The WFS system at La Casa del Suono, Parma

Fons Adriaensen
Casa della Musica
Pzle. San Francesco
43000 Parma (PR),
Italy,
fons@kokkinizita.net

Abstract

At the start of 2009 a 189-channel Wave Field Synthesis system was installed at the Casa del Suono in Parma, Italy. All audio and control processing required to run the system is performed by four Linux machines. The software used is a mix of standard Linux audio applications and some new ones developed specially for this installation. This paper discusses the main technical issues involved, including sections on the audio hardware, the digital signal processing algorithms, and the software used to control and manage the system.

Keywords

Wave field synthesis, spatial audio, distributed audio systems

1 Introduction

La Casa del Suono in Parma, Italy, is one of the cultural entities managed by the city of Parma, located in a small restored church and open to the general public. It is in the first place a museum dedicated to the history of audio technology, showing a collection of vintage audio equipment. It also aims to provide to its visitors a view on current developments in this area.

One of the installations designed for that purpose is the Wave Field Synthesis system installed in the *Sala Bianca*, shown in fig. 1. This is a room of around 7.5 by 4.5m meters and 4.5m high. When the doors (seen in the back) are closed, there is a continuous ring of 189 small speakers running along the complete inner circumference of the room, one every 12cm (with some small exceptions due to constructional constraints). The speakers are hidden behind a white tissue covering all the walls, and are constructed in 'blocks' of 10, 15, or 17 units. They are a bass-reflex design consisting of a small four inch bass and midrange unit (from Ciare) and a tweeter, driven by a passive crossover network. The height of the ring is a compromise between typical ear height for a seated and a standing audience. The usable frequency range is 50Hz to 20kHz, and sound quality is remarkably good for a design of this size. Except for the space taken by the speakers the walls are completely covered by sound absorbing material, leading to reasonably good 'dead' acoustics.

Figure 1: La Sala Bianca

As a project the Sala Bianca is the result of a collaboration between the city of Parma and the Engineering Department of the University of Parma. The university in turn contracted the author to specify the audio hardware and design the complete software.

The system is intended to be used both as a public demonstration of WFS technology, and as a scientific research instrument. This has re-

sulted in some quite peculiar and contradictory requirements. In the first role the system is operated by the museum staff, and it has to run fully automatically every day and without any technical support. As a scientific instrument on the other hand it has to be reconfigurable without any artificial limits, allowing the researchers to e.g. easily replace part of the software with experimental versions, or use it as a listening room for psycho-acoustic experiments using entirely different rendering algorithms. After the system was constructed a third way of using it was added as the direction of La Casa della Musica (who are managing the installation) also expressed their desire to use the Sala Bianca as an instrument for electro-acoustic music.

This is of course not the first large-scale WFS installation using Linux. A much larger one was constructed in 2007 at the Technical University of Berlin, see [Baalman et al., 2007]. The two systems are however quite different both in the hardware and software solutions that were adopted.

2 A short introduction to WFS

Wave Field Synthesis operates by reconstructing the wavefront that would be generated by a real sound source, called the *primary source,* by a large number of real *secondary sources*. The method used is based on the Huygens principle, and formally expressed by the Kirchoff integral [Verheyen, 1998]. It only works up to a frequency determined by the distance between the secondary sources, which should be less than half the wavelenght. Above this frequency, spatial aliasing will lead to false images of the primary source being generated as well as the correct one. For this reason most WFS systems use linear arrays of closely spaced secondary sources, as filling a plane would increase the required number above practical limits. Reducing a WFS system to 2-D operation has some consequences: it complicates the signal processing, and it also leads to an approximate solution that however still works very well in practice. Marije Baalman's recently published book [Baalman, 2008] provides a good overview of the state of WFS technology today, including some features not covered by the system discussed in this paper.

A WFS system can generate virtual sound sources either behind or in front of the secondary sources. The latter, in a complete surround setup as the Sala Bianca or the Berlin system, means virtual sources that appear to be *inside* the room.

