The IEM Demosuite, a large-scale jukebox for the MUMUTH concert venue

Peter PLESSAS and **IOhannes m ZMÖLNIG**
Institute of Electronic Music and Acoustics (IEM), University of Music and Performing Arts (KUG)
Inffeldgasse 10/III
8010 Graz,
Austria
{plessas, zmoelnig}@iem.at

Abstract

In order to present the manifold possibilities of surround sound design in the new MUMUTH concert hall to a broad audience we developed an easy to use application running and interacting with audio demonstration scenes from a mobile computing device. An extensible number of such demonstrations using the large-scale and variable loudspeaker hemisphere in the hall are controlled with a user interface optimized for touch-sensitive input. Strategies for improving operating safety, as well as the introduction of a client/server model using the programming language Pure Data, are discussed in connection with a 3D Ambisonics rendering server, both implemented using the Linux operating system.

Keywords

demonstration, interaction, client-server systems, Ambisonics, Pure Data

1 The Mumuth

The University of Music and Performing Arts Graz (KUG) opened a new building, the *Music and Music Theatre House* (MUMUTH), in 2009. It holds a 600 sqm. concert hall, named in honor of the Austrian composer György Ligeti. This new venue is used for productions by the university's different departments. Therefore, the hall has to suit music performances of different styles as well as theatre and opera works. This multi-faceted usage pattern demands a flexible performance space in terms of stage machinery, room acoustics, sound reinforcement, lighting and seating. As consequence MUMUTH has no fixed stage and audience position but offers an array of pedestals making up the hall's floor and which can be individually raised to provide an elevated audience platform or stages of variable size and height. While this flexible layout allows for creative staging of different performances it also imposes a serious challenge for the installation of a sound reinforcement equipment, since any part of the room can now become the stage or audience area and as such be of varying size.

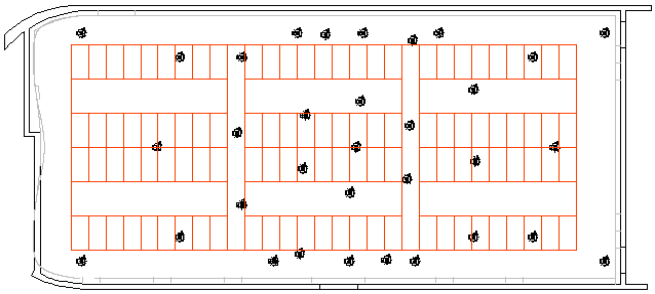

Figure 1: MUMUTH top view: Floor pedestals and loudspeaker hang points

1.1 Variable position loudspeaker hemisphere

The university's Institute of Electronic Music and Acoustics (IEM) designed a unique sound reinforcement system in order to cope with the many different demands. It consists of 33 loudspeakers, which can be lowered from the ceiling on telescopic lifts and be positioned at arbitrary heights as well as rotation and elevation angles. Figure 2 gives an impression of the room. The speaker layouts, which can be easily refined within seconds, are stored and recalled, if desired also dynamically, during a show. Electroacoustic measurements and listening tests with different loudspeaker positions permit instantaneously perceivable comparisons. It is this very setup that is used in 3D sound field rendering using higher order ambisonics from IEM's own software. It is also employed in the reproduction of the different audio scenes in the IEM Demosuite.

