A Real-Time Full-Duplex Software Environment for Microphone Array Application Development

Martin Eichler and Arild Lacroix
J. W. Goethe Universität, Institut für Angewandte Physik, 60438 Frankfurt am Main
Email: {Eichler, Lacroix}@iap.uni-frankfurt.de

Abstract

Microphone arrays are widely used in applications such as beam-forming, sound field analysis, sound source localization, speaker tracking, noise cancellation and others. Prototyping such applications requires a suitable hardware and software environment capable of multi-channel, real-time, full-duplex audio data processing. In our institute, a software environment has been developed which runs on an MS Windows XP platform, allowing for easy implementation and real-time evaluation of microphone array filter algorithms.

System Requirements

The main requirements regarding the functionality of the system were defined as follows:

- Real-time signal processing
- Arbitrary number of input and output channels
- Arbitrary number of interconnectable filters

Further, all filters should be controllable/steerable via the graphical user interface (GUI) and it should be easy to implement a new filter for use with the framework. Also, recording and replay were desired as well as time-domain and frequency-domain analysis functions.

Solution

In order to realize a system that allows both flexible GUI programming and easy real-time algorithm implementation, the software architecture was split up into three coding levels, using C#.NET, C++.NET and native C++ languages ([1, 2]); each language was chosen to fulfill certain requirements (see fig. 1). Special attention was paid to the synchronisation of the various audio devices, as the Windows API maps even multichannel hardware to a set of stereo devices, such that, for example, eight of these need to be accessed and synchronized for 16-channel operation. The resulting class architecture of the real-time core is shown in fig. 2. Managed *ref classes* form the interface to the C#.NET layer while native C++ classes access the hardware through Windows APIs. One central object of class CAudioProcessorKernel synchronizes all CFilterHandler, CRealtimeDeviceHandler and CHarddiskRecordingDeviceHandler objects being used. All these are derived from CAudioDataHandler which provides CSignalSourceBuffer and CSignalSinkBuffer objects functioning as I/O channel data streams: hardware input and filter output channels are handled by CSignalSourceBuffer objects, while hardware output and filter input channels are handled by CSignalSinkBuffer objects. One signal source can be linked to multiple signal sinks, thus allowing for a very flexible signal routing between filters, I/O and recording devices.

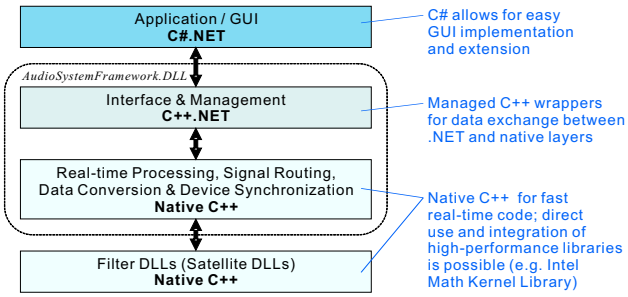

Figure 1: Coding layers (C#/C++.NET & native C++).

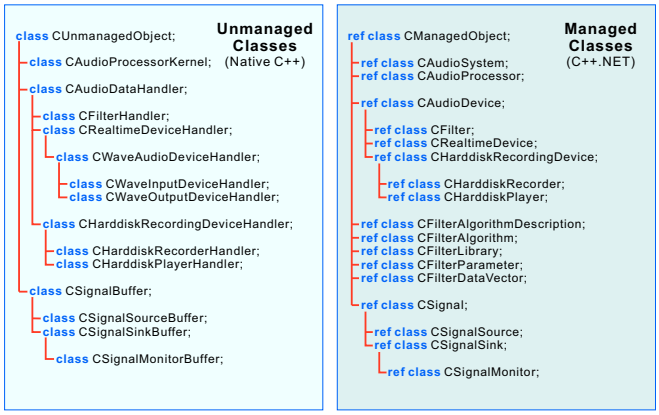

Figure 2: Real-time core: Native and .NET class hierarchies.

On the .NET side, the application creates a CAudioProcessor object and assigns CAudioDevice and CFilter objects to it. Calling the respective connection methods causes the corresponding CAudioDataHandler objects to be cross-referenced by pointers, thus establishing the desired data flow between hardware I/O and filters.

Filter algorithms are coded in native C++ using a generic C++ API, and are compiled to individual dynamic link libraries (DLLs). Through this API, information is provided at runtime which enables the GUI framework to build an individual control panel for each filter. A filter may have any number of I/O channels; the C++ language also allows direct use of high-performance math libraries (such as the Intel Math Kernel Library). At runtime, multiple filter DLLs can be loaded and multiple instances of each filter can be active simultaneously.

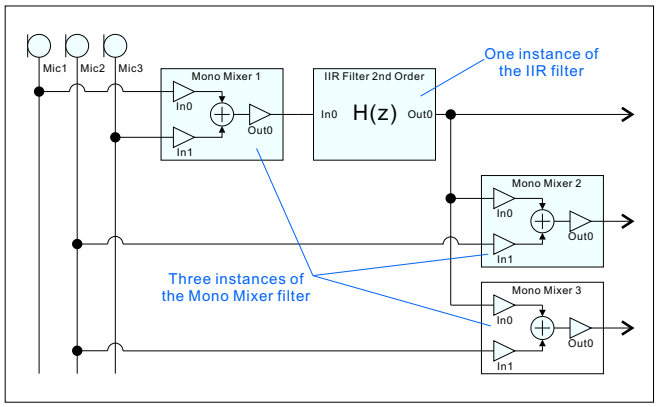

Figure 3: Example circuitry with three microphones.

