PUBLICATION P6

Ilmonen, Tommi. Mustajuuri - An Application and Toolkit for Interactive Audio processing. Proceedings of the 7th International Conference on Auditory Displays, pages 284–285, Helsinki, Finland, 2001.

MUSTAJUURI – AN APPLICATION AND TOOLKIT FOR INTERACTIVE AUDIO PROCESSING

Tommi Ilmonen

Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400, FIN-02015 HUT
Tommi.Ilmonen@hut.fi

ABSTRACT

Mustajuuri is a freeware application and toolkit for audio signal processing. It is designed for quick prototyping, testing and combination of audio or MIDI processing modules. Its main focus is on efficient and low-latency real-time operation. Mustajuuri offers an extremely flexible plugin architecture. By creating new plugins programmmers can extend Mustajuuri to meet new needs. The C++ API takes into account the necessary features of audio signal processing and application development: low-latency real-time audio signal processing, easy debugging, graphical and text-mode user interfaces, internationalization and portability (currently supported on IRIX and Linux, but Windows port is possible). Mustajuuri also offers developers a rich set of support libraries that contain audio-related signal processing and graphics tools. Mustajuuri is licensed under the Library GNU Public License and it is available for download from the home-page: http://www.tml.hut.fi/ tilmonen/mustajuuri/.

1. INTRODUCTION

This paper presents a modular audio application Mustajuuri. Mustajuuri has been created to act both as an audio signal processing test-bench and as an end user application. The demands for modularity, flexibility, interactive use, high-performance signal processing, low latency and platform independence have been taken into account in the core of the design. Mustajuuri is an open source effort aimed primarily at the UNIX/Linux operating environment. It is portable to Windows as well.

Mustajuuri can be used in many ways. One can use it as an interactive audio DSP engine. This is done by starting the application with graphical user interface and using it as a normal desktop application, as in figure 1. A more involved user can create new plugins for the system and simply use Mustajuuri as a host to run the plugins. This saves the user the trouble of creating from scratch all the necessary infrastructure that is needed in interactive audio applications – audio I/O, real-time considerations and user interface. Mustajuuri has also been used as a stand-alone DSP engine without the GUI. In this case it was used to run a 15-channel virtual acoustics sound system [1]. Rather than having local control it was controlled over the internet. To achieve this there was no need to modify the base code. Instead the users wrote the necessary DSP and control functionality in a few plugins.

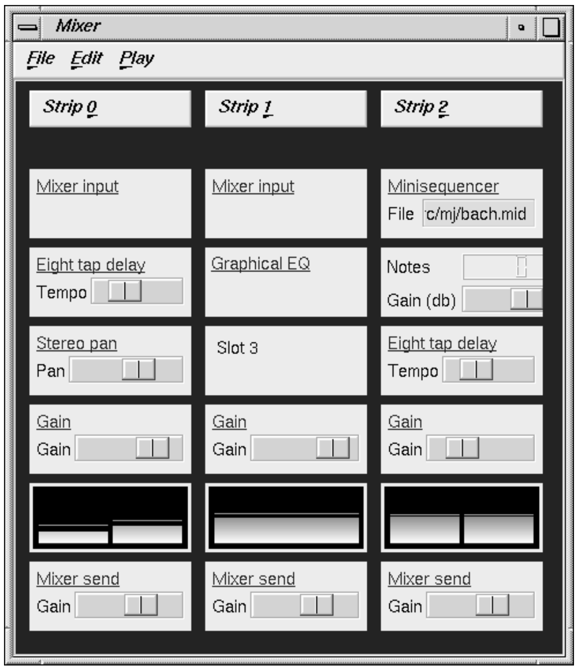


Figure 1: *A an example of a simple mixer.*

2. BACKGROUND AND RELATED WORK

There are already several audio processing libraries and applications available. The aim of Mustajuuri is not to compete with them, but provide functionality that other applications and APIs do not offer. At the same time it offers two-way compatibility with other toolkits – other toolkits can be used within Mustajuuri and other projects can use code available in Mustajuuri. It is even possible to load Mustajuuri plugins to other applications.

The venerable CSound system is the academia-standard for sound processing [2]. While it offers a collection of tools it lacks interactive elements and a graphical user interface (there have been attemps at both, but the CSound architecture has made these attemps difficult). More modern systems are the Synthesis ToolKit [3] and Sound Processing Kit [4]. These are pure DSP engines – neither of these systems have been designed for interactive use.

The Mustajuuri plugin API is not the only available plugin API for UNIX/Linux environment. A newer and much more restricted plugin API is the LADSPA (Linux Audio Developers Simple Plugin API) [5]. Mustajuuri is two-way compatible with LADSPA. Thus, Mustajuuri can use LADSPA plugins and many Mustajuuri plugins can be exported as LADSPA plugins.