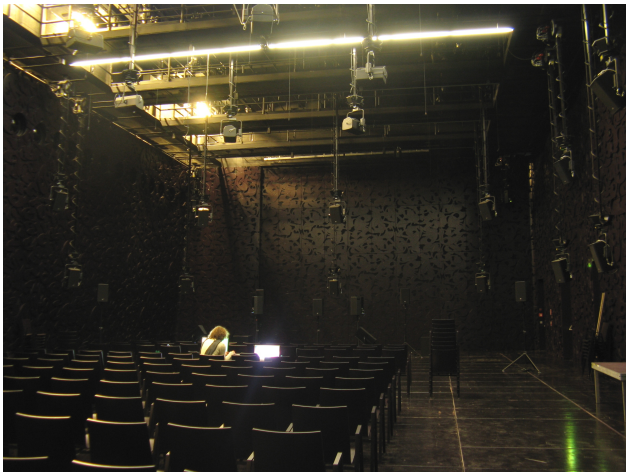

Figure 2: Loudspeaker hemisphere

1.2 The integration of higher-order Ambisonics

With many years of experience in the design and development of Ambisonics capture and playback systems, IEM developed the scalable CUBEmixer ambisonics authoring and rendering system [Musil et al., 2008], which is distributed under a GPL license[1]. This system consists of a mixing application that can run on general purpose hardware and operating systems, and which is open to user extention and external control being implemented in the Pure Data (Pd) programming language. CUBEmixer is employed in the discrete or Ambisonic spatialization of an arbitrary number of sources on the loudspeaker hemisphere. The helical positions of the mount points on the ceiling as shown in Figure 1 allow to arrange the loudspeakers in a hemisphere encircling the venue's available space with a minimum angular difference between the speaker positions. CUBEmixer integrates very well with the MUMUTH audio infrastructure, consisting of a *Lawo HD Core* mixing engine and signal matrix, controlled through the *Lawo mc*^{2}66 fanless mixing console. Inputs and outputs are via dedicated interface racks interconnected via MADI optical cables, providing a total of 4096 physical input and output channels. Interfacing to the main rendering computer is done via two *RME HDSP MADI* PCIe cards using Alsa drivers with 64 bit *Debian/GNU Linux*.

[1] http://ambisonics.iem.at/xchange/products/cubemixer

1.3 Enhanced room acoustics

In addition to the sound reinforcement system described above, MUMUTH's concert hall is equipped with a variable room-acoustics system[2], allowing to switch between different reverberation scenarios enhancing reverb envelopment and speech intelligibility. With this system the venue's acoustics can be tailored to different performance styles such as theatre, classical music, jazz and to IEM's concerts of electronic music. As the variable room-acoustics system is optimized for sound sources located on the floor level, it is in general not used together with the loudspeaker hemisphere.

2 Presenting MUMUTH's sonic abilities to a broad audience

Having build a venue the size of MUMUTH, it is important to convey the idea of a concert hall being "a musical instrument on its own" to the general public. In order to demonstrate how sound in space can be creatively composed as well as interactively explored, we decided to create a suite of music demos that would immediately catch the listener's attention and present the abilities of this new location in an entertaining and convincing way. An easy-to-use and interactive demo application for visitors, playable even while the room may be set up for a different production, had to be developed. While this suite of demos would provide ready-to-run sound scenes, it should at the same time invite pre-rendered or interactive extensions to its repertoire playlist from other artists and researchers alike.

3 Software and Hardware

While looking for a suitable software platform in which to implement the Demosuite, we decided it would be preferable to use an environment that future contributors would most likely be familiar with. In our research we and our colleagues often use Pure Data, so taking that environment as the starting point for implementing the Demosuite seemed a good choice: in theory, software taken out of our development department could be transferred directly to the suite. An additional bonus was the graphical nature of Pd, as there was no need to switch between environments

[2] *Meyersound Constellation*

when creating the visual user-interface and the DSP side of a demo. Finally, the cross-platform nature of Pd would make it possible to develop demos on one's own preferred platform, before deploying it on a stable and low-latency operating system in the venue.

The MUMUTH had already been equipped with a PC for realtime signal processing running *Debian GNU/Linux*, so we decided to use that machine as a stable and well-tested base platform that was already well integrated into the existing audio infrastructure.

This computer is usually controlled from the MUMUTH's separate sound recording studio, though it is possible to extend keyboard/monitor/mouse into the concert hall.

However, for the needs of the Demosuite, this approach was discarded as it would involve the need to set up a workstation in the concert hall every time the demo should be presented, which might in fact happen spontaneously between two rehearsals of an opera without advance warning.

Instead we decided to control this computer with a convenient and portable remote control the presenter could carry around while running the demos. While the environment of our choice (Pd) is known to run well on both iOS and Android based smartphone and tablet devices [Brinkmann et al., 2011], this is unfortunately only true for the DSP-part of Pd and not for the user interface requiring Tcl/Tk. The only available touch device we found that could run the Pd-GUI was the *Neophonie WePad*. This tablet device runs a stock i386 based Linux system[3], making it very easy to develop new and deploy already existing software on it.