Example

For illustration, we consider an experimental setup with three microphones as shown in fig. 3. The filter circuitry is realized by three instances of the Mono Mixer and one instance of an IIR filter. Fig. 4 shows the GUI main window with the signal routing. Each filter can be controlled by its own control window (fig. 5); signals can be viewed and analysed using oscilloscope and FFT analyser functions; spectral data can be written to a Matlab *m-file* for further external analysis and visualization (fig. 6).

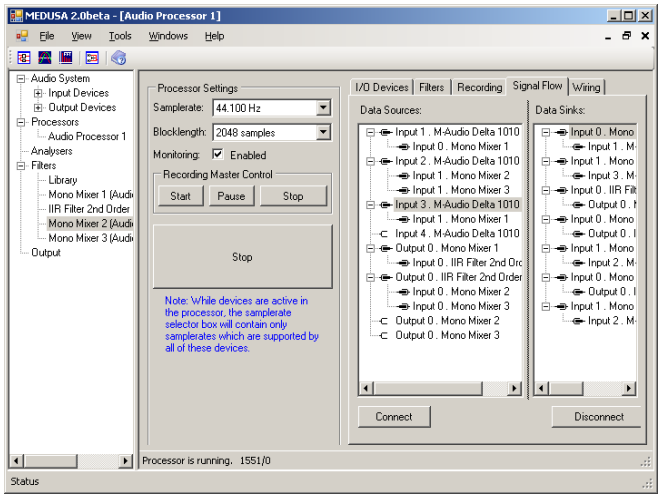

Figure 4: The main GUI window with signal routing.

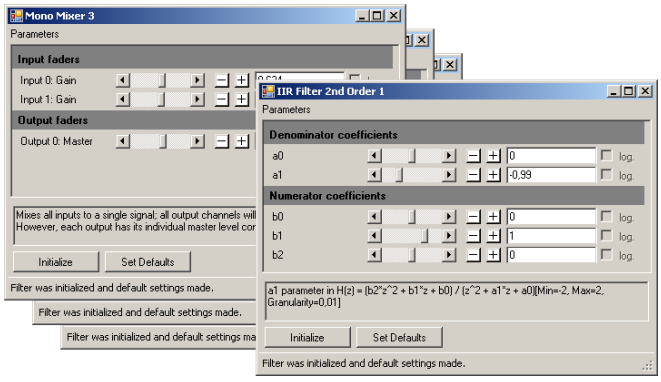

Figure 5: Control windows of Mono Mixer and IIR Filter.

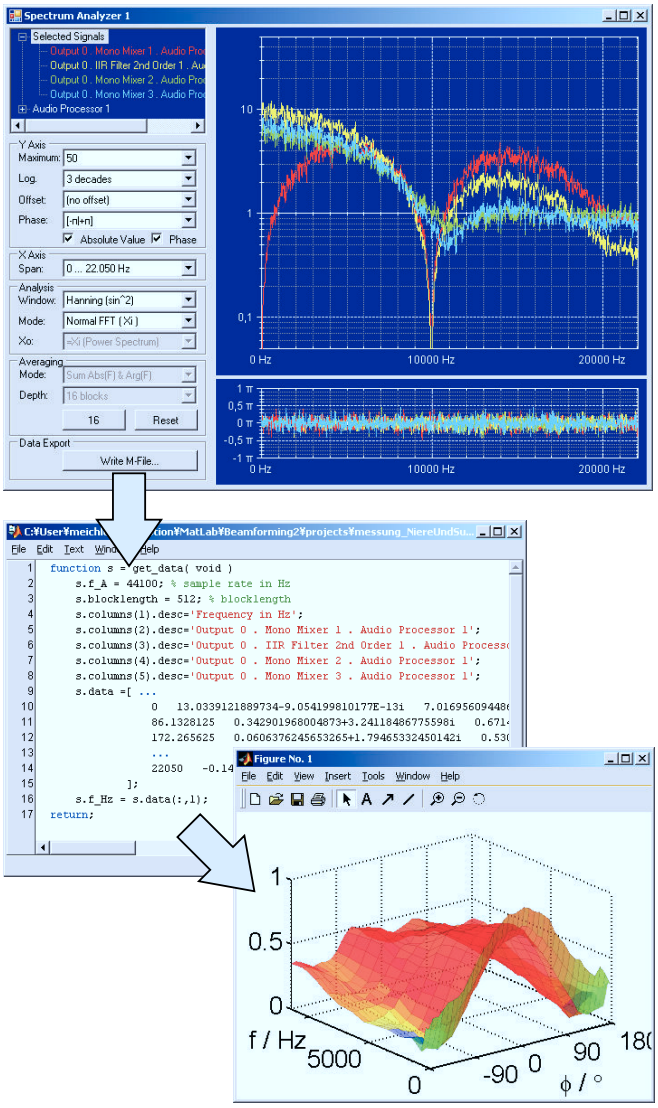

Figure 6: FFT analyser and data export (Matlab script file).

Summary

The software environment presented in this contribution operates in real-time and can process an arbitrary number of filters with any number of I/O channels simultaneously. Time and frequency domain analysis (oscilloscope, FFT, FRF) as well as recording/replay functions are provided. The strength of the system lies in its flexibility to work with any hardware using standard Windows drivers, and in its effortless extensibility. The GUI control of each filter enables the user to intuitively evaluate filters. In the context of microphone arrays, the software allows building prototypic setups, helping to auralize algorithms even before the target DSP hardware is developed.

References

[1] A. Hejlsberg, Sc. Wiltamuth, P. Golde, "The C# Programming Language", *Microsoft .NET Development Series*, ISBN 0-321-15491-6, Oct. 2003.

[2] G. Hogenson, "C++/CLI - The Visual C++ Language for .NET", Springer-Verlag, Berlin, Heidelberg, New York, ISBN 1-59059-705-2, 2006.