Advanced Gtk+ Sequencer

Joël Krähemann
http://nongnu.org/gsequencer
Amriswil
Switzerland
jkraehemann@gmail.com

Abstract

The Advanced Gtk+ Sequencer libraries are introduced and explained. Whereas GSequencer is an application providing a frontend to it. The entire code is written in GNU C99. Libags1 provides a ready to use tic based system as well hierarchical processing structures. The parallel computed audio tree, provides one nesting level allowing you to have recycled audio data. This tree supporting wormhole like accesses, reduces much copying overhead. You learn how GSequencer uses libags-audio1 to do additive mixing of audio data organized in a tree structure, provides envelope information to builtin step-sequencers or notation based playback. To use the libraries in your own application, it is recommended to know about GObject.

Keywords

Audio processing, parallelism, ANSI C99.

1 Introduction

Advanced Gtk+ Sequencer provides an audio processing thread per output line of an instrument. This can lead to massive concurrency as combining multiple instruments and bundling their lines by a mixer. The libraries make use of conditional-locks to orchestrate work within well defined semantics. Thread-safety is guaranteed by the core library that makes use of a tic-based system. So called tasks might be called exclusively within time-line. Those can be added async non-blocking to the task thread. There is a message delivery system available to notify about events asynchronously.
Libags-audio1 distinguishes between in-/output channels. They can be mapped 1:1 or 1:n. The nested recycling tree, makes reusing audio data easy. The API is well populated and offers for most work a function. You have to know setting AgsAudio related recycling flags and ags_channel_set_link() function, all remainig work is solved by the library.
GSequencer relays on the data model of libags-audio1 and does no audio processing on its own. The strict separation of GUI and audio layer makes the code reusable. The entry point to the framework is the AgsApplicationContext object.
The core library supports global configuration and commonly used utility and helper functions. Further it provides interface abstraction as AgsSoundcard, AgsSequencer and others. Most of the additional marshallers used by GSequencer are located in it, as well.
AgsTurtle object maps Terse RDF Triple files to XML. You can lookup content using XPath as usual. The AgsTurtleManager keeps track of your loaded turtle files.

2 The AgsConcurrencyProvider interface

GObject allows you to do interface-inheritance[1], so your application context only needs to implement AgsConcurrencyProvider in order to use libags-thread.so.1. It is important to have AgsTaskThread in place because most of the audio API relays on it.

2.1 AgsThread object

The AgsThread class provides the ::run() method and shall synchronize all tics within the thread tree by the ::clock() event. E.g. you synchronize 1000 times per second all your threads ::clock() is invoked for each tic. Whereas ::run() is only called as many times as ::clock() tells you to run. This might be 0 or 1 times. Further the count of tics per second an audio thread is actually run, is calculated as following:

tps = (buffer_size / samplerate) * max_precision

2.1.1 Timestamps

AgsTimestamp supports unix timestamps or the relative counter used by libags-audio1. The timestamp can only be of one type.

2.1.2 Mutex manager

The mutex manager provides a singleton pattern to insert your mutexes and lookup afterwards. As you use certain GObject derivates which type is actually not know, the call to ags_mutex_manager_lookup() might help you out of the misere, not being able to access the mutex within its struct.

2.1.3 Intermediate pre-/post-sync

Since all threads do its work synchronized, it offers you 2 very useful flags. As setting intermediate pre-/post-sync the child thread is run either direct before or after the parent's ::run() event.

[1] It is allowed only once, no interface sub-types.

This is good as you want all sources to be read, then process its data, after you just output to the sinks.

2.1.4 AgsThreadPool

The thread pool provides already running AgsReturnableThread instances. You can connect to its ::safe-run() event, to get called during next tic.

2.1.5 Non-blocking tasks

The basic abstraction is done in libags-thread.so.1 but only libags-audio.so.1 provides a set of common tasks.
Non-blocking is only from point of the caller. Since AgsTaskThread usually resides within the multi-threaded tree, which is synced, too.
The task engine ensures, there is no race-condition and work is done conflict free.

2.1.6 Worker threads

If you have non-realtime work to get fulfilled the AgsWorkerThread provides you this context. You can even add it to your tree without interfering to any synchronization.

