The MusE Sequencer: Current Features and Plans for the Future

Werner SCHWEER
Ludgerweg 5
33442 Clarholz-Herzebrock, Germany
ws@seh.de

Frank NEUMANN
Bärenweg 26
76149 Karlsruhe, Germany
beachnase@web.de

Abstract

The MusE MIDI/Audio Sequencer[1] has been around in the Linux world for several years now, gaining more and more momentum. Having been a one-man project for a long time, it has slowly attracted several developers who have been given cvs write access and continuously help to improve and extend MusE. This paper briefly explains the current feature set, gives some insight into the historical development of MusE, continues with some design decisions made during its evolution, and lists planned changes and extensions.

Keywords

MIDI, Audio, Sequencer, JACK, ALSA

1 Introduction

MusE is a MIDI/Audio sequencer which somewhat resembles the look 'n' feel and functionality of software like Cubase for Windows. It is based on the concepts of (unlimited) tracks and parts, understands both internal and external MIDI clients (through the ALSA[2] driver framework) and handles different types of tracks: MIDI, drum and audio tracks. For audio, it provides a built-in mixer with insert effects, subgroups and a master out section and allows to send the downmix to a soundcard, a new audio track or to a file.

Through support of the LADSPA[3] standard, a large amount of free and open effect plugins are available to be used on audio tracks.

By being based on the JACK audio framework, it is possible to both route other program's sound output into MusE and route MusE's output to other programs, for instance a mastering application like Jamin[4].

MusE's built-in editors for MIDI/audio data come close to the average PC application with expected operations like selection, cut/copy/paste, Drag&Drop, and more. Unlimited Undo/Redo help in avoiding dataloss through accidental keypresses by your cat.

2 Historical Rundown

MusE has been developed by german software developer Werner Schweer since roughly January 2000. Early developments started even years before that; first as a raw Xlib program, later using the Tcl/Tk scripting language because it provided the most usable API and nicest look at that time (mid-90s). It was able to load and display musical notes in a pianoroll-like display, and could play out notes to a MIDI device through a hand-crafted kernel module that allowed somewhat timing-stable playback by using the kernel's timers. MIDI data was then handled to the raw MIDI device of OSS. As the amount of data to be moved is rather small in the MIDI domain, reasonable timing could be reached back then even without such modern features as "realtime-lsm" or "realtime-preempt" patches that we have today in 2.6 kernels.

With a growing codebase, the code quickly became too hard to maintain, so the switch to another programming language was unavoidable. Since that rewrite, MusE is developed entirely in C++, employing the Qt[5] user interface toolkit by Trolltech, and several smaller libraries for housekeeping tasks (libsndfile[6], JACK[7]).

In its early form, MusE was a MIDI only sequencer, depending on the Open Sound System (OSS) by 4Front Technologies[8]. When the ALSA audio framework became stable and attractive (mid-2000), ALSA MIDI support was added, and later on also ALSA audio output. Summer 2001 saw the introduction of an important new feature, the "MESS" (MusE Experimental Soft Synth"). This allows for the development of pluggable software synthesizers, like the VSTi mechanism on Steinberg's Cubase

sequencer software for Windows.

At some point in 2003 the score editor which was so far a part of MusE was thrown out (it was not really working very well anyway) and has then been reincarnated as a new SourceForge project, named MScore[9]. Once it stabilizes, it should be able to read and export files not only in standard MIDI file format, but also in MusE's own, XML-based .med format.

In October 2003 the project page moved to a new location at SourceForge. This made maintenance of some parts (web page, cvs access, bug reporting) easier and allowed the team of active developers to grow. Since this time, MusE is undergoing a more streamlined release process with a person responsible for producing releases (Robert Jonsson) and the Linux-typical separation into a stable branch (only bug fixes here) and a development branch (new features added here).

In November 2003 the audio support was reduced to JACK only. Obviously the ALSA driver code inside JACK was more mature than the one in MusE itself, and it was also easier to grasp and better documented than the ALSA API.

Additionally, fully supporting the JACK model meant instant interoperability with other JACK applications. Finally, it had also become too much of a hassle for the main developer to maintain both audio backends. The data delivery model which looked like a "push" model from outside MusE until now (though internally it had always used a separate audio thread that pulls the audio data out of the engine and pushes it into the ALSA pcm devices) has now become a clear "pull" model, with the jackd sound server collecting audio data from MusE and all other clients connected to it, and sending that data out to either pcm playback devices or back to JACK-enabled applications.

