WHO))
    elif THECMD == 'help':
    connection.send("PRIVMSG %s :Prefix messages with osc for OSC data,
midi for MIDI data, talk for speech synthesis implementation or anything else
for raw ASCII data, %s \r\n" % (WHERE, WHO))
    else:
    raw = parse_raw(rawLine)
    udpsock.send(linetostring(raw))
    except:
    # print "fuck! ... Unexpected error:", sys.exc_info()[0], sys.exc_info()[1]
    # ---Uncomment above to see debug info---
    print "Connected to IRC Server! \r\n"
```

```
    # The above works simply because an exception should occur only when
connecting to the server!
    if(line[0]=="PING"):
    connection.send("PONG %s\r\n" % line[1])
    if line[1] == "KICK" and line[3] == BOTNICK:
    connection.send("JOIN %s\r\n" % line[2])
```

B Apache 2 Configuration files

Based on the original configuration files that come with the Apache 2 Ubuntu Package from the Ubuntu 8.04 repositories[66].

B.1 /etc/apache2/apache2.conf

```
#
    # Based upon the NCSA server configuration files originally by Rob McCool.
    #
    # This is the main Apache server configuration file.  It contains the
    # configuration directives that give the server its instructions.
    # See http://httpd.apache.org/docs/2.2/ for detailed information about
    # the directives.
    #
    # Do NOT simply read the instructions in here without understanding
    # what they do.  They're here only as hints or reminders.  If you are unsure
    # consult the online docs. You have been warned.
    #
    # The configuration directives are grouped into three basic sections:
    #  1. Directives that control the operation of the Apache server process as
a
    #     whole (the 'global environment').
    #  2. Directives that define the parameters of the 'main' or 'default' server,
    #     which responds to requests that aren't handled by a virtual host.
    #     These directives also provide default values for the settings
    #     of all virtual hosts.
    #  3. Settings for virtual hosts, which allow Web requests to be sent to
    #     different IP addresses or hostnames and have them handled by the
    #     same Apache server process.
    #
    # Configuration and logfile names: If the filenames you specify for many
    # of the server's control files begin with "/" (or "drive:/" for Win32), the
    # server will use that explicit path.  If the filenames do *not* begin
    # with "/", the value of ServerRoot is prepended -- so "/var/log/apache2/foo.log"
    # with ServerRoot set to "" will be interpreted by the
```

[66] Apache 2[The Apache Software Foundation, 2008] and Ubuntu Linux [Canonical, Inc., 2008] as mentioned in the References.

```
    # server as "//var/log/apache2/foo.log".
    #

    ### Section 1: Global Environment
    #
    # The directives in this section affect the overall operation of Apache,
    # such as the number of concurrent requests it can handle or where it
    # can find its configuration files.
    #

    #
    # ServerRoot: The top of the directory tree under which the server's
    # configuration, error, and log files are kept.
    #
    # NOTE!  If you intend to place this on an NFS (or otherwise network)
    # mounted filesystem then please read the LockFile documentation (avail-
able
    # at <URL:http://httpd.apache.org/docs-2.1/mod/mpm_common.html#lockfile>);
    # you will save yourself a lot of trouble.
    #
    # Do NOT add a slash at the end of the directory path.
    #
    ServerRoot "/etc/apache2"

    #
    # The accept serialization lock file MUST BE STORED ON A LOCAL
DISK.
    #
    #<IfModule !mpm_winnt.c>
    #<IfModule !mpm_netware.c>
    LockFile /var/lock/apache2/accept.lock
    #</IfModule>
    #</IfModule>

    #
    # PidFile: The file in which the server should record its process
    # identification number when it starts.
    #
    PidFile /var/run/apache2.pid

    #
    # Timeout: The number of seconds before receives and sends time out.
    #
    Timeout 300

    #
```

```
    # KeepAlive: Whether or not to allow persistent connections (more than
    # one request per connection). Set to "Off" to deactivate.
    #
    KeepAlive On