Each virtual source is in principle reproduced by all speakers (in practice about half of them). For each pair of primary and secondary sources different gain factors, delays and filtering are required. So the complexity of the DSP software is proportional to $N_{primary\_sources} \times N_{secondary\_sources}$

The system installed in the Sala Bianca is designed to create up to 48 moving virtual sources in real time. Movement of the primary sources is continuous - it is not implemented by crossfading between fixed points but by adjusting the synthesis parameters at the full sample rate. This means that also the Doppler effect of a moving source is reproduced faithfully.

3 The audio system

Figure 2 shows the structure of the audio system. There are four PCs, all of them from the Siemens/Fujitsu Celsius range of workstations. This choice was influenced largely by the IT department of the city of Parma which imposed its requirements regarding suppliers, warranty conditions etc., but it turned out to be a very good one. At the time of writing all these machines are running Fedora 8 (installed two years ago), but they will all be converted to ArchLinux in the coming weeks. This distro makes it easy to use a much leaner system (e.g. without a fat desktop such as Gnome or KDE).

All machines are fitted with an RME HDSP-MADI interface providing 64 channels of digital audio input and output. RME ADI648 units are used to convert between MADI and ADAT, the latter being the format required by all AD/DA units used in the system. All machines are on a local gigabit LAN, and for audio they are connected to each other using an RME MADI matrix. The matrix has actually 8 inputs and 8 outputs, the remaining ones being used to connect to the 'Lampadario Acustico', another audio system installed at the CdS, and to provide connections for external PCs of researchers and musicians. All units share the same sample clock, transmitted either by the MADI links or by explicit word clock connections.

The **wfsmaster** machine coordinates the operation of the the WFS system, and provides all the access points to external applications using it. It is also the only machine having a human interface.

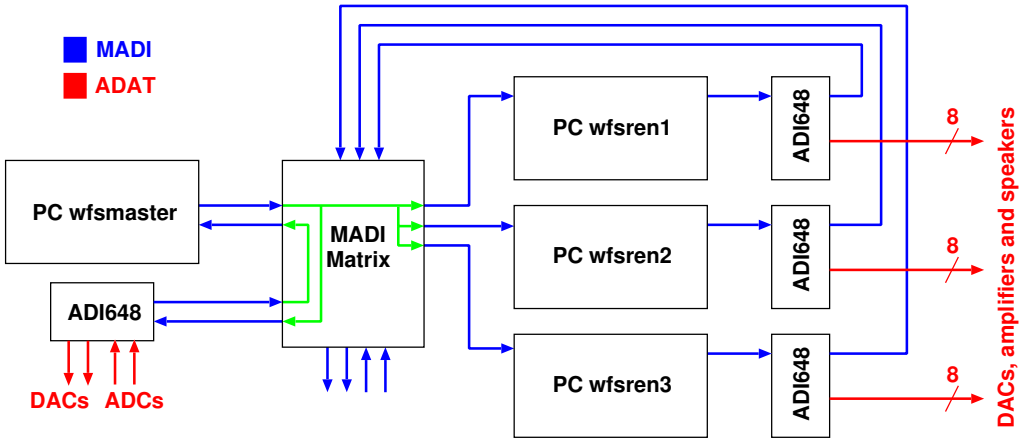

Figure 2: The audio system

The three **wfsrender** machines each take care of 64, 61, and 64 speaker outputs respectively, for a total of 189. They run 'headless' in Unix runlevel 3, and are fully remotely controlled.

All the equipment shown in fig 2, except the ADI648 units of the rendering machines is located in a small control room adjacent to the Sala Bianca. The three ADI648 connected to the rendering machines and all DA convertors and amplifiers are located two large racks in a separate technical room. The entire system is powered by a large UPS.

The normal MADI connections are as shown in fig 2: the 64 outputs of the the master machine are connected to the inputs of all three rendering machines. The hardware ensures that audio is synchronised system-wide - if the rendering machines all use the same Jack period size and would just copy one input to all outputs then an audio signal provided by the master machine would appear on all 192 outputs at exactly the same time. But even if the Jack period sizes are the same, the period boundaries on each machine are of course not synchronised, and this has to be taken into account in the processing code. This is discussed in section 4.1.