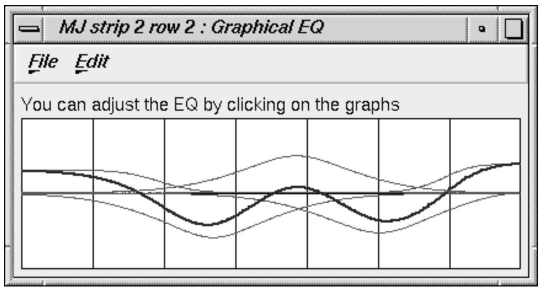
3. OVERALL DESIGN

Mustajuuri is composed of several libraries. An optimized low-level digital signal processing (DSP) library – *libmjdsp* – is supplied so that it would be easier to write new plugins. Often-needed extra widgets for the graphical user interface (GUI) are supplied in *libmjwidgets* library. More exotic extensions have their own library – *libmjutils*. The primary library – *libmj* – contains the plugin API and several support classes. The plugin architecture is very flexible. It supports audio streams (constant stream of audio signal at fixed sampling rate) and control messages (named and timestamped messages containing arbitrary data, for example MIDI).

Mustajuuri has been written with the C++ programming language. The strong typesafety checks of C++ compilers eliminate many bugs that might creep in with some other languages. It relies on the Qt class library that provides platform-independent base functionality (file input/output, XML parsing, graphical user interface, C++ template classes) [6]. Mustajuuri also uses other libraries depending on the platform; it uses the native audio device interfaces of each operating system (IRIX audio, Linux/OSS and Linux/ALSA) and the SGI audio file library (IRIX and Linux).

The plugin programming interface has been designed to help create plugins with ease. To create a new plugin type one only needs to inherit the plugin base class and extend it to match the needs. The plugin API supports both very simple and very complex plugins. An example of a simple plugin might be a monophonic low-pass filter. Another extreme would be a hard-disk recorder with any number of input and output channels.

The user interface supports multiple options as well. A plugin may have a small graphical user interface with just a slider or two. Alternatively the user can use a separate interface that allows more detailed modification of the plugin. Custom plugins can also be created for cases where it gives better usability (example in figure 2). To make users' life easier these interfaces can be open simultaneously.



You can adjust the EQ by clicking on the graphs

Figure 2: *The custom GUI of a graphical equalizer.*

The plugin API allows recursive plugins – a plugin may host other plugins. A practical example is the most commonly used top-level plugin: the mixer (illustrated in figure 1). The mixer acts as a framework for work that fits well into the classical mixer-oriented audio processing. It is possible to create other top-level plugins as well. An interesting variant would be an interface that mimics an analogue synthesizer with the wires-and-boxes approach.

The need for low latency drove us to use a threaded design. This way a high-priority DSP thread can operate without any drop-outs while the user interface thread runs the user interface.

4. APPLICATIONS

Mustajuuri has been designed to be a general-purpose system that can be customized for particular porposes. So far it has been used 1) as an effect processor for musical work, 2) as a software mixing tool, 3) as a support libary for a software synthesizer and 4) as an engine to process the sound in our virtual reality installation "EVE" [1] [7].

5. EXPERIENCE

The largest obstacle was the immaturity of Linux multimedia subsystems. Fortunately many of these problems are becoming problems of the past with the rapid development rate of Linux multimedia subsystems. Mustajuuri web site has information about suitable sound cards, Linux tweaks and installation instructions [8].

On the positive side the application structure has proved to work. Mustajuuri can operate with very low latency (less than ten milliseconds) on tuned systems. While this is already practical we are working to further lower this limit to the 1-3 milliseconds range.

The graphical user interface has been a major benefit – even for users who did not anticipate they would need it.

6. REFERENCES

[1] J. Hiipakka, T. Ilmonen, Matti. Lokki T., Gröhn, and L. Savioja, "Implementation issues of 3D audio in a virtual room," in *Proc. of 13th Symposium of IS&T/SPIE, Electronic Imaging 2001*, January 2001, vol. 4297B.

[2] Richard Boulanger, Ed., *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*, MIT Press, October 1999.

[3] Perry Cook and Gary Scavone, "The Synthesis Toolkit (STK)," in *Proceedings of the International Computer Music Conference*, Beijing, China, 1999, pp. 164–166.

[4] Kai Lassfolk, "Sound Processing Kit – An Object Oriented Signal Processing Framework," in *Proceedings of the International Computer Music Conference*, Beijing, China, 1999, pp. 422–424.

[5] LADSPA-team, "Linux Audio Developer's Simple Plugin API (LADSPA)," WWW-site, Cited 1.5.2001, URL=http://www.ladspa.org/.

[6] Troll-Tech, "The Qt class library," WWW-site, Cited 1.5.2001, URL=http://www.trolltech.com/.

[7] EVE-team, "Experimental Virtual Environment," WWW-site, Cited 1.5.2001, URL=http://eve.hut.fi/.

[8] Tommi Ilmonen, "Mustajuuri Home Page," WWW-site, Cited 1.5.2001, URL=http://www.tml.hut.fi/ tilmonen/mustajuuri/.