In order to make Pd suitable for tablet use, its visual appearance was slightly adapted with a "kiosk mode" gui plug-in [zmoelnig, 2011] displaying a single patch window in full-screen mode without any menus.

4 Demosuite Software Design

The Demosuite software consists of two semantically different parts:

- a static *framework* that takes care of selecting and activating a demo and that is controlling some global parameters such as master volume, all optimized for touch sensitive user interfaces.
- a growing number of *demos*, which are interactive audio applications implemented as Pd patches.

[3] *Linux Foundation's MeeGo*

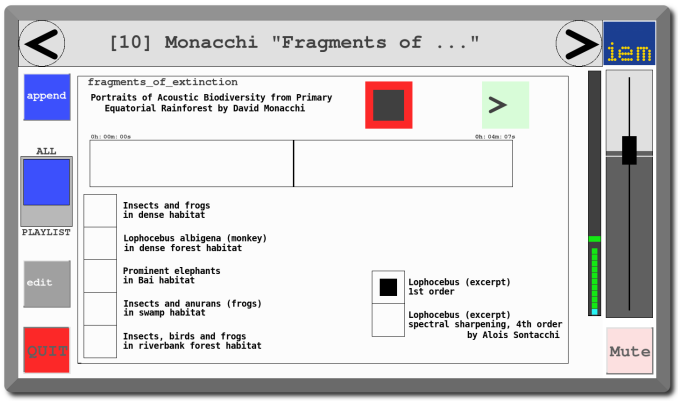

Figure 3: Example demo GUI with global volume fader right and demo selector strip on top

Demos are mutually exclusive, meaning that if a given demo is currently loaded/running, no other demo can be loaded/running at the same time. An example of a user interface for a selected demo, controlling the playback of pre-rendered sound files allowing directly switchable different Ambisonics orders is shown in Figure 3. Here, playback of pre-rendered sound files is controlled along with switchable Ambisonics orders for direct comparison, with additional written comments about the demo's contents.

To complicate things a bit, the Demosuite software is to run on two hosts in parallel:

- a powerful *DSP* server, capable of processing and interfacing multichannel sound fields in realtime
- a lightweight *GUI* client that controls this server over a network connection

The Demosuite software has to provide the communication between the DSP and the GUI parts and ensure that the internal states of the two parts stay synchronized even during network disruptions. In order to add a new demo to the playlist, corresponding GUI and DSP patches have to be copied to the two machines respectively.

4.1 Slot Management

We call the container mechanism, that ensures that the matching parts (GUI and DSP patches) of a single demo are active and synchronized across the two computers, the "slot".
Basically two different approaches to managing a variable number of demos via this slot mechanism exist:

- either all demos are preloaded into a matching number of slots and, upon selection of a given demo its containing slot gets activated while all other slots are deactivated
- or there is only a single slot that manages the synchronous loading and activation of an arbitrary demo across the two hosts.

Both methods have their advantages and drawbacks. In the case of the MUMUTH Demosuite we identified the following relevant characteristics of the "pre-load" technique:

- *Pros*
  - fast (since initialization can be done at startup time)
- *Cons*
  - more resources needed (scales badly)
  - "deactivated" demos can remain half-active (e.g. by continuously generating messages)
  - co-existing demos can influence each other (if not carefully implemented)

Whereas with the technique of "dynamic loading into a single slot" can be characterized as follows:

- *Pros*
  - resources allocated on demand (scales well)
  - "deactivated" demos don't exist and can therefore not take any resources
  - demos never co-exist
- *Cons*
  - resources allocated on the fly (slow, since no pre-loading is possible)
  - instantiation locks the Pd process
  - requires dynamic patching which makes code less readable (bad for long term maintenance)

Since the demos are to be implemented by multiple collaborators and will presumably be of diverse code quality, the perfect isolation between demos in the "dynamic loading" approach was the winning argument. Assuming that a single demo will consume moderate resources in return, the elongated load time due to dynamic resource allocation was considered acceptable. Furthermore, this approach scales well even for a very large number of demos.

