Sound Server Round-up

for GU4DEC (June 16th-18th, 2003, Dublin)

by Ronald Bultje <rbultje@ronald.bitfreak.net>
May 12th, 2003
Contents

1. Introduction ............................................................................................. p. 3
2. Status quo – Enlightened Sound Daemon (ESD) ................................. p. 4
3. The other side – Analog Realtime Synthesizer (aRts) ......................... p. 6
4. X taking responsibility – Media Application Server (MAS) ................ p. 8
5. Straight from the professionals – Jack Audio Connection Kit ............ p. 10
6. Other products available ......................................................................... p. 11
8. Conclusion & Discussion ....................................................................... p. 13
9. References ............................................................................................ p. 14
Introduction

Since the beginning of the Gnome era back in the previous millennium, the official gnome sound server has been the Enlightened Sound Daemon. Back then, this choice was not hard to make, since esd was the only product available. There was also the Analog Realtime Synthesizer (aRts), which was evilly cooperating with the dark group of people called 'KDE' and had some evil dependencies, too.

Back to here, 2003, nothing has changed. KDE is still using aRts and Gnome is still using ESD. From the KDE community, there is an oncoming critical movement that wants to replace the aRts sound server and media framework with a modern, integrated, more functional alternative. We will come back to this in the appropriate chapter.

The Gnome movement, too, is not undoubtedly supportive to their own sound server of choice. A growing group of people criticizes the dead ESD project for not moving on with current market/desktop needs, not keeping up with current Gnome standards, etcetera. This is not a strange thing: projects pop up, grow and die on the road leading to Total World Domination™. ESD seems to have gone into the last step. More on this in the ESD chapter.

The main point of this piece of history is that there is a turnaround going on in the current line of thinking. Both of the major desktop projects are ready to move over to something totally new. The best step to take from here would be a cooperative decision to support one sound server together (maybe officially supported by freedesktop.org?), which would lead to a huge improvement for end users. No more clashes between KDE- and Gnome-apps concerning “which is started first” to see which one actually gets access to sound cards. Unified efforts in improving one sound server rather than in maintaining one – badly. Or do we want to leave the decision to the user and create a generic infrastructure in which every sound server can be plugged – GStreamer?

The choice is here and now: are we ready for something new? And is the new stuff ready for us? If so, what will it be?
Status quo – Enlightened Sound Daemon (ESD)

The Enlightened Sound Daemon (ESD) was adopted as official Gnome sound server back in the Gnome-1.0 days, or probably even before. The last real update (anything more than required maintainance) comes from 1999. On the official ESD pages, there's two news items from 1999 and 2000, both stating that the developer is very busy. This normally means the project is dead, but – fortunately – there's some Gnome maintainance going on. Gnome releases bugfix releases for every new major desktop release.

How it works

ESD is fairly easy to use. It allows a developer to open a socket (`esd_play_stream () and `esd_record_stream () to the sound daemon and use standard posix functions (`write (), `read () and `close ()) to get data from it or play audio data or close the connection. The server mixes incoming streams and writes it to the audio device or diverts the data from the device over to the ESD clients, transparent to the application. ESD also has a somewhat higher-level API that allows one-call file playback (audiofile-based).

What it can do

• ESD is network-transparent.
• It allows for sample uploading (to 'cache' sound samples and actually play them later on, without having to reload the audio data to the sound server).
• It handles format conversion, if needed by the soundcard.
• If the soundcard supports hardware mixing, ESD uses that instead of software mixing.
• Supports a huge amount of kernel sound subsystems, including OSS, ALSA, Solaris, AIX, IRIX, Tru64 (OSF), HP-UX, OS/2 (DART) and Windows (please note that some of these have never been tested).
• Provides a utility (esddsp) that lets applications that use the sound device directly be rerouted to use ESD.

... and what it cannot

• ESD provides no information about when the samples are done playing, or how far they are.
• There is no possibility to synchronize several streams – this means that an application that opens multiple connections to the ESD server cannot guarantee in any way that samples written over the socket at the same time will also be played at the same time – together.
• DMA/Shm is not possible with ESD, because the data first has to be written over a socket (also locally) before it can be written to the sound card.
• ESD's mixing routines are suboptimal.
• Sound sometimes crackles because of lack of real-time processing capabilities.
• Server always runs in stereo in 16-bit mode at a samplerate of 44.1kHz. This is problematic because there's a fair chance that all input is in another format. This is suboptimal.
• Practically only supports two channels and only 8/16 bit audio. There is currently sound cards that can do more than two channels, and 8/16 bit audio might also be deprecated some day.
• A device can only be selected in the server. The client API does not allow for device selection.

