Case Study

A "Ducking" Delay

Steve Dwyer

Since completing my Certificate of Higher Education in the Mathematical Sciences, I've been working through Will Pirkle's book *Designing Audio Effect Plugins in C++*. I've since completed a number of effects pedals.

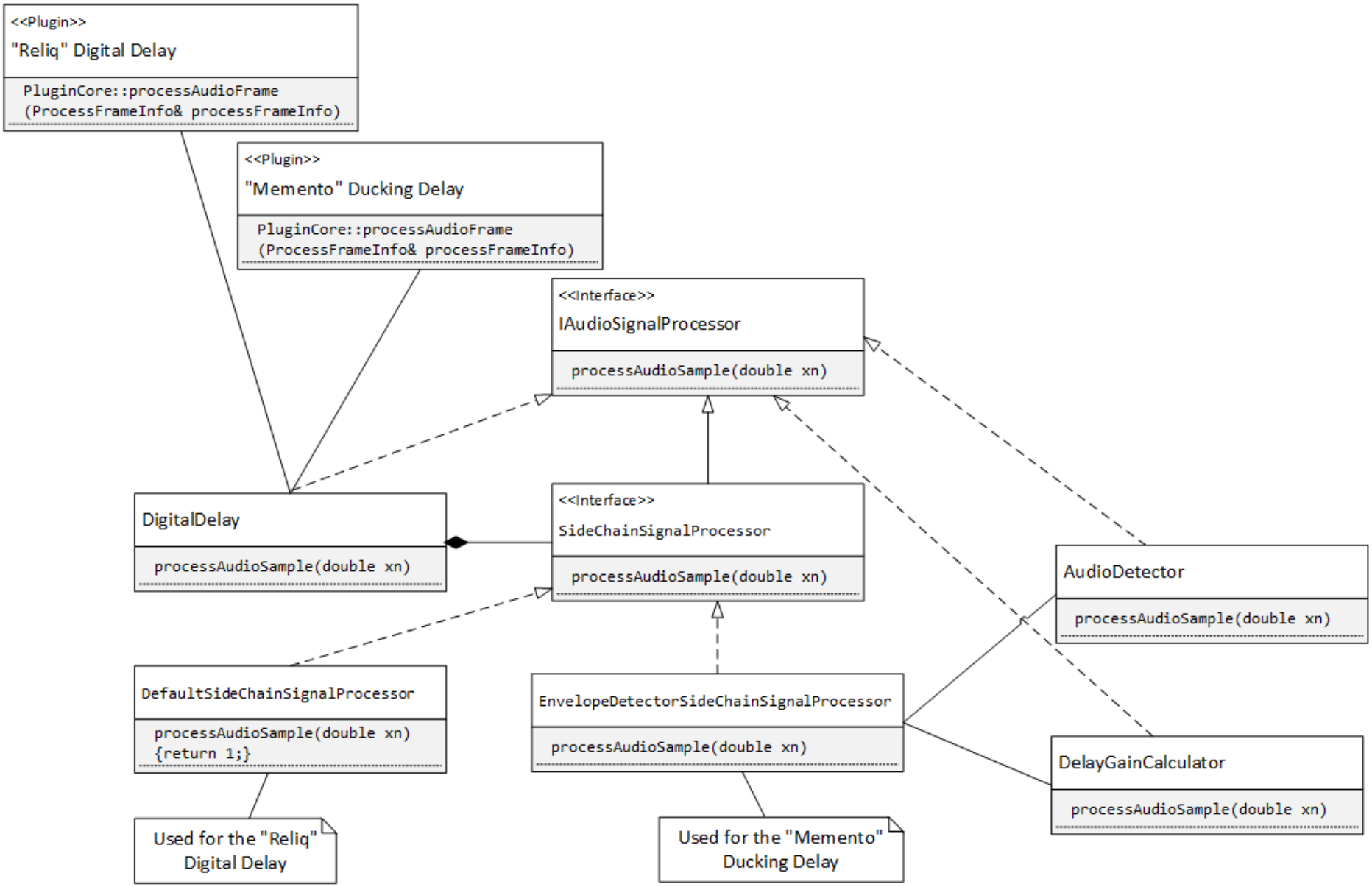
I would like to invite you to visit my GitHub repository https://github.com/DoomyDwyer/ASPiKProjects where my work is available for review. Each project uses classes from the files `customfxobjects.h` and `customfxobjects.cpp`. These files are maintained in a second repository, which contains common classes used across all of the effects pedals, and is available here: https://github.com/DoomyDwyer/ASPiKCommon. This is where I further worked on the exercises from the book, to give the effects my own flavour, and change the code to suit my needs and tastes.

After completing the chapter on Delay Effects and Circular Buffers, one of the homework assignments was to design a "Ducking" Delay (Pirkle 2019, page 412)[1], which uses a Side Chain Processor with an Envelope Detector, to attenuate the wet signal when its amplitude exceeds a given threshold (configurable via a knob). After having already completed the *Reliq* Delay Effect, with a stereo Ping-Pong Delay, I saw that there should be no need to copy this logic into a new class, just to add the Side Chain Processor, rather I could augment the "Plain Old" *Reliq* Delay, with additional behaviour using Object Composition.

My first step was to create a "stubbed out" `SideChainSignalProcessor`, called the `DefaultSideChainSignalProcessor`, which simply always returns a value of 1 from its signal processing algorithm. This could be used for the *Reliq* "Plain Old" Delay, thereby leaving the wet signal unaffected by the Side Chain Processing. The next step was to create the more elaborate `EnvelopeDetectorSideChainSignalProcessor` for the *Memento* "Ducking" Delay, which amplifies or attenuates the wet signal, dependent on whether the output from the Envelope Detector exceeds the threshold. The result was a single `DigitalDelay` class, whose behaviour could be changed at compile time by passing a different implementation of `SideChainSignalProcessor` to the `DigitalDelay`'s constructor. This Programming to Interface was achieved using templates.

The UML Class Diagram below communicates the intent of this design. `PluginCore` is a class and `IAudioSignalProcessor` is an interface from Will Prikles ASPiK framework, which can be used to create VST, AAX and AU plugins from a single code base. `AudioDetector` is a standard DSP class from the ASPiK framework. The other classes in the design I either built from scratch or customised examples from the book.

[1] *Designing Audio Effect Plugins in C++* Will B. Pirkle, Routledge, 2019. ISBN: 978-1-138-59193-6


<<Plugin>>
"Reliq" Digital Delay
PluginCore::processAudioFrame
(ProcessFrameInfo& processFrameInfo)
<<Plugin>>
"Memento" Ducking Delay
PluginCore::processAudioFrame
(ProcessFrameInfo& processFrameInfo)
<<Interface>>
IAudioSignalProcessor
processAudioSample(double xn)
DigitalDelay
processAudioSample(double xn)
<<Interface>>
SideChainSignalProcessor
processAudioSample(double xn)
AudioDetector
processAudioSample(double xn)
DefaultSideChainSignalProcessor
processAudioSample(double xn)
{return 1;}
EnvelopeDetectorSideChainSignalProcessor
processAudioSample(double xn)
DelayGainCalculator
processAudioSample(double xn)
Used for the "Reliq"
Digital Delay
Used for the "Memento"
Ducking Delay