    #
    # MaxKeepAliveRequests: The maximum number of requests to allow
    # during a persistent connection. Set to 0 to allow an unlimited amount.
    # We recommend you leave this number high, for maximum performance.
    #
    MaxKeepAliveRequests 100

    #
    # KeepAliveTimeout: Number of seconds to wait for the next request from
the
    # same client on the same connection. (was 15)
    #
    KeepAliveTimeout 5

    ##
    ## Server-Pool Size Regulation (MPM specific)
    ##

    # prefork MPM
    # StartServers: number of server processes to start
    # MinSpareServers: minimum number of server processes which are kept
spare
    # MaxSpareServers: maximum number of server processes which are kept
spare
    # MaxClients: maximum number of server processes allowed to start
    # MaxRequestsPerChild: maximum number of requests a server process
serves
    <IfModule mpm_prefork_module>
        StartServers          5
        MinSpareServers       5
        MaxSpareServers      10
        MaxClients          150
        MaxRequestsPerChild   0
    </IfModule>

    # worker MPM
    # StartServers: initial number of server processes to start
    # MaxClients: maximum number of simultaneous client connections
    # MinSpareThreads: minimum number of worker threads which are kept
spare
```

```
    # MaxSpareThreads: maximum number of worker threads which are kept
spare
    # ThreadsPerChild: constant number of worker threads in each server pro-
cess
    # MaxRequestsPerChild: maximum number of requests a server process
serves
    <IfModule mpm_worker_module>
        StartServers          2
        MaxClients          150
        MinSpareThreads      25
        MaxSpareThreads      75
        ThreadsPerChild      25
        MaxRequestsPerChild   0
    </IfModule>

    User www-data
    Group www-data

    #
    # AccessFileName: The name of the file to look for in each directory
    # for additional configuration directives.  See also the AllowOverride
    # directive.
    #

    AccessFileName .htaccess

    #
    # The following lines prevent .htaccess and .htpasswd files from being
    # viewed by Web clients.
    #
    <Files ~ "^\.ht">
        Order allow,deny
        Deny from all
    </Files>

    #
    # DefaultType is the default MIME type the server will use for a document
    # if it cannot otherwise determine one, such as from filename extensions.
    # If your server contains mostly text or HTML documents, "text/plain" is
    # a good value.  If most of your content is binary, such as applications
    # or images, you may want to use "application/octet-stream" instead to
    # keep browsers from trying to display binary files as though they are
    # text.
    #
    DefaultType text/plain
```

```
    #
    # HostnameLookups: Log the names of clients or just their IP addresses
    # e.g., www.apache.org (on) or 204.62.129.132 (off).
    # The default is off because it'd be overall better for the net if people
    # had to knowingly turn this feature on, since enabling it means that
    # each client request will result in AT LEAST one lookup request to the
    # nameserver.
    #
    HostnameLookups Off

    # ErrorLog: The location of the error log file.
    # If you do not specify an ErrorLog directive within a <VirtualHost>
    # container, error messages relating to that virtual host will be
    # logged here.  If you *do* define an error logfile for a <VirtualHost>
    # container, that host's errors will be logged there and not here.
    #
    ErrorLog /var/log/apache2/error.log

    #
    # LogLevel: Control the number of messages logged to the error_log.
    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    #
    LogLevel warn

    # Include module configuration:
    Include /etc/apache2/mods-enabled/*.load
    Include /etc/apache2/mods-enabled/*.conf

    # Include all the user configurations:
    Include /etc/apache2/httpd.conf

    # Include ports listing
    Include /etc/apache2/ports.conf

    #
    # The following directives define some format nicknames for use with
    # a CustomLog directive (see below).
    #
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-
Agent}i\"" combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common
    LogFormat "%{Referer}i -> %U" referer
    LogFormat "%{User-agent}i" agent