4.2 Slot Interface

A slot consists of two separate programs (the DSP and the GUI part) that communicate with each other as well as produce perceptible output (the DSP will usually render sound, whereas the GUI will visualize an interface to control the former). The communication channels for the two parts are standardized, in order to be able to globally replace them with different networking techniques.

4.2.1 control communication

Usually GUI and DSP will run on different computers that are linked via a network connection. To keep the implementation simple, this communication is abstracted as unidirectional and asynchronous busses. The network message interface inside PD is implemented as single `[inlet]` resp. `[outlet]` objects in the GUI and DSP patch of each demo. Each Pd message that is to be sent from one peer to the other, has to go through that single `[outlet]` and will then appear on the other side's sole `[inlet]` object. No assumption must be made on the speed of the transmission.

4.2.2 audible and visible output

In addition to choosing and interacting with a demo, the user should be able to control the global volume with a single fader, in a unified way for each and every demo. It is therefore important that the DSP patch does not access the soundcard's DACs (or rather Pd's representation thereof) directly, but through a global audio bus. The Demosuite applies a few post-processing steps on this multi channel audio bus, namely master volume, multichannels limiting and a global mute. This is achieved by using

a special [throw~ chn-<N>] object rather than Pd's own [dac~ <N>].
The GUI part is trying to make best use of the allocated screen estate by being implemented as a Pd "graph on parent" interface of a fixed size, defining the available area for the UI part of a demo.[4]

4.3 Loading a Demo

Since the GUI and DSP parts are asynchronous processes, it is necessary to pay special attention to the sequence of actions in which to load a demo in order to guarantee consistent behavior and defined states of both systems at each instant in time.
Loading is initiated by the user selecting a scene (e.g. "foobar") on the interface. The GUI will then delete the previously loaded demo, so it can no longer send control messages to the DSP part of the demo, and enter a transitional state, (showing a "loading..." splash screen). Once this is accomplished, a request "/playlist_entry foobar" is sent to the DSP part.
Once DSP receives such a request, it will first delete its part of the the previous scene and subsequently load the "foobar/dsp.pd" patch, which receives an initialization trigger. Once the DSP part of the demo is initialized, it must then send a "/dsp_init_done" message, which will cause the DSP to send a "/create_gui_slot foobar" request back to the GUI. Now that the GUI knows that the DSP is properly loaded, it can load and display the "foobar/gui.pd" patch. The GUI part of demos having non-trivial internal states can now query the DSP to transmit the current state of internal parameters.

5 Communication between GUI and DSP

Since the DSP and the GUI are running on two different machines, their communication is done via a network connection. In MUMUTH, the tablet is connected via a wireless LAN to ensure maximum mobility of the presenter. This required the following considerations.

[4]This is probably the most tedious part when writing new demos. Unfortunately, at the time of this writing, Pd does not provide a usable auto-scaling function of "graph on parent" interfaces, which means that whenever the screen resolution of the remote control changes, the GUI patches have to be adapted...

5.1 TCP/IP vs. UDP

Traditionally, network connections involving realtime audio environments use the simple UDP rather than the more reliable TCP/IP protocol, mainly because of the following two reasons:

- *overhead*: due to it's simpler nature UDP has less traffic overhead than TCP/IP, and will therefore consume less resources
- *robustness*: since UDP does not track active connections, it handles loss of connectivity more gracefully than TCP/IP; esp. it does not cause the remote point to hang if the local point experiences a network outage. This is especially important considering the volatile connection with a wireless mobile device and the non-threaded network implementation in Pd (that would cause the main audio thread to hang in case the TCP/IP remote control left the coverage of the wireless LAN access point!)

5.2 to OSC or not to OSC...

State of the art communication between networked audio workstations is usually implemented using Open Sound Control (OSC). While Pd still has no built-in support for OSC and instead provides objects that speak PD's own FUDI protocol, it can be extended using Martin Peach's "osc" library. Unfortunately we found the various available network transport implementations needed for this "osc" library to be less stable than expected, especially in cases where the connection can easily drop, which is expected to happen once the tablet leaves the W-LAN coverage. Rather than fixing these implementations, we decided to write a set of abstractions that hide the actual implementation of the OSC transport.
On the application layer directly accessible within a demo (and within most parts of the framework), "OSC-style" messages are used. On the layer below, these messages are currently transmitted within FUDI containers. Once the issues of the network transport libraries are addressed with the ongoing development of Pd's externals, the wrapper objects can easily be switched back to use OSC containers.

