Musical Engineering in JamMo

Mikko Gynther
Lappeenranta University of Technology
P.O.Box 20, 53851 Lappeenranta, Finland
mikko.gynther@lut.fi

Henrik Hedberg
Department of information Processing Science,
University of Oulu
P.O.Box 3000, 90014 University of Oulu, Finland
henrik.hedberg@oulu.fi

Abstract

This paper discusses implementation of musical engineering components in the UMSIC project. The implemented components are parts of JamMo, an open source musical learning and collaboration software tool for children. JamMo is targeted at Maemo operating system and Nokia N900 mobile device. In the implementation GStreamer streaming media framework, known from media players for Linux, was utilized. Available GStreamer elements were used when applicable and more were created to meet the requirements for JamMo. All GStreamer elements were controlled and accessed through implemented classes. The implemented elements consisted of virtual instruments, a metronome and a pitch detector. These elements utilized sound synthesis and analysis. GStreamer framework was found effective in creating complex audio pipelines and suitable for creating new kinds of audio elements beyond its most common context.

Index Terms: UMSIC, JamMo, Maemo, Nokia N900, GStreamer, sound synthesis, sound analysis

I. Introduction

UMSIC is a multidisciplinary and transnational science and technology project that started in 2008 in will end in 2011. It aims at measuring and increasing social inclusion of children by the means of musical collaboration. In the UMSIC project software called JamMo is developed. JamMo is a music collaboration and learning tool intended at children aged from 3 to 12. Children aged from 3 to 6 can play a singing game and a simple loop composition game. Children aged from 7 to 12 can also play instruments and use recorded singing or playing with instrument performances as parts of compositions.

The JamMo application is targeted at Maemo [8] operating system and Nokia N900 [11] mobile device. Thus, a suitable musical engineering framework for mobile use had to be implemented. It was decided to be built upon existing multimedia libraries to reduce the amount of work, and because Maemo platform has them available. The specific needs of multi-track sequencer, unique instruments and simultaneous audio playing and recording were implemented during the project.

This paper answers to the question *what is the most suitable way to implement a sequencer framework based on existing libraries in mobile environment*. It is based on the experiences we got during the project.

II. Background

JamMo is written in C language and released under GPL 2 license. Some of its modules utilize GObject [2] model from GLib [1] which is a cross-platform software utility library. Although, JamMo is targeted at Nokia N900 [11] and Maemo [8] operating system it can easily be built and run on other Linux distributions as well [6]. JamMo source code can be downloaded at Gitorious project hosting service and compiled binaries are available for Maemo and Ubuntu Linux [5].

Maemo is a Linux-based operating system for Nokia Internet Tablets. It is derived from Debian GNU/Linux which is a popular Linux distribution. The current version is Maemo 5 and it is run on Nokia N900 which is the latest addition to Nokia Internet Tablet product line. Lately, Nokia has started to collaborate with Intel, and thus Maemo and Intel's Moblin project have been combined to form MeeGo platform [10].

For music programs Maemo had several application programming interfaces (API): GStreamer, ESound, ALSA user space and PulseAudio [9]. GStreamer [3] and PulseAudio [12] were suitable for creating new software as the others were legacy APIs that use PulseAudio wrappers [9].

PulseAudio is an advanced sound server. It acts between applications and hardware. PulseAudio offers a raw audio API for programmers. In Maemo PulseAudio uses ALSA kernel space drivers for device access [9] but on other platforms it may use other libraries as well.

GStreamer is a streaming multimedia framework. GStreamer applications construct pipelines of elements. Pipelines may contain any kind of streams but most common are audio and video. Elements do the actual processing when they are linked to each other. They also hide complexity from the application programmer. Elements are supplied in plugin packages. They can also be programmed by deriving from GStreamer element classes. Applications that utilize GStreamer framework are typically media players, although other kind of multimedia applications using GStreamer have been created. GStreamer audio applications typically use PulseAudio for sound output on Maemo.

III. Musical engineering framework

GStreamer API was selected for music programming in JamMo because it included many needed functionalities. Raw audio input and output, file input and output, codecs for various audio formats, audio data conversions, filtering, effects and mixing were already available as GStreamer elements. In addition there was a need for virtual instruments, a metronome and pitch detection as this kind of elements were not available or they were unsuitable for JamMo. The missing functions were implemented as new GStreamer elements. Both readily available and implemented GStreamer elements were designed to be controlled and accessed through the GObject classes of Musical Engineering and Authoring Module (MEAM) component of JamMo. In the following chapters architecture of MEAM and implemented GStreamer elements are described.