    #
```

```
    # ServerTokens
    # This directive configures what you return as the Server HTTP response
    # Header. The default is 'Full' which sends information about the OS-Type
    # and compiled in modules.
    # Set to one of:  Full | OS | Minor | Minimal | Major | Prod
    # where Full conveys the most information, and Prod the least.
    #
    ServerTokens Minimal

    #
    # Optionally add a line containing the server version and virtual host
    # name to server-generated pages (internal error documents, FTP directory
    # listings, mod_status and mod_info output etc., but not CGI generated
    # documents or custom error documents).
    # Set to "EMail" to also include a mailto: link to the ServerAdmin.
    # Set to one of:  On | Off | EMail
    #
    ServerSignature On



    #
    # Customizable error responses come in three flavors:
    # 1) plain text 2) local redirects 3) external redirects
    #
    # Some examples:
    #ErrorDocument 500 "The server made a boo boo."
    #ErrorDocument 404 /missing.html
    #ErrorDocument 404 "/cgi-bin/missing_handler.pl"
    #ErrorDocument 402 http://www.example.com/subscription_info.html
    #

    #
    # Putting this all together, we can internationalize error responses.
    #
    # We use Alias to redirect any /error/HTTP_<error>.html.var response
to
    # our collection of by-error message multi-language collections.  We use
    # includes to substitute the appropriate text.
    #
    # You can modify the messages' appearance without changing any of the
    # default HTTP_<error>.html.var files by adding the line:
    #
    #   Alias /error/include/ "/your/include/path/"
    #
    # which allows you to create your own set of files by starting with the
```

```
    # /usr/share/apache2/error/include/ files and copying them to /your/include/path/,
    # even on a per-VirtualHost basis.  The default include files will display
    # your Apache version number and your ServerAdmin email address regard-
less
    # of the setting of ServerSignature.
    #
    # The internationalized error documents require mod_alias, mod_include
    # and mod_negotiation.  To activate them, uncomment the following 30
lines.

    #    Alias /error/ "/usr/share/apache2/error/"
    #
    #    <Directory "/usr/share/apache2/error">
    #        AllowOverride None
    #        Options IncludesNoExec
    #        AddOutputFilter Includes html
    #        AddHandler type-map var
    #        Order allow,deny
    #        Allow from all
    #        LanguagePriority en cs de es fr it nl sv pt-br ro
    #        ForceLanguagePriority Prefer Fallback
    #    </Directory>
    #
    #    ErrorDocument 400 /error/HTTP_BAD_REQUEST.html.var
    #    ErrorDocument 401 /error/HTTP_UNAUTHORIZED.html.var
    #    ErrorDocument 403 /error/HTTP_FORBIDDEN.html.var
    #    ErrorDocument 404 /error/HTTP_NOT_FOUND.html.var
    #   ErrorDocument 405 /error/HTTP_METHOD_NOT_ALLOWED.html.var
    #    ErrorDocument 408 /error/HTTP_REQUEST_TIME_OUT.html.var
    #    ErrorDocument 410 /error/HTTP_GONE.html.var
    #    ErrorDocument 411 /error/HTTP_LENGTH_REQUIRED.html.var
    #   ErrorDocument 412 /error/HTTP_PRECONDITION_FAILED.html.var
    #   ErrorDocument 413 /error/HTTP_REQUEST_ENTITY_TOO_LARGE.html.var
    #   ErrorDocument 414 /error/HTTP_REQUEST_URI_TOO_LARGE.html.var
    #   ErrorDocument 415 /error/HTTP_UNSUPPORTED_MEDIA_TYPE.html.var
    #   ErrorDocument 500 /error/HTTP_INTERNAL_SERVER_ERROR.html.var
    #    ErrorDocument 501 /error/HTTP_NOT_IMPLEMENTED.html.var
    #    ErrorDocument 502 /error/HTTP_BAD_GATEWAY.html.var
    #   ErrorDocument 503 /error/HTTP_SERVICE_UNAVAILABLE.html.var
    #   ErrorDocument 506 /error/HTTP_VARIANT_ALSO_VARIES.html.var