5.3 Network Security

In the current implementation, absolutely no network security is built into the system. An intruder

who has knowledge of the protocol, the network address and port of the DSP machine can send arbitrary commands. Since the wireless network is only available within the concert hall and is encrypted, we have not experienced any security related problems so far. In any case, it should be easy enough to protect the two hosts via a clever set of firewall rules and network tunnelling technology such as ssh, SSL or VPN.

5.4 Operating Safety

Whenever the connection between the main DSP computer machine and the GUI remote control is interrupted, it is necessary to fallback into a safe state. In order to monitor the status of the UDP connection, a special "heartbeat" message is constantly exchanged between the two computers. Whenever the user interface assumes to be connected to the DSP server, it sends out a message "/client_alive 1" in regular intervals (currently about 3 times per second), which is answered by a "/server_alive 1" message in return. If either side does not receive its peer's "alive" message for a certain amount of time, it considers the remote side to have disconnected. Once the DSP server detects a disconnected GUI, it will immediately mute its audio outputs in order to prevent damage to the listener's ears as well as to the equipment by uncontrollable audio signals. Whenever the GUI detects a disconnected DSP server, it displays a warning message of the lost connection and locks the interface that otherwise would control the DSP part of a demo. This is last measure is important in order to ensure that the state on both DSP and GUI sides divert as little as possible, making the reconnection and resuming of a demo an easy task. The GUI will try to reestablish the connection by periodically attempting to reconnect, send a "/conn_request 1" message. This message will eventually trigger an reconnection attempt of the DSP server.
As soon as the connection is re-established, the GUI queries the DSP server about it's current state and adjusts its internal state and hence the state of the user controls accordingly.

6 Repertoire

Current examples of demos available in the suite are:

- Audio scenes of pre-rendered Ambisonic sources on different spatialisation paths on the hemisphere with dynamically changing early reflections and late reverberation, with Ambisonics-encoded reverb return signals.
- Ambisonic sources in comparison to discrete loudspeaker playback with additional depth perception cues.
- Extracts from IEM concert productions.
- Dedicated pieces from IEM staff and featured guest composers.
- A 3D panning demo using allowing interactive user control.
- Ambisonics field recording examples.

7 Summary

In this contribution we described how an easy-to-use system of audio demo scenes for the interactive exploration of the new MUMUTH concert space has been realised. We presented an approach towards networked Pd patches using connection state monitoring together with dynamic patch creation on different host computers. A solution allowing for easy extensibility of the demo repertoire while preserving system stability was discussed along with an alternative network transport layer for Pd messages. The adaption of Pd to touch-based tablet interfaces and considerations regarding operational reliability with regard to high-volume PA systems were shown.

8 Acknowledgements

While the design and realisation of the audio infrastructure in MUMUTH has been the result of a huge team of collaborators, the authors would especially like to express their gratitude to IEM/KUG members Gerhard Eckel for initiating the Demosuite project, and Thomas Musil, Stefan Warum and Ulrich Gladisch for their exceptional support.

References

Peter Brinkmann, Peter Kirn, Richard Lawler, Chris McCormick, Martin Roth, and Hans-Christoph Steiner. 2011. Embedding pure data with libpd. In *Proceedings of the Pure Data Convention 2011*, Weimar, Germany. `http://www.uni-weimar.de/medien/wiki/images/Embedding_Pure_Data_with_libpd.pdf`.

Thomas Musil, Winfried Ritsch, and Johannes Zmölnig. 2008. The cubemixer a performance-, mixing- and masteringtool. In *Proceedings of the Linux Audio Conference 2008*, Cologne, Germany. `http://old.iem.at/projekte/publications/paper/cm/cm.pdf`.

IOhannes zmoelnig. 2011. New clothes for pure data. In *Proceedings of the Linux Audio Conference 2011*, Maynooth, Ireland. `http://lac.linuxaudio.org/2011/download/lac2011_proceedings.pdf`.