2.1.7 Message delivery

As Gtk+-2.0 doesn't allow you to manipulate widgets from other threads than thread #0, the AgsMessageDelivery and AgsMessageQueue was introduced. The libraries don't produce any messages until you add the appropriate message queue. You would usually poll using g_timeout_add() the message queue.

3 The AgsSoundProvider interface

To use libags-audio.so.1 you implement AgsConcurrenyProvider and AgsSoundProvider. Thought there are ready to use application contexts, you could provide your very own.
This interface provides getter and setter methods of a GList containing AgsSoundcard and AgsSequencer.

3.1 AgsAudioApplicationContext

The audio application context provides the basic entry point for more complex tasks involved using the API. There common helpers to handle audio buffers and Standard MIDI File not requiring any of it. You can even write to soundcard or export to audio file without having the need of it. This is possible because the entire API follows the principles of a modular setup. Note some objects may relay on others, certain with a tighter coupling and some rather lossy.

3.1.1 Processing audio

The hierarchical representation of your audio data can be processed with 3 different parallelism contexts. Whereas audio data is processed lock-free. This can be achieved due to dedicated audio data.
AgsRecall provides basic abstraction to process your audio data. It happens during 3 stages, each emitting an event. Those might have additional events e.g. AgsDelayAudioRun::notation-allocate-input()

Stage	Purpose
::run-pre()	Counter & buffer allocation
::run-inter()	Ordinary effect processing
::run-post()	Interpreting audio data

Table 1: Stages of audio processing

The 3 stages of processing are called continuesly in their well defined order. During one audio tic all 3 events are emitted.

3.1.2 Instantiate recalls

ags_recall_factory_create() shall do most of the work. It populates the AgsAudio and AgsChannel with the recall specified by a string to the given boundaries.
The returned GList of AgsRecall objects might have a need for assigning additional dependencies. Further you might want to set some initial values of AgsPort(s) contained by the returned recalls.

3.1.3 Custom recalls

AgsRecall is your base object to implement your own effect processor. Its class provides abstraction for most common use cases. But feel free to implement your own events to listen to.
The AgsDynamicConnectable's interface methods ::connect-dynamic() and ::disconnect-dynamic() are related to dynamic dependencies. These are dependencies requiring to be resolved. This needs to be done during AgsRecall::resolve-dependencies() event as part of the initialization process.
::run-init-pre(), ::run-init-inter() and ::run-init-post() are the 3 stages of initialization. This applies to AgsRecallAudioRun and AgsRecall-ChannelRun, which have a dynamic scope. These methods are called only once per lifetime of the recall.
The recall may provide AgsPort instances to modify the behaviour of your effect. Each port might be automated. The ::automate() signal does usually apply the automation data. It is provided by AgsRecallAudio and AgsRecallChannel

4 AgsPattern

AgsPattern is a storage class providing 2 banks and a bitfield you can toggle. The AgsPreset allows you to assign envelope information to such pattern based playback.
In GSequencer step-sequencer UIs like AgsDrum or AgsMatrix usually make use of it.

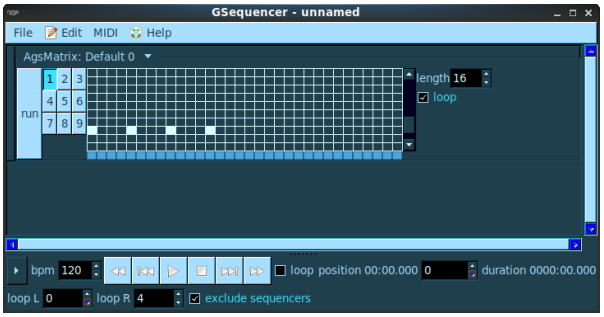

Illustration 1: AgsMatrix - screenshot

5 AgsNotation

AgsNotation is segmented and chunks are identified by its timestamp field. The notation contains AgsNote objects. Those specify the note's x-offset and x-endpoint as well the y-alignment. Every note contains envelope information that can be applied using provided recalls.
You can edit this object with a notation editor provided by GSequencer or record the data by a MIDI instrument. Assigning MIDI sequencer and channel can be performed by a dialog available from the machine's context menu.
The editor is available to all built-in instruments and DSSI or LV2 synthesizer plugins. It follows the principles of multi-channel editing, you can enable/disable the addition of an AgsNote for a certain channel.