The MADI connections from the rendering machines back to the master can be used to monitor and measure the signals generated by the rendering machines. They have proven to be very useful for verifying the correct operation of the signal processing algorithms running on the rendering machines, but are not used during normal operation.

3.1 Using network connections for audio

At the time the system was designed (end of 2007 and early 2008), the use of a dedicated gigabit network to transfer audio between the master and rendering machines was considered, but in the end this solution was rejected for several reasons.

- It would require some solution to ensure that audio signals would remain synchronised. This was actually a minor problem. By providing audio feedback paths for the synchronisation signal described in section 4.1 from the renderers back to the master the delays could be easily measured and automatically adjusted to be equal.
- While the capacity of a gigabit ethernet would be sufficient for the amount of data to be transferred, there were some serious doubts regarding the performance of the network adaptor drivers in a real-time system.
- Using network connections for the audio links would not permit significant savings in the audio hardware. The MADI cards and ADI648 units for the renderers would still be required, as would be those for the master machine (which has to provide a multichannel audio access point). So the only item saved would be the MADI matrix, and that is actually one of the less expensive units in the system.

In the best case the use of network connections for audio signals would have increased the latency of the system by an unpredictable

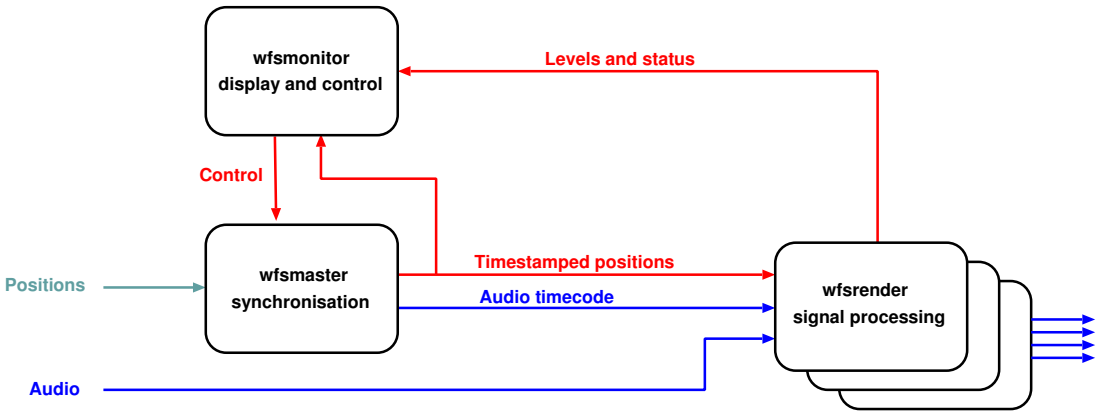

Figure 3: WFS processing structure

amount. The risk of doing this was deemed to be too high at the time. Today the picture could look different, but at least the third point mentioned above remains valid - there would not be any significant savings.

4 The WFS processing architecture

Figure 3 shows the basic WFS processing system which consists of 3 applications which communicate with each other using network messages. All of these command and monitoring messages use the OSC format.

The main function of the **wfsmaster** is to provide a single access point for controlling the WFS rendering system. Apart from generating the synchronisation signal (discussed in section 4.1), this program does not perform any audio processing. It receives commands from external programs specifying the position and movements of each virtual source (more on this in section 5), and transforms these in the format required by the rendering engines.

One instance of **wfsrender** runs on each of the rendering machines. It performs all the signal processing required for the set of speakers controlled by its host. Its inputs are the audio signals - one for each virtual source - from the MADI interface, and the processing parameters transmitted by the wfsmaster. The details of the DSP algorithms used are discussed in section 6. It also transmits monitoring messages containing e.g. the levels of all its output signals along with its internal state and any errors.