    # Include of directories ignores editors' and dpkg's backup files,
    # see README.Debian for details.
```

```
# Include generic snippets of statements
Include /etc/apache2/conf.d/

# Include the virtual host configurations:
Include /etc/apache2/sites-enabled/
```

B.2 /etc/apache2/httpd.conf

B.3 /etc/apache2/ports.conf

```
Listen 80

<IfModule mod_ssl.c>
    Listen 443
</IfModule>
```

C CGI:IRC Configuration files

Based on the original configuration files that come with the CGI:IRC Ubuntu Package from the Ubuntu 8.04 repositories[67].

C.1 /etc/cgiirc/cgiirc.config

```
# CGI:IRC configuration file.
    #
    # Check /usr/share/doc/cgiirc/examples/cgiirc.config.full.gz
    # for more details.
    # Take care about applying debian-specific settings like
    # `image_path' if you intend to just copy it!

    default_server = otherside
    default_port = 8079
    default_channel = #OtherSide
    default_name = AnotherSide User
    default_nick=OtherSider???

    #Appearance Settings (my own addition)

    login basic = Nickname

    login advanced = Nickname, Realname, Format, Character set

    #What commands a user has access to (Safety) (my own addition)
```

[67] CGI:IRC[Leadbeater et al., 2008] and Ubuntu Linux [Canonical, Inc., 2008] as mentioned in the References.

```
#access_command = msg me help join nick quit !

#Timeout Time (my own addition)

session_timeout = 18000

#Maximum number of users (my own addition)

max_users = 1000

# Don't change these, they're specific to Debian:
# Not working at the moment! (path = /usr/share/images/cgiirc)
image_path = /images/cgiirc
# ------------------------------------------------

script_nph = nph-irc.cgi
script_form = client-perl.cgi
script_login = irc.cgi

ip_access_file = ipaccess
```

C.2 /etc/cgiirc/ipaccess

```
# CGI:IRC ipaccess file. (For CGI:IRC versions from 0.5.3).
#
# Check /usr/share/doc/cgiirc/examples/ipaccess.example
# for more details.
#
# Safe default configuration: Access only for localnet.
#143.167.0.0/16

#Global access
*.*.*.*
```

D Icecast 2 Configuration files

Based on the original configuration files that come with the Icecast 2 Ubuntu Package from the Ubuntu 8.04 repositories[68].

[68] Icecast 2[The XIPH Open Source Community, 2008] and Ubuntu Linux [Canonical, Inc., 2008] as mentioned in the References.

D.1 /etc/icecast2/icecast.xml

```
<icecast>
        <limits>
            <clients>100</clients>
            <sources>2</sources>
            <threadpool>5</threadpool>
            <queue-size>524288</queue-size>
            <client-timeout>30</client-timeout>
            <header-timeout>15</header-timeout>
            <source-timeout>10</source-timeout>
            <!-- If enabled, this will provide a burst of data when a client
                 first connects, thereby significantly reducing the startup
                 time for listeners that do substantial buffering. However,
                 it also significantly increases latency between the source
                 client and listening client.  For low-latency setups, you
                 might want to disable this. -->
            <burst-on-connect>1</burst-on-connect>
            <!-- same as burst-on-connect, but this allows for being more
                 specific on how much to burst. Most people won't need to
                 change from the default 64k. Applies to all mountpoints  -->
            <burst-size>65535</burst-size>
        </limits>

        <authentication>
            <!-- Sources log in with username 'source' -->
            <source-password>password</source-password>
            <!-- Relays log in username 'relay' -->
            <relay-password>password</relay-password>

            <!-- Admin logs in with the username given below -->
            <admin-user>user</admin-user>
            <admin-password>password</admin-password>
        </authentication>

        <!-- Uncomment this if you want directory listings -->
        <!--
        <directory>
            <yp-url-timeout>15</yp-url-timeout>
            <yp-url>http://dir.xiph.org/cgi-bin/yp-cgi</yp-url>
        </directory>
         -->