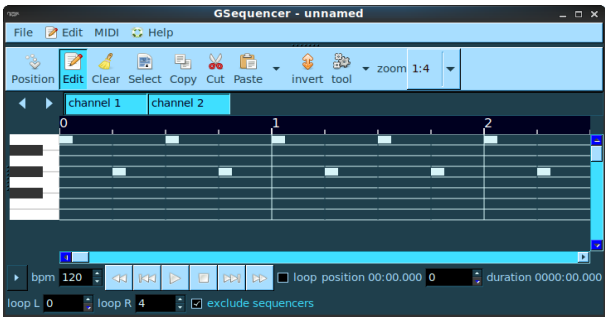

Illustration 2: notation editor - screenshot

6 AgsAutomation

AgsAutomation is segmented and chunks are identified like AgsNotation by its timestamp field. The automation contains AgsAcceleration objects. Those specify the increase of a value by its x-offset and y-value. These pairs aren't related anyhow to AgsNote fields.
AgsRecall::automate() implementations carry the current matching y-value of x-offset to the matching AgsPort. Basically all ports can be automated but some built-in ports don't export their control.
You can edit this object with an automation editor provided by GSequencer. It follows the multi-channel editing principles, too.

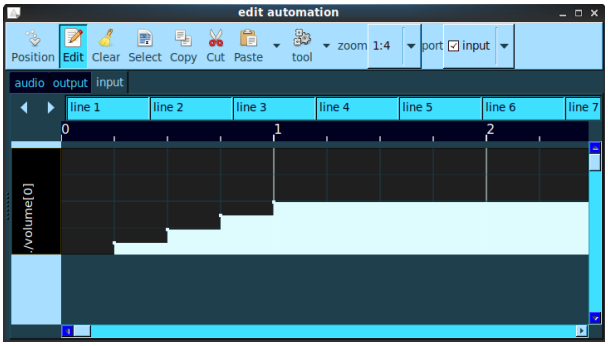

Illustration 3: automation editor - screenshot

7 Supported backends

There is support for various backends like ALSA, OSSv4, JACK Audio Connection Kit, Pulseaudio and Core-Audio.

7.1 Soundcards

AgsSoundcardThread does talk to your soundcard or "virtual" soundcard by the AgsSoundcard interface. Prior only output was supported but audio input is currently developed as part of the 1.4.x release.
The thread does intermediate post synchronization in view of the audio processing thread AgsAudioLoop. It can poll your file descriptors to schedule work.
AgsClearBuffer task clears some of the ring-buffer in order next additive mixing can do its work. This is performed by "ags-play-channel-run" or alike.
AgsSwitchBufferFlag controls the currently used part of the ring-buffer.
Note polling is done by AgsPollingThread and the library maps it to AgsPollFd object.

7.2 Export to audio files

AgsExportThread does output to an audio file. Currently supported formats are WAV, FLAC, AIFF and OGG. However libsndfile provides more formats but their suffix isn't checked, yet.
AgsAudioFile is used to export your data in realtime. FYI the buffer is only flushed as the file object is closed.

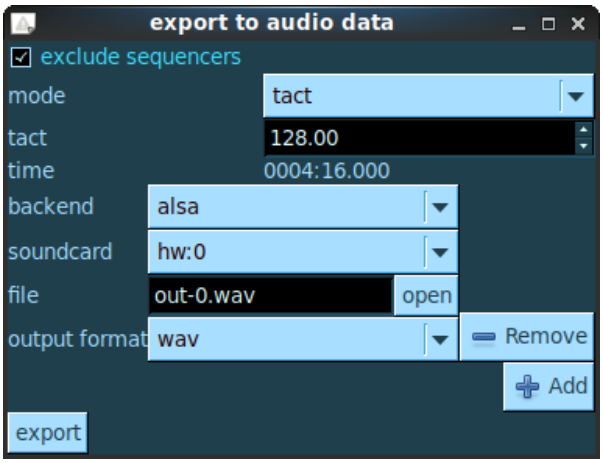

Illustration 4: export window - screenshot

7.3 MIDI instruments

AgsSequencerThread gets MIDI data from your instrument, by using AgsSequencer interface. Currently only MIDI input is supported as version 1.4.x.
The thread does intermediate pre synchronization in view of the audio processing thread AgsAudioLoop. It checks for new bytes to be read and provides it as ring-buffer.
AgsClearBuffer task clears some of the ring-buffer in order next bytes can be provided. Them are used by "ags-record-midi" recall. The MIDI note-on and note-off messages are interpreted to AgsNote objects.