IV. Architecture

In JamMo application multiple audio tracks are played simultaneously and synchronously. In addition, optional synchronous audio recording is supported. Simultaneous playback was implemented by creating a top-level JammoSquencer class and various track classes (see Figure 1). JammoSequencer stores and controls the tracks. The subclasses of JammoTrack class store and control GStreamer elements that are needed for a particular type of sound production.

JammoSequencer controls elements that mix the output of multiple tracks together and forward the mixed audio stream to either PulseAudio (or another sound system on a different platform) for playback or an ogg audio file for storage. It has functions for creating a GStreamer pipeline from GStreamer elements of tracks and API for playing, pausing, stopping, seeking and changing between audio production and file creation modes.

JammoTrack is an interface for all track classes. It provides a single track interface for JammoSequencer and thus, new track classes can be created without altering JammoSequencer. The interface consists of functions for getting the duration of a track, GStreamer element setup and configuration and setting tempo and pitch. Actual tracks are all derived from JammoTrack class and presented in Table 1.

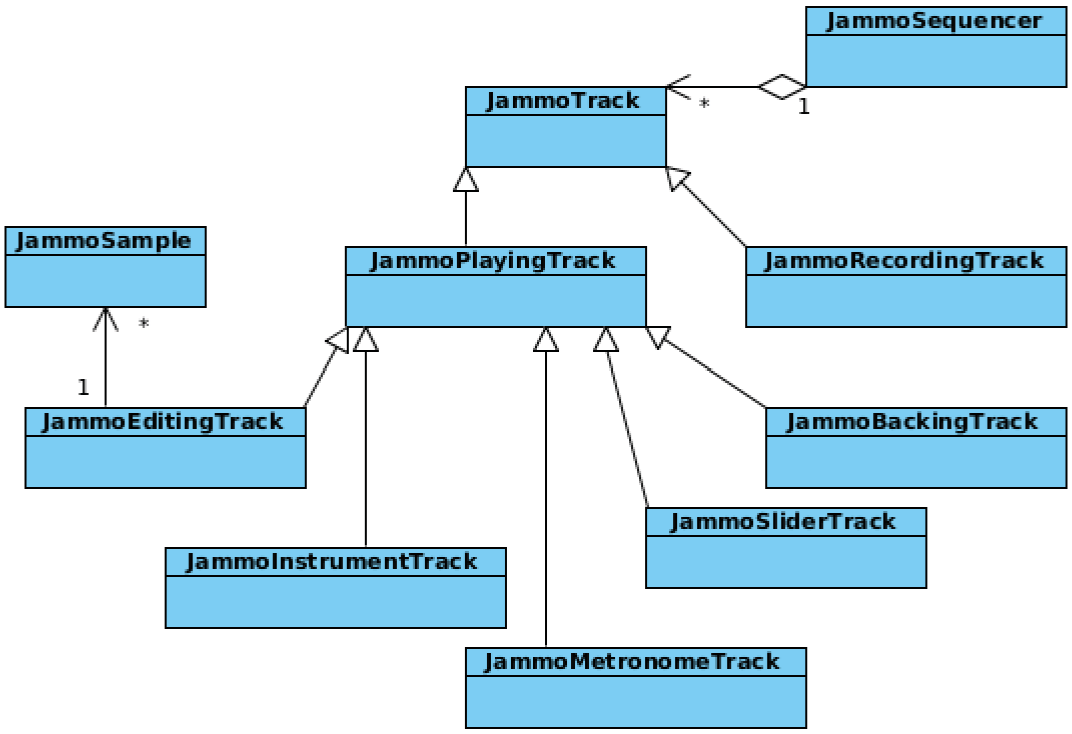

Figure 1: MEAM Classes

V. IMPLEMENTED GSTREAMER ELEMENTS

Four JammoTrack subclasses required implementation of new GStreamer elements (see Figure 2). Required functionalities consist of virtual instruments, a metronome and pitch detection.

Virtual instruments were implemented as two separate elements, GstJammoSampler and GstJammoSlider. GstJammoSampler uses sampling synthesis whereas GstJammoSlider utilizes various means to produce audio without samples. Both were derived from GstBaseSrc class and can be used in real-time and playback modes. In real-time mode a performance is recorded and the sound is simultaneously played back. Playback mode reproduces a previously recorded performance.

Table 1: JammoTrack Subclasses

Track Class	**Description**
JammoRecordingTrack	Records singing of a child to a file.
JammoPlayingTrack	Base class for tracks that produce sound.
JammoBackingTrack	Plays a single accompaniment audio track.
JammoEditingTrack	Controls many audio loops that are played sequentially. Audio loops are stored in JammoSample class.
JammoInstrumentTrack	Controls a virtual instrument.
JammoSliderTrack	Controls a virtual instrument.
JammoMetronomeTrack	Controls the metronome.