        <!-- This is the hostname other people will use to connect to your server.
        It affects mainly the urls generated by Icecast for playlists and yp
        listings. -->
```

```
<hostname>http://otherside.servebeer.com</hostname>

<!-- You can use these two if you only want a single listener -->
<!--<port>8000</port> -->
<!--<bind-address>127.0.0.1</bind-address>-->

<!-- You may have multiple <listener> elements -->
<listen-socket>
    <port>8000</port>
    <!-- <bind-address>127.0.0.1</bind-address> -->
</listen-socket>
<!--
<listen-socket>
    <port>8001</port>
</listen-socket>
-->

<!--<master-server>127.0.0.1</master-server>-->
<!--<master-server-port>8001</master-server-port>-->
<!--<master-update-interval>120</master-update-interval>-->
<!--<master-password>hackme</master-password>-->

<!-- setting this makes all relays on-demand unless overridden, this is
     useful for master relays which do not have <relay> definitions here.
     The default is 0 -->
<!--<relays-on-demand>1</relays-on-demand>-->

<!--
<relay>
    <server>127.0.0.1</server>
    <port>8001</port>
    <mount>/example.ogg</mount>
    <local-mount>/different.ogg</local-mount>
    <on-demand>0</on-demand>

    <relay-shoutcast-metadata>0</relay-shoutcast-metadata>
</relay>
-->

<!-- Only define a <mount> section if you want to use advanced options,
     like alternative usernames or passwords
<mount>
    <mount-name>/example-complex.ogg</mount-name>

    <username>othersource</username>
    <password>hackmemore</password>
```

```
        <max-listeners>1</max-listeners>
        <dump-file>/tmp/dump-example1.ogg</dump-file>
        <burst-size>65536</burst-size>
        <fallback-mount>/example2.ogg</fallback-mount>
        <fallback-override>1</fallback-override>
        <fallback-when-full>1</fallback-when-full>
        <intro>/example_intro.ogg</intro>
        <hidden>1</hidden>
        <no-yp>1</no-yp>
        <authentication type="htpasswd">
                <option name="filename" value="myauth"/>
                <option name="allow_duplicate_users" value="0"/>
        </authentication>
        <on-connect>/home/icecast/bin/stream-start</on-connect>
        <on-disconnect>/home/icecast/bin/stream-stop</on-disconnect>
    </mount>

    <mount>
        <mount-name>/auth_example.ogg</mount-name>
        <authentication type="url">
        <option name="mount_add"      value="http://myauthserver.net/notify_mount.php"/>
        <option name="mount_remove"   value="http://myauthserver.net/notify_mount.php"/>
        <option name="listener_add"   value="http://myauthserver.net/notify_listener.php"/>
        <option name="listener_remove" value="http://myauthserver.net/notify_listener.php"/>
        </authentication>
    </mount>

    -->

    <fileserve>1</fileserve>

    <!-- set the mountpoint for a shoutcast source to use, the default if not
         specified is /stream but you can change it here if an alternative is
         wanted or an extension is required
    <shoutcast-mount>/live.nsv</shoutcast-mount>
    -->

    <paths>
<!-- basedir is only used if chroot is enabled -->
        <basedir>/usr/share/icecast2</basedir>

        <!-- Note that if <chroot> is turned on below, these paths must both
             be relative to the new root, not the original root -->
        <logdir>/var/log/icecast2</logdir>
        <webroot>/usr/share/icecast2/web</webroot>
        <adminroot>/usr/share/icecast2/admin</adminroot>
```

```
        <!-- <pidfile>/usr/share/icecast2/icecast.pid</pidfile> -->

        <!-- Aliases: treat requests for 'source' path as being for 'dest' path
             May be made specific to a port or bound address using the "port"
             and "bind-address" attributes.
          -->
        <!--
        <alias source="/foo" dest="/bar"/>
          -->
        <!-- Aliases: can also be used for simple redirections as well,
             this example will redirect all requests for http://server:port/ to
             the status page
          -->
        <alias source="/" dest="/status.xsl"/>
    </paths>