AgsSwitchBufferFlag controls the currently used part of the ring-buffer.
The MIDI connection dialog allows you to specify the channel to be used by the instrument. Further you can adjust the start mapping between MIDI key and audio channel as well the range of data to be used.

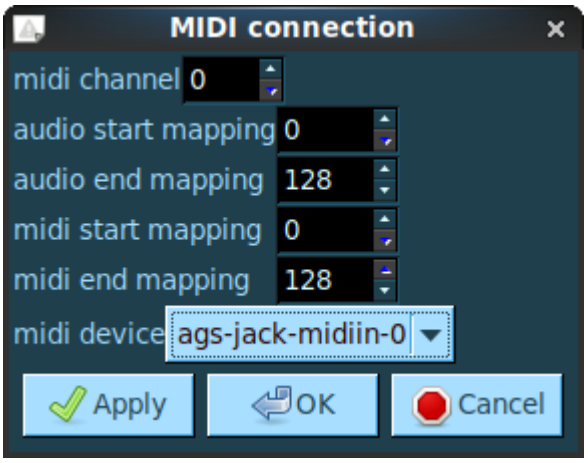

Illustration 5: MIDI connection - screenshot

8 Conclusion

Having a multi-threaded concept is only the half work. The biggest problem is to strict follow the lock strategy. The big amount of source code and the time it takes to develop an audio sequencer, gives some bad taste. Recently some key components of the library routines have been refactored. It is probably not the last code rewrite the application has seen.
Most annoying during development was the lack of good documentation. However the Linux audio community was there for help. It was not always evident howto interact, it just goes on. My appologies for being an attitude man should be mentioned. I am sorry.
The importance of unit-tests just showed during these refactorings. Unlike the current coverage, future development brings defenitely more tests.
A new era of GSequencer has just begun. As much of the work is already done. The proceeding development is taken with more patience.
There are still some wished features not implemented, yet. Further some optimizations and real performance improvements could be achieved. The main focus was set on concurrency, what gives the engine much power. But it still does dynamic memory allocation during audio processing.

9 Acknowledgements

My thanks go to Felix Rabe, friends and Linux Audio Developers on irc.freenode.net. The amazing private social network, showing their interests. Finally my family never gave me up. Special thanks to my mother.

Illustration 6: about – screenshot

References

[1] GObject-2.0 API Reference Manual, https://developer.gnome.org/gobject/stable/
[2] Libxml-2.0 API Reference Manual, http://xmlsoft.org/html/book1.html
[3] Libinstpatch-1.0 API Reference Manual, http://www.swamiproject.org/api/libinstpatch/
[4] Libasound-4.0 API Reference Manual, https://www.alsa-project.org/alsa-doc/alsa-lib/
[5] OSSv4 API Reference Manual, http://manuals.opensound.com/developer/
[6] RDF 1.1 Turtle, https://www.w3.org/TR/turtle/
[7] Gtk-2.0 API Reference Manual, https://developer.gnome.org/gtk2/stable/
[8] JACK API Reference Manual, http://jackaudio.org/files/docs/html/index.html
[9] Pulseaudio API Reference Manual, https://freedesktop.org/software/pulseaudio/doxygen/
[10] Cairo Graphics API Reference Manual, https://www.cairographics.org/manual/
[11] GNU Libc Manual, https://www.gnu.org/software/libc/manual/
[12] Libsndfile-1.0 API Reference Manual, http://www.mega-nerd.com/libsndfile/api.html
[13] Libsamplerate API Reference Manual, http://www.mega-nerd.com/SRC/api_misc.html
[14] LADSPA API Reference Manual, http://www.ladspa.org/ladspa_sdk/overview.html
[15] DSSI API Reference Manual, http://dssi.sourceforge.net/RFC.html
[16] LV2 API Reference Manual, http://lv2plug.in/doc/html/
[17] The MIDI Specification, https://www.midi.org/specifications/item/the-midi-1-0-specification
[18] Randal E. Bryant and David R. O'Hallaron, Pearson; Computer Systems – A Programmer's Perspective; ISBN: 0-13-713336-7
[19] Curtis Roads, MIT; the computer music tutorial; ISBN: 0-252-18158-4