**Future development**

There is currently no ongoing development – the project is basically dead apart from the usual maintainance fixes from the Gnome community.
The other side – Analog Realtime Synthesizer (aRts)

aRts is the official sound server of the KDE project. Actually, it is much more than a sound server; it tries to position itself as a media framework (similar to GStreamer), most of this is still in alpha, though. Although it can do more than being a plain sound server, this part will mainly focus on the sound-server side of aRts. Firstly, because that is mostly the only part that has ever been included in a stable release. Secondly, because that is all this article cares about. Just like ESD, aRts is toolkit-independent. It uses the MCOP (Midibus Corba Protocol) for communication between server and client, though, which is a KDE-only thing (and which is deprecated in favour of DCOP in KDE).

How it works

aRts basically works fairly similar to ESD. A socket with the sound server is opened with \texttt{arts\_play\_stream ()} and \texttt{arts\_record\_stream ()}. Data is read from or written to the sound server using \texttt{arts\_write ()} and \texttt{arts\_read ()}. The connection is closed using \texttt{arts\_close\_stream ()}. The server mixes incoming streams and writes it to the audio device or diverts the data from the device over to the aRts clients, transparent to the application. Also, just like ESD, aRts has a somewhat higher-level API that allows one-call file playback (audiofile-based).

What it can do

- aRts is network-transparent.
- It handles format conversion, if needed by the soundcard.
- Provides a utility (artsdsp) that lets applications that use the sound device directly be rerouted to use aRts.
- It uses CORBA.

... and what it lacks

- Support for any kernel sound subsystem other than OSS.
- aRts provides no information about when the samples are done playing, or how far they are.
- Just like with ESD, there is no possibility to synchronize several streams. An application that opens multiple connections to the aRts server cannot guarantee in any way that samples written over the socket at the same time will also be played at the same time – together.
- DMA/Shm is not possible with aRts, for the same reason as for ESD: all data goes through a socket first.
- Sound sometimes crackles because of lack of real-time processing capabilities.
- A device can only be selected in the server. The client API does not allow for device
Future Development

According to their website, the developers are busy redesigning aRts to have more media framework-like capabilities, while maintaining the sound server capabilities. There hasn't been any official release providing all this, though, just bits and pieces. Video support seems to be rather hackishly. It can be seen as a nice start, but will need lots of more work before it is actually technically at the same level as competitors as GStreamer.

Part of the effort seems to focus on a new tool called CSL (Common Sound Layer), which is basically a new sound server altogether, where aRts focusses more and more on being a media framework. Since there has not been wide use of CSL yet, it is not taken into account largely in this overview.

Even though there are a lot of interesting things going on in the aRts camp, KDE does not seem to be undividedly happy about all this. Some of the multimedia programmers have stepped over to GStreamer for their media support. The status of aRts as part of the KDE desktop is under discussion.
**X taking responsibility – Media Application Server (MAS)**

In 1999, the Media Application Server (MAS) sound server project was started – funded by X.org and their partners. MAS tries to present audio processing in a similar way as the X protocol, integrated with the X-server functionalities. This provides yet-unseen features such as hardware-level video/audio synchronization.

MAS has had their latest stable release in the beginning of 2003. The project claims to be ready for prime-time, which means wide-ranged use under the two desktop projects, KDE and Gnome. Integration in both Gnome and KDE have been discussed before.

**How it works**

The MAS library is initialized by `mas_init ()`. After this, an audio channel and a server device have to be selected by calling `mas_make_data_channel ()` and `mas_asm_instantiate_device ()`. After this, format connection between the client and server takes place. This means that the client requests the server playback format, and then has to create a list of filters (rate/bps/channel conversion routines) in order to make the local stream properties match the server properties. In all cases, the application has to do that itself! An output or input device can then be selected by `mas_asm_connect_source_sink ()`. Data can be sent using `mas_send ()` or received using `mas_recv_package ()`.

The MAS team has actually admitted that this API is fairly hard to use. They are currently working on a higher-level API that should work like the ESD API, but we have not yet seen a final version of this.

**What it can do**

- MAS is network-transparent.
- Setup encoded network streaming, which lowers the network data traffic compared to raw data transfer.
- It integrates with X.
- It provides hardware-level synchronization between audio and video (well, actually, there are plans for this and it is not terribly hard to implement in the environment. It is not actually there).
- Portable codebase.
- Allows for synchronization of multiple sources.
- Provides a full server-side plugin interface.
- Protocol-based rather than library-based (just like the X protocol).

2. [http://lists.kde.org/?l=kde-multimedia&m=104669966423967&w=2](http://lists.kde.org/?l=kde-multimedia&m=104669966423967&w=2)
**Where it fails...**

- The setup between the server and client is horribly complicated, especially compared to ESD and aRts. There are plans to fix this, but there is no actual API or code yet.
- It is fairly bloated in the features it provides. It shares several features with products like GStreamer, which are already part of the Gnome desktop.
- Only has an OSS device output plugin.
- No DMA/Shm.

**Future Development**

X.org wants to profile MAS as the primary sound server for X desktops. Although this sounds like a nice idea, MAS is not there yet. The client interface needs some serious simplification.