The **wfsmonitor** is the only program having a graphical user interface. It shows the current position of all virtual sources, the levels of all 189 speaker outputs, and error and status info from the two others. It can also be used to 'solo' a single speaker, or to send signals directly to a speaker for testing and alignment. There can be any number of instances of *wfsmonitor* - it can be run e.g. on portable PC inside the Sala Bianca, but the system will also run perfectly without it.

4.1 Synchronisation issues

As mentioned already in section 3 the basic audio architecture ensures that all outputs of the rendering machines will be exactly synchronised. If the system would provide only *static* virtual sources that would be all that is required. But to support *moving* sources also the application of the position parameters must be synchronised to sample accuracy on all outputs.

With its standard configuration *wfsmaster* will update and transmit all the source positions every 1024 frames. This is synchronised to (but independent of) the Jack period on the master machine. To allow the renderers to apply this information to the correct audio samples a separate synchronisation signal is generated by the master, and transmitted to the renderers following the same path as the source audio signals. It is similar to the one used by *jack_delay* to measure latency, and consists of a mix of six sine tones with frequencies that are chosen such that their phases encode the current position in a sequence that repeats exactly every 2^{20} samples (around 22s). So every sample has an implicit timestamp in the range $0 \ldots 2^{20} - 1$ that can be recovered easily by any application receiving the signal.

The timestamp corresponding to the first frame of the current position update period is included in the messages transmitted by the master program. A fixed offset (currently 800

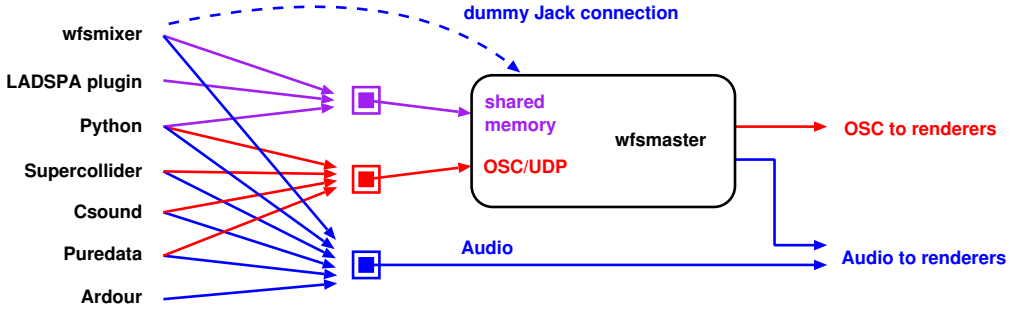

Figure 4: External application interfaces

frames) is added to allow for the worst case expected network latency. The rendering applications use the timestamp and the decoded synchronisation signal to re-align audio and data. They also check the arrival time and integrity of the position messages and the presence of a valid audio time code, and report these in their monitoring messages. An occasional 'late' position message is reported but ignored, as is a short interruption of the time code. Consistent errors will make the renderers mute their output until the situation returns to normal.

4.2 The structure of wfsrender

The *wfsrender* application is the most complicated one in the system. It has two basic functions: compute the internal WFS processing parameters in function of source position and movement, and perform the actual DSP work.

In order to make this application more easily adaptable it uses two plugins to perform most of this work. Both are loaded at runtime (as specified by a global configuration file), and can be replaced without recompiling the whole application.

The **layout** plugin defines the complete geometry of the installation. It provides methods to e.g. find out the exact position of each speaker, to which channel on which host it is connected, etc. It also performs some specific calculations, such as determining if a given x, y coordinate is internal or external and finding its distance to the line of speakers.

The **engine** plugin performs the computation of all required internal parameters and the actual DSP work, i.e. it defines the WFS algorithm used.

The interfaces to these plugins are defined as abstract C++ classes. The 'host' program takes care of the audio and network interfaces, OSC encoding and decoding, the synchronisation signal, status reporting etc. In principle any C++ programmer who understands the required algorithms could create the plugins without being distracted by the system level aspects.