It has been asked whether MusE can be used as a simple "MIDI only" sequencer without any audio support. The current CVS source contains some "dummy JACK code" which gets activated when starting MusE in debug mode. If the interest in this is high enough, it might get extended into a real "MIDI only" mode.

In early 2004, the user interface underwent substantial changes when Joachim Schiele joined the team to redesign a lot of pixmaps for windows, menus, buttons and other user interface elements. Finally, MusE also received the obligatory splash screen!

Another interesting development of 2004 is that it brought a couple of songs composed, recorded and mixed with MusE. This seems to indicate that after years of development work, MusE is slowly becoming "ready for the masses".

3 Examples of Coding Decisions

(1) In earlier versions of MusE, each NoteOn event had its own absolute time stamp telling when this event should be triggered during playback. When a part containing a set of elements was moved to a different time location, the time stamps of all events in this part had to be offset accordingly. However, when the concept of "clone copies" was introduced (comparable to symlinks under Linux: Several identical copies of a part exist, and modifying an event in one part modifies its instance in all other cloned copies), this posed a problem: The same dataset is used, but of course the timestamps have to be different. This resulted in a change by giving all events only a timestamp relative to the start of the part it lives in. So, by adding up the local time stamp and the offset of the part's start from the song start, a correct event playback is guaranteed for all parts and their clone copies.

(2) MIDI controller events can roughly be separated into two groups: Those connected to note events, and those decoupled from notes. The first group is covered by note on/off velocity and to some degree channel aftertouch. The second group contains the classic controllers like pitchbender, modulation wheel and more. Now, when recording several tracks of MIDI data which are all going to be played back through the same MIDI port and MIDI channel, how should recording (and later playback) of such controller events be handled? Also, when moving a part around, should the controller data accompanying it be moved too, or will this have a negative impact on another (not moved) part on another track? Cubase seems to let the user decide by asking him this, but there might be more intelligent solutions to this issue.

(3) The current development or "head" branch of the MusE development allows parameter automation in some points. How is parameter

automation handled correctly, for both MIDI and audio? Take as an example gain automation. For MIDI, when interpolating between two volumes, you do normally not want to create thousands of MIDI events for a volume ramp because this risks flooding a MIDI cable with too much data and losing exact timing on other (neighbouring) MIDI events. For audio, a much finer-grained volume ramp is possible, but again if the rate at which automation is applied (the so-called "control rate") is driven to extremes (reading out the current gain value at each audio frame, at *audio rate)*, too much overhead is created. So instead the control rate is set to a somewhat lower frequency than the audio rate. One possible solution is to go for the JACK buffer size, but this poses another problem: Different setups use different values for sample rate (44.1kHz? 48kHz? 96kHz?) or period size, which means that the same song might sound slightly different on different systems. This is an ongoing design decision, and perhaps communication with other projects will bring some insight into the matter.

4 Weak Spots

There are a couple of deficiencies in MusE; for all of these efforts are already underway to get rid of them, though:

- There is a clear lack of documentation on the whole software. This is already tackled, however, by a collaborative effort to create a manual through a Wiki-based approach[10].
- The developer base is small compared to the code base, and there is a steep learning curve for prospective new developers wishing to get up to speed with MusE's internals. However, this seems to be true for a lot of medium-sized or large Open Source projects these days. Perhaps better code commenting (e.g. in the Doxygen[11] style) would help to increase readability and understandability of the code.
- Not all parts of MusE are as stable as one would want. One of the reasons is that the focus has been more on features and architecture than on stability for quite some time, though since the advent of stable and development versions of MusE this focus has changed a bit and MusE is getting more stable.

5 Future Plans

The plans for the future are manifold - as can be seen very often with open-source projects, there are gazillions of TODO items and plans, but too little time/resources to implement them all. What follows is a list of planned features, separated into "affects users" and "affects developers". Some of these items are already well underway, while others are still high up in the clouds.