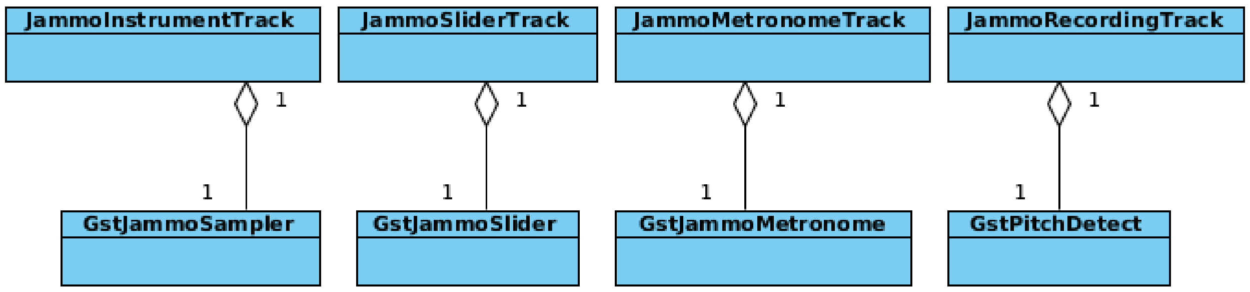

Figure 2: Implemented GStreamer elements

GstJammoSampler element is a multi tonal, single timbre sampler which can be operated in three modes to simulate different kinds of instruments. In looping mode sample files are looped to produce sounds that may last for a shorter, equal or longer time than the original sample file, e.g. bowed string instruments and wind instruments. In non-looping mode sounds last for a shorter or equal time as the sample file, e.g. plucked string instruments. In one-shot mode a sample is always played in its full duration, e.g. many percussion instruments. Sampler is configured with a text file which contains names of sample files, range of notes each sample is supposed to produce, loop start and end points in number of samples, length of sample in number of samples, crossfade length in number of samples and fade out length in number of samples.

In producing multiple different pitched notes from a single sample file GstJammoSampler uses resampling and interpolation. To ensure that looping produces as little audible artifacts as possible crossfading was implemented. GstJammoSampler uses JamMo Midi format in real-time input and storing of events. JammoMidi is a subset of MIDI standard omitting unnecessary features and changing notations for simplicity. It consists of a note value similar to MIDI, a type (note-on and note-off) and a timestamp, which is GStreamer time instead of event delta time of MIDI file format. Velocity was omitted to simplify the musical materials of JamMo. Moreover, other unnecessary MIDI messages such control changes, channel mode and system mode messages were omitted. Instead of binary data streams JammoMidi is used in C language structures to simplify implementation.

GstJammoSlider element is a single tone, single timbre synthesizer. It can produce slides and slurs not typical to Western music. This is achieved by rounding user input frequencies immediately to 24-tone equal temperament (TET). Sustained notes are rounded to 12-TET (standard tuning system in Western music) in order to suit the other audio material in JamMo. GstJammoSlider does not use samples to produce the tones of instruments because long slides would have been problematic to produce with sound samples. Using a sound sample to generate sound that is multiple tones away from samples original pitch would have sounded unrealistic regardless of the used pitch shifting algorithm. Changing a sample in the middle of audio generation would have required an elegant algorithm for hiding the tonal differences of samples. GstJammoSlider uses JammoSliderEvent format for real-time input and storing of events. JammoSliderEvent consists of a frequency, a type (on, off and motion) and a timestamp. Motion type is needed in providing slides with speed and direction changes. Timestamp is similar to JammoMidi.

GstJammoSlider can produce three different sounds: a string instrument, a flute and an organ. Each sound is produced with a different algorithm. String utilizes Karplus-Strong algorithm which uses a simple filtered delay loop to simulate a vibrating string [7]. Initially the delay loop is fed with noise. Wavelength and thus frequency can be adjusted by changing the length of the delay loop. Flute uses frequency modulation combined with sampling synthesis. In frequency modulation one sine oscillator produces the fundamental frequency which is modulated by another sine oscillator [14]. With suitable parameters frequency modulation produces a flute like tone. White noise is added from a sample file to simulate blowing sound of a flute. Combined with a low pass filter the sound is quite flute like. Organ utilizes additive synthesis. In additive synthesis many sine components are summed together [14]. Usually the components model overtones of the fundamental frequency. All sounds are processed with a subtle tremolo effect to make them more lively. Also a volume Attack Decay Sustain Release -envelope (ADSR), which controls the loudness of produced audio over time, can be used. All sounds can be filtered with a low pass and high pass filter using an available GstAudioChebLimit element [4].

GstJammoMetronome element produces ticks according to the tempo and the time signature of a piece. It is derived from GstBaseSrc class. GstJammoMetronome uses two different pitched sine wave ticks. Higher pitched tick is used as an accent to emphasize the first beat of a bar. Accentuation can be turned on or off.