    <logging>
        <accesslog>access.log</accesslog>
        <errorlog>error.log</errorlog>
        <!-- <playlistlog>playlist.log</playlistlog> -->
      <loglevel>4</loglevel> <!-- 4 Debug, 3 Info, 2 Warn, 1 Error -->
      <logsize>10000</logsize> <!-- Max size of a logfile -->
        <!-- If logarchive is enabled (1), then when logsize is reached
           the logfile will be moved to [error|access|playlist].log.DATESTAMP,
             otherwise it will be moved to [error|access|playlist].log.old.
             Default is non-archive mode (i.e. overwrite)
        -->
        <!-- <logarchive>1</logarchive> -->
    </logging>

    <security>
        <chroot>0</chroot>

        <changeowner>
            <user>icecast2</user>
            <group>nogroup</group>
        </changeowner>

    </security>
</icecast>
```

E IRC Server Configuration Files

Based on the original configuration files that come with the IRCd IRC2 Ubuntu Package from the Ubuntu 8.04 repositories[69].

E.1 /etc/ircd/iauth.conf

```
# If iauth timeouts, then reject user
    notimeout

    # This makes the IRC server require that iauth performs the authentication
    # in order for a new user connection to be accepted
    required

    # Perform ident lookups
    module rfc931

    # Modules below this keyword will work in delayed execution mode.
    # This means client will be allowed to enter irc and if any module below
    # decides it shouldn't have, this client will be removed.
    #delayed

    # Check and reject open SOCKS proxies
    #module socks
    # port = 1080
    # option = reject,paranoid
    # reason = Denied access (insecure proxy found)

    #module socks
    # port = 559
    # option = reject,paranoid
    # reason = Denied access (insecure proxy found)

    # Check and reject HTTP CONNECT proxies on port 8080
    #module webproxy
    # port = 8080
    # option = reject
    # reason = Denied access (insecure proxy found)

    # Check and reject HTTP CONNECT proxies on port 3128
    #module webproxy
    # port = 3128
    # option = reject,careful
    # reason = Denied access (insecure proxy found)
```

[69] IRC-Net IRC Daemon[Oikarinen et al., 2008] and Ubuntu Linux [Canonical, Inc., 2008] as mentioned in the References.

E.2 /etc/ircd/ircd.conf

```
# This is ircd's config-file. Look at /usr/share/doc/ircd-irc2/example.conf
    # and /usr/share/doc/ircd-irc2/example.conf for more detailled information
    # and instructions

    # M-Line
    M:OtherSide.Chat::Other Side Chat Server::826A

    # A-Line
    A:Other Side:Modular Chat and Sonic Interface:Contact <ppsycho@mailcity.com>
for questions::OtherSideNet

    # O-Lines
    O:*:EncryptedPassword:Username::10:A:

    # Y-Lines
    Y:1:90::100:512000:5.5:100.100
    Y:2:90::300:512000:5.5:250.250

    # I-Line
    I:*:::0:1
    I:127.0.0.1/32:::0:1

    # P-Line
    P::::8079:
```

E.3 /etc/ircd/ircd.motd

```
                    [ The Other Side ]
    |------------------------------------------------------------------------|
    | Welcome to the Other Side. The OtherSide.Chat IRC Server provides high
|
    | quality chat and control interface services. You can either chat with  |
    | Other Siders or control a sophisticated sound synthesis engine which   |
    | is streamed in real-time, or both.                -Jesus-              |
    |------------------------------------------------------------------------|
```

F Startup Script

Based on the implementation by the Ubuntu operating system[70].

[70] Ubuntu local startup script[Canonical, Inc., 2008].

F.1 /etc/rc.local

```
#!/bin/sh -e
    #
    # rc.local
    #
    # This script is executed at the end of each multiuser runlevel.
    # Make sure that the script will "exit 0" on success or any other
    # value on error.
    #
    # In order to enable or disable this script just change the execution
    # bits.
    #
    # By default this script does nothing.