For a desktop, MAS might just provide too much. There is a lot of work going on to make things network-transparent, but the local speedups aren't built as an effect of that. This is comparable to XFree86 without XShm.

However, for any form of perfect multimedia handling, you need hardware-level synchronization. MAS is the only sound server that is able to provide that as of now.
Straight from the professionals – Jack Audio Connection Kit

The answer from the “professional linux audio people” to the currently described sound server mess was called “Jack Audio Connection Kit”, mostly pronounced as just Jack. The Jack sound server has a totally different setup from other sound servers, in that it is callback based rather than provision-based. The sound server library requests samples from the client rather than the client providing samples to the server.

The Jack project was founded after discussion on a sound server API for professional audio editing (the proposal was known as the Linux Audio Applications Glue API, in short LAAGA\(^3\)). According to the creators, it allows for low-latency, high-bandwidth audio throughput with client synchronization, something not provided by any of the other sound servers out there.

**How it works**

A connection with the sound server is opened by `jack_client_new ()`. After that, the client has to set a processing callback that will be called whenever the sound server thinks something should be done. This is done by `jack_set_process_callback ()`. For each recording/playback session, the client has to open an input or output port using `jack_port_register ()`, and connect it to the sound server using `jack_connect ()`. Processing is started and stopped with `jack_activate ()` and `jack_deactivate ()`. The input/output port is closed using `jack_port_unregister ()`. The client closes down with `jack_client_close ()`.

**What it can do**

- Low latency.
- Allows synchronization between streams.
- Designed by the professional Linux audio community – known to suit their needs for professional applications.
- Allows synchronization with extern (video) sources.

**... why even jack is not perfect**

- only ALSA and solaris support. No OSS!
- Requires a high-quality sound-card. Works badly with low-end soundcards.
- Complicated, non-intuitive callback-based API.
- No network transparency.

\(^3\) [http://eca.cx/laaga/](http://eca.cx/laaga/)
Other products available

These are only a few of the whole lot of widely available sound servers, some only in experimental design phase, others reality already. Examples include NAS, Gnome-Streamer, CSL, and there is probably a full list of candidates on Freshmeat.

All of these have special features and are worth looking at. This was omitted simply because of a lack of time in writing this paper.
A totally different approach – GStreamer plugging

Time for something completely different. Why do we need a sound server at all? Let the user decide! This idea was first mentioned on the Gnome mailinglist, where it was proposed to make a thin layer where sound servers can be plugged in. The idea was quickly rejected because it would take too much effort and would start a track of “making everything configurable”, something less-userfriendly than one would think.

However, that layer already exists! With one Gconf entry, sound outputs (direct device access, sound servers, or anything pluggable in GStreamer) can be switched in Gstreamer-applications reading this key (Rhythmbox, GStreamer Media Player). This same mechanism could be used for Gnome as a whole, too. This would provide the Gnome desktop with a pluggable sound-server mechanism without having to write one any new code. This code would probably have to be written to provide a thin layer over GStreamer in libgnome, but that would not be much work.

Why this is a good idea

• Configurable sound server selection.
• No new code or plugin mechanism has to be written. Existing interfaces can be used instead.
• Support for 4 sound servers already (Jack, MAS, ESD, aRts).
• Optional direct device access (ALSA, OSS).

... and why it should not be done (yet)

• GStreamer has not proven to be rock stable yet (requirement for core gnome dependency).
• GStreamers API/ABI is not rock stable either (for a period of X years).
Conclusion & Discussion

The question that everyone is waiting for is simple: which sound server should Gnome use? That question is not easy to answer. Surely, Jack and MAS offer some fantastic advantages over the current choices, mostly meant for professional audio or the video/audio combination. The thing is, however, that both Jack and MAS miss some functionality that Gnome will probably need. MAS is too hard, Jack does not do OSS. Both are essential in order to be successful. So the Gnome sound server will probably stay ESD for some more time, unless some of the above problems are fixed quickly. Fortunately, Gnome-3.0 (where this question would be most appropriate) is a long time away, so is KDE-4.0. So there is still some time left for both to fix themselves up.

The other choice, using GStreamer as a sound server plugin system, is interesting, although it would add an unstable dependency to libgnome. This is probably not a solution either, for now.

In short, Gnome will probably use ESD for some more time.

Interesting questions are whether MAS or Jack would be willing (and able) to fix their problems. For both, some complaints have been listed in their respective chapters; would fixing these be good enough to consider them as a replacement for ESD in Gnome (and maybe KDE)? And what about the proposed GStreamer solution? Is this type of solution (wrapping rather than choosing) wanted at all? These are discussions for Gnome to consider.
References

- **One year aRts – a status report**, Stefan Westerfeld <stefan@space.twc.de>. See http://www.arts-project.org/doc/1styear/index.html.
- **MAS Client API specifications**, See http://www.mediaapplicationserver.net/mas-api-0.6.0.pdf.