5.1 Planned changes on the User level

- Synchronisation with external MIDI devices. This is top priority on the list currently, and while some code for MIDI Time Code (MTC) is already in place, it needs heavy testing. MusE can already send out MMC (MIDI Machine Control) and MIDI Clock, but when using MusE as a slave, there is still some work to be done. There have been plans for a while in the JACK team to make the JACK transport control sample-precise, and this will certainly help once it is in place.
- A file import function for the old (0.6.x) MusE .med files. This has been requested several times (so there *are* in fact people using MusE for a while! ☺), and as all .med files (XML-based) carry a file version string inside them, it is no problem to recognize 1.0 (MusE 0.6.x) and 2.0 (0.7.x) format music files.
- Complete automation of all controllers (this includes control parameters in e.g. LADSPA plugins).
- Mapping audio controllers to MIDI controllers: This would allow using external MIDI control devices (fader boxes, e.g. from Doepfer or Behringer) to operate internal parameters and functions (transport, mixer etc).
- A feature to align the tempo map to a freely recorded musical piece (which will alter the tempo map in the master track accordingly). This would bring a very natural and "human" way of composing and recording music while still allowing to later add in more tracks, e.g. drums.
- Support for DSSI soft synths (see below)
- A configurable time axis (above the track list) whose display can be switched between "measure/beat/tick", SMPTE (minute / second / frame) and "wallclock time".
- The "MScore" program which has been separated from MusE will start to become useful for real work. It will make use of the above mentioned new libraries, AWL and AL,

and will have a simple playback function built in (no realtime playback though), employing the fluidsynth software synthesizer. It will be able to work with both .mid and .med files, with the .med file providing the richer content of the two. MScore is still lacking support for some musical score feature like triplets and n-tuplets, but this is all underway. A lot of code is already in place but again requires serious testing and debugging.

5.2 Planned changes on the Developer level

- Better modularisation. Two new libraries are forming which will become external packages at some point: "AWL" (Audio Widget Library) which provides widgets typically found in the audio domain, like meters, location indicators, grids or knobs, and "AL" (audio library) which features house-holding functions like conversion between a tempo map (MIDI ticks) and "wallclock time" (SMPTE: hh:mm:ss:ff).
- Also, MESS soft synths shall get detached from the MusE core application so that they can get built easier and with less dependencies. This reduces the steepness of the learning curve for new soft synth developers.
- The new "DSSI"[12] (Disposable SoftSynth Interface) is another desired feature. Already now the "VST" and "MESS" classes have a common base class, and adding in support for DSSI here should not be too complicated.
- The creation of a "demosong regression test suite" will be helpful in finding weak spots of the MIDI processing engine. This could address issues like MIDI files with more than 16 channels, massive amounts of controller events in a very short time frame, SysEx handling, checks for "hung notes" and more. Getting input and suggestions from the community on this topic will be very helpful!

6 Conclusions

MusE is one of the best choices to compose music under Linux when the "classical" approach (notes, bars, pattern, MIDI, samples) is required.

Thanks to the integration with the existing ALSA and JACK frameworks, it can interact with other audio applications to form a complete recording solution, from the musical idea to the final master.

7 Other Applications

There are some other MIDI/audio applications with a similar scope as MusE; some of them are:

- Rosegarden[13], a KDE-based notation software and sequencer
- Ardour[14], turning a computer into a digital audio workstation
- seq24[15], a pattern-based sequencer (MIDI-only) and live performance tool
- Cheesetracker[16], a "classic" pattern-oriented tracker application
- Beast[17], an audio sequencer with a built-in graphical modular synthesis system

Besides these, there are already numerous soft synthesizers/drum machines/samplers etc that are able to read MIDI input through ALSA and send out their audio data through JACK; these programs can be controlled from within MusE and thus extend its sound capabilities. However, connecting them with MusE MIDI- and audiowise is more complicated, and can cause more overhead due to context switches.

8 Acknowledgements

The authors wish to acknowledge the work of everyone helping in turning a Linux-based computer into a viable and free alternative to typical commercial audio systems. No matter whether you are developing, testing, documenting or supporting somehow – thank you!

References

[1] http://www.muse-sequencer.org

[2] http://www.alsa-project.org

[3] http://www.ladspa.org

[4] http://jamin.sourceforge.net

[5] http://www.trolltech.com/products/qt/index.html

[6] http://www.mega-nerd.com/libsndfile/

[7] http://jackit.sourceforge.net

[8] http://www.4front-tech.com

[9] http://mscore.sourceforge.net

[10] http://www.muse-sequencer.org/wiki/index.php/Main_Page

[11] http://www.doxygen.org

[12] http://dssi.sourceforge.net

[13] http://www.rosegardenmusic.com

[14] http://www.ardour.org

[15] http://www.filter24.org/seq24/index.html

[16] http://www.reduz.com.ar/cheesetronic

[17] http://beast.gtk.org