    su - access -c "pd -rt -nogui /home/access/otherside/otherside.pd &"

    sleep 50

    su - access -c "otherbot &"

    sleep 10

    exit 0
```

G Block Diagrams

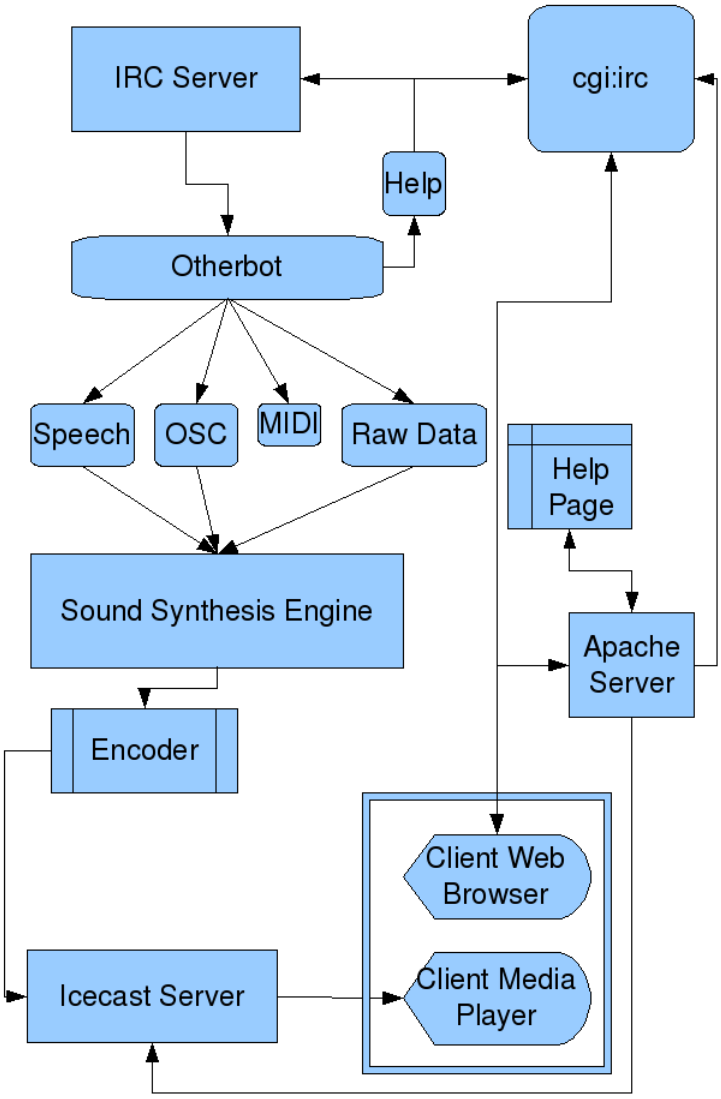
The following Block Diagrams demonstrate the functions and modules of the Otherside Server, the data flow between them and what the interaction is between a client[71] and the Otherside Server. The blocks either represent separate programs, or modules within one large program. The arrows on the connectors represent the signal flow direction.

On the General Block Diagram[72], the only Audio connections are from the Sound Synthesis Engine to the Encoder, from the Encoder to the Icecast Server and from the Icecast Server to the Client Media Player. Therefore, the only network connection that is being used to transfer audio is the one from the Icecast Server to the Client Media Player. All other connections are carrying control data.

On the Sound Synthesis Engine Diagram[73], a color coding scheme is in effect, to make it easier to distinguish between the Audio connections and the control data connections. No actual network connections exist within the Sound Synthesis Engine as shown on the Diagram. All the connections are between modules of Pure Data, Festival and the audio encoder.

[71] A client is any user of the Otherside Server.
[72] Appendix D.1
[73] Appendix D.2

G.1 General Block Diagram



G.2 Sound Synthesis Engine

The black connections represent control data, the red ones represent audio signals and the green one represents the final output.