Another feature added recently is multithreaded audio processing in order to use SMP systems more efficiently (the machines used are dual-core). This divides the DSP work over a configurable number of threads, each of them handling a subset of the output channels. This multithreading is transparent to the plugins.

5 External application interfaces

The system described so far just implements the basic WFS algorithm for up to 48 moving sources. To create content, spatialisation, room simulation etc. it depends on external applications supplying both the audio and source position and/or movement information. Figure 4 shows some possibilities.

Audio signals can originate or be processed on the master machine which has almost no CPU load due the WFS system itself. External sources can be connected via MADI, ADAT, or analog inputs. In all cases the signals to be used as WFS sources are just sent to the first 48 outputs of the MADI interface. Time code and test signals typically use channels 49-56, while the last eight channels are normally used for control room monitoring or for providing external analog signals (e.g. for driving subwoofers).

The *wfsmaster* program has two interfaces for source position data.

The **shared memory** interface can be used locally only, but permits sample-accurate control if the position data is generated by a Jack client. A dummy Jack connection to the master program will ensure correct order of execution in that case. This interface is also used by

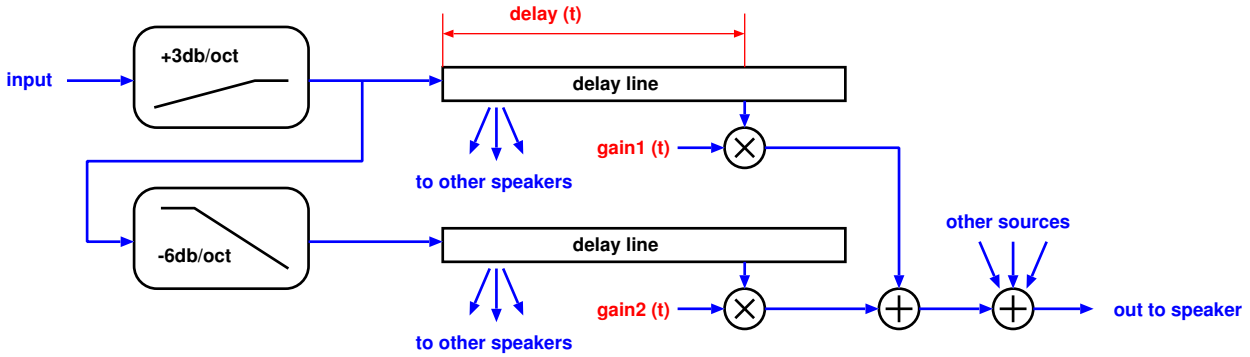

Figure 5: The basic DSP operation

some LADSPA plugins that allow source positions and movements to be encoded as automation data in Ardour. The *wfsmixer* application provides a graphical tool to control source positions. It also allows basic room simulation using Ambisonic convolution reverbs.

The **OSC/UDP** interface can be used from all the standard synthesis programs, from Python (scripts or interactive), or other languages and applications, and from any PC on the local network. The position commands allow a source to move at a controlled speed, with the master program interpolating the movement if it takes longer than one update period.

Both interfaces also include a per-source gain factor which is applied at the end of the DSP processing. This is necessary to still allow full level for a source positioned at a large distance, for example when simulating virtual speaker sets for Ambisonics.

6 The DSP algorithms

The processing required for WFS is not really very complex, but it has to be repeated $N_{in} \times N_{out}$ times, or around half that number for a 'surround' layout such as the Sala Bianca.

6.1 The basic real-time processing

The form used in the *wfsrender* application is based on equation (29) in Sascha Spor's very practical summary of WFS theory [Spors et al., 2008], which shows the driving function for a spherical wave reproduced by a linear array. This translates in to the diagram shown in fig. 5. For each single source, and for each speaker reproducing that source, the two filtered signals are delayed by a time *delay* and then combined using gain factors *gain*1 and *gain*2. Using two delay lines allows the second filter to be shared for all outputs, as are the delay lines themselves. The three parameters are computed from the source position and the system geometry. The *wfsrender* code performs this computation at every position update (i.e. every 1024 frames) and interpolates them linearly in between. Some optimisations are applied in the actual code for cases where one or more of the three parameters remain fixed.