GstPitchdetect element analyses the fundamental frequency of audio data. It is derived from GstBaseTransform class which is the base class simple transform filters. Despite the parent class GstPitchDetect does not alter the data in any way. For finding the fundamental frequency GstPitchdetect uses autocorrelation algorithm [13] which is fast, simple to implement and reliable. Used wavelengths in the autocorrelation algorithm are focused on 12-TET tones and the vocal range of children to avoid excessive processing and to make GstPitchdetect element lighter. After GstPitchDetect has determined the fundamental frequency of a signal it sends a message over the GStreamer bus. The sent message can be caught by the JamMo application and used for visualization purposes.

VI. Evaluation

The selected architecture has been found successful. It is simple and extendable, but at the same time it abstracts the internal implementation details of the sequencer. Thus, it has been possible to optimize GStreamer pipelines and even change the way the sequencer is constructed during the project without altering the API. The most significant modification was a test to switch from GStreamer-based elements to use directly underlying PulseAudio interface.

The harware resources of N900 are limited, and we have had constantly optimize the audio processing. Especially loading of large audio files through a narrow I/O channel into small memory cause noticiable lagging. In additon, we have continuously collided to situations where the usual solution works in a desktop computer, but fails to provide decent results in N900. For example, the song singing with simultaneous synchronized audio input and output has been rewritten many times in order to get it work right. However, the test to remove the GStreamer framework has proved that the limitation is mainly in the hardware itself but not in the software stack.

N900 was found capable of playing four simultaneous GStreamer-based JammoInstrumentTracks, JammoSliderTracks or JammoEditingTracks along a backing track when uncompressed wav backing track and loops were used. Decoding ogg compression was found costly. Using ogg loops and backing track allowed simultaneous playback of only one JammoEditingTrack along a backing track.

VII. Conclusion

The required musical engineering functionalities in JamMo were implemented using GStreamer streaming media framework. Many functionalities were implemented by using readily available elements. Virtual instruments, a metronome and the pitch detection of audio signal required implementing new elements. Classes for controlling all GStreamer elements were created. The work presented in this paper implicates that GStreamer framework is effective in constructing complex audio pipelines and can be used beyond media player application context. As projects requirements and constraints affected implementation, suitability for selected device and environment, many issues could have been done differently on a different platform, e.g. selection of audio libraries and use of existing open source software.

REFERENCES

[1] GLib Reference Manual, http://library.gnome.org/devel/glib/, Retrieved 2010-09-16.
[2] GObject Reference Manual, http://library.gnome.org/devel/gobject/, Retrieved 2010-09-16.
[3] GStreamer, Open source multimedia framework, http://gstreamer.net/, Retrieved 2010-09-16.
[4] GStreamer Good Plugins Reference Manual http://www.gstreamer.net/data/doc/gstreamer/head/gst-plugins-good-plugins/html/index.html, Retrieved 2010-09-16.
[5] JamMo – Jamming mobile game for kids website, http://jammo.garage.maemo.org/, Retrieved 2010-09-16.
[6] JamMo Developers Manual, http://jammo.garage.maemo.org/developers_manual.html, Retrieved 2010-09-16.
[7] M. Karjalainen, V. Välimäki, T. Tolonen, “Plucked-String Models: From the Karplus-Strong Algorithm to Digital Waveguides and Beyond,” Computer Music Journal, vol. 22, no. 3, 17- 32, 1998, http://www.acoustics.hut.fi/~vpv/publications/cmj98.pdf, Retrieved 2010-09-16.
[8] Maemo, Operating system for Nokia Internet Tablets, http://maemo.org/, Retrieved 2010-09-16.
[9] Maemo 5 Developer Guide/Architecture/Multimedia Domain http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/Architecture/Top_Level_Architecture, Retrieved 2010-09-16.
[10] Meego, Operating system, http://meego.com/, Retrieved 2010-09-26.
[11] Nokia N900, InternetTablet, http://maemo.nokia.com/n900/, Retrieved 2010-09-16.
[12] PulseAudio, Sound system, http://www.pulseaudio.org/, Retrieved 2010-09-16.
[13] S. Sood, A. Krishnamurthy, “A robust on-the-fly pitch (OTFP) estimation algorithm,” MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia , 2004.
[14] T. Tolonen, V. Välimäki, M. Karjalainen, “Evaluation of Modern Sound Synthesis Methods,” Report no. 48 / Helsinki University of Technology, Department of Electrical and Communications Engineering, Laboratory of Acoustics and Audio Signal Processing. TKK, Otaniemi, 1998, http://www.acoustics.hut.fi/publications/reports/sound_synth_report.pdf, Retrieved 2010-09-16.