Since the system allows for moving sources and uses the the actual delays corresponding to the postion of a primary source (which means it will reproduce the Doppler effect exactly), it can't handle the case of plane waves, as these would correspond to infinite distance and delay. It would be easy to add these as a special case, but so far there has been no need to do this.

6.2 Handling sources near the speakers

The equation cited above is an approximation that is valid only for primary sources that are not too close to the line of speakers. This is because its derivation assumes a continuous distribution of the secondary sources. Since the system supports arbitraty movements and virtual sources can cross the line of speakers without restriction these cases have to be handled separately.

The exact solution becomes quite complex, and it can't be derived as a limit case of the standard equation. To find it one has to go back to the full three-dimensional case, apply the secondary source quantisation there, and then reduce the result to a linear array.

The *wfsrender* code uses a pragmatic approach to handle this situation. If a primary source comes too close to the secondary ones, two parameter sets are computed: one for a source at a fixed distance (for which the normal equation is still valid) behind the speakers,

and one for the same distance in front. The two parameter sets are then combined according to the actual source position. The result is an approximation of the exact solution, but works well in practice.

7 System organisation and use

All source code, scripts, data files etc. required to install the system are kept on an NFS share available on all machines. A single click on a desktop menu will install or update all necessary components system-wide. Compilation is always on the target machines, i.e. everything gets installed from source.

Binaries, plugins and most scripts are kept in the standard places under /usr/local. The only exceptions would be experimental versions that would be installed per-user.

The basic setup of *wfsmaster, wfsrender* and *wfsmonitor* is defined in a global configuration file. 'Technical' users would normally just run *wfsmonitor* which then permits to run and control the others without having to use remote logins, set up Jack servers, etc. They would then add anything else they require on the master machine manually. Alternatively the Python components described below provide an easy way to create sessions that need to be recreated automatically many times, and have in fact become the preferred way to use the system.

7.1 Automatic configuration

For the 'museum' user the situation is quite different. A single click on a desktop icon launches the complete system under the control of a Python program that configures everything and that acts as a server to a remote control application. The remote control looks like a typical desktop audio player, with stop/start/loop buttons, volume control, a playlist etc. It runs locally on the master machine and also on one or more 'EEE' notebooks with wireless network access, used by the museum staff. Selecting an item from the playlist reconfigures the complete system according to the requirements for that item.

Apart from the components that implement the remote control server and the playlist, the Python program has classes for all required audio applications, each new instance of them becoming a separate Jack client. These include *py_jackctl, py_jplayer, py_ambdec, py_jconvolver* and some others. More will be added in the future.

8 Acknowledgements

I'd like to thank Prof. Angelo. Farina (University of Parma) and the direction of La Casa della Musica for having provided the opportunity to create the system described in this paper. A warm thanks also to Stefano Cantadori and to all my former colleagues at Audio Link and AIDA.

The realisation of this project would not have been possible without the work of all the developers who have created Linux and its audio system. An uncountable number of people on the Linux Audio mailing lists have provided valuable and often essential help and hints. My sincere thanks go to all of them.

References

Marije Baalman, Torben Hohn, Simon Schampijer, and Thilo Koch. 2007. Renewed architecture of the swonder software for wave field synthesis on large scale systems. In *Proceedings of the 6th Linux Audio Convention*, Berlin, Germany.

Marije Baalman. 2008. *On Wave Field Synthesis and Electro-acoustic Music.* VDM Verlag Dr. Mueller AG, Saarbruecken, Germany.

Sacha Spors, Rudolf Rabenstein, and Jens Ahrens. 2008. The theory of wave field synthesis revisited. In *Proceedings of the 124th AES Convention*, Amsterdam, The Netherlands. Audio Engineering Society.

Edwin Verheyen. 1998. *Sound Reproduction by Wave Field Synthesis.* Technische Universiteit Delft, Delft, The Netherlands. Doctoral thesis.