AN ARCHITECTURE FOR CREATING HOSTING PLUG-INS FOR USE IN DIGITAL AUDIO WORKSTATIONS

Darrell Gibson
School of Design Engineering & Computing,
Bournemouth University,
Talbot Campus,
Fern Barrow,
Poole, Dorset.
BH12 5BB
gibsond@bournemouth.ac.uk

Dr Richard Polfreman
Department of Music,
Faculty of Humanities,
University of Southampton,
University Road
Southampton
Hampshire
SO17 1BJ
r.polfreman@soton.ac.uk

ABSTRACT

Although modern software-based DAWs (Digital Audio Workstations) offer the ability to interconnect with plug-in effects, they can be restrictive due to their architecture being largely based on hardware mixing desks. This is especially true when complex multi-effect sound design is required. This paper aims to demonstrate how a plug-in that can host other effects plug-ins can help improve the sound design possibilities in a DAW. This hosting plug-in allows other effects to be "inserted" at specific points in its internal signal flow. Details are given of a "proof of concept" plug-in that was created to demonstrate that it was possible to create plug-ins that can host other plug-ins, using Apple's AU (Audio Unit) format. The proof of concept is a delay effect that allows other effects plug-ins to be inserted in either the "delay path", "feedback path" or both. This Audio Unit has been extensively tested using different DAWs and has been found to work successfully in a variety of situations. Finally, details are given of how improvements can be made to the plug-in hosting delay.

1. INTRODUCTION

The concept of the audio effect plug-in[1] has been with us for a number of years now and these have revolutionized the audio recording industry. The DAW acts as a host that will then allow audio plug-ins to be "plugged in" at various points in the signal-chain to process the digital audio. The plug-in software can then manipulate the audio information from the host and feed the resulting audio back to the host, in real-time. Although plug-in technology is well established, most DAWs are created to replicate the "traditional" infrastructure of a hardware-based recording studio. This being the case, effects plug-ins can usually only be used as "insert" or "aux bus" effects, meaning they can only be applied at specific points in the signal chain. This means if the end-user wants to explore complex multi-effects sound design in typical production environment, then the routing provided by a DAW can be restrictive. Although some DAWs do support "free routing", the track infrastructure tends to get in the way and makes sound design a cumbersome and overly complex operation.

For example, consider a simple delay effect. This is usually implemented as a delay line delaying the incoming audio signal by a user definable number of samples within pre-defined limits. Some of this delayed signal is then fed back and added to (or mixed with) the incoming signal. Generally the level of this fed back signal can be adjusted between 0 and 100%. Now consider that there is a desire to create an effect where the fed back signal has a second process applied to it. This could be achieved by "inserting" another effect into the feedback path; commonly a (low-pass) filter to modify the frequency content of the delayed repeats (echoes) of the signal. However, more unusual effects could be created, such as a modulation effect or compressor in the feedback path. The structure of such an effect is show in **Figure 1**.

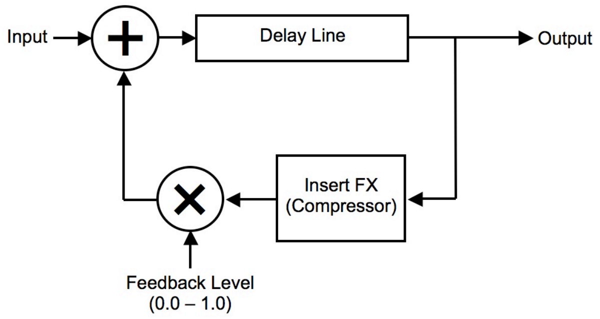

Figure 1. Inserted Plug-in Feedback Effect Delay

Unfortunately existing delay plug-ins with an effect in the feedback chain offer little or no flexibility in the type of effect provided; it is a fixed process, although usually with some control over parameters. If a more general effect were to be created with a DAW it would have to be implemented with one audio track and two auxiliary busses. The required connections to achieve this structure in a DAW are shown in **Figure 2**.

Although this connection system can be realized in most DAWs, others are more restrictive on the routing

[1]Virtual instrument plug-ins are also available, and to which similar principles apply, but are not the focus of this work.

allowed and will not allow a feedback path to be created[2]. As a result, to implement the same effect a physical connection must be made on the audio interface to create the feedback connection from an input to an output. This means two audio tracks and one auxiliary bus are required, plus typically the addition of digital to analogue and analogue to digital conversions.

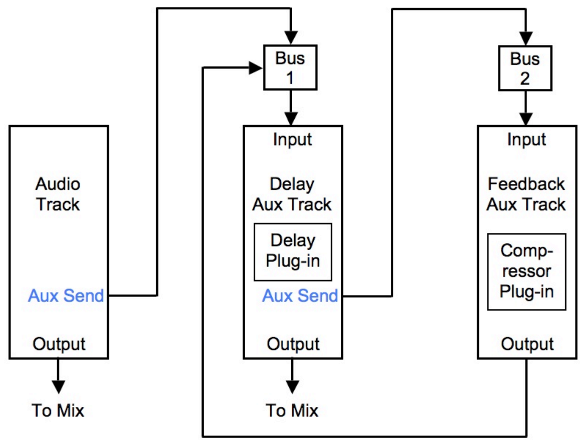

Figure 2. Connection Diagram to Implement a Feedback Effect in a DAW

While more flexible alternatives to using a DAW do exist, such as Max/MSP and Pure Data, they are less frequently found in typical music production facilities and require programming knowledge to build the effects.

Although here only delay-based multi-effects have been considered, the same principles can be applied to many other categories of audio effect, particularly where "feedback" is a natural component of the process.

1.1. Hosting Audio Plug-ins

Rather than a track-based solution, an alternative is to use audio plug-ins that allow other plug-ins to be "inserted" at specific points within their signal flow. In this way, the hosting plug-in must be able to perform its given operation, but must also be able to host other plug-ins. To do this the effects plug-in must also have a host built within it. For example, if a delay plug-in were built with a plug-in "insert slot" in the feedback path, the same multi-effect as described above could be created without additional DAW resources and in a much more convenient manner for the user.

With such an effect, if no plug-in is used in the available insert slot then it will just operate as an ordinary delay effect. However, complex multi-effects can be created if the insert slot is used to modify the fed back audio signal.

[2]For example, Steinberg's Cubase does not allow a feedback path to be created.

2. AN ARCHITECTURE FOR CREATING A HOSTING PLUG-IN

The integration of an insert slot within an effect requires that a plug-in host be built within the effect plug-in. To clarify how this is being created and to establish consistent terminology the architecture shown in **Figure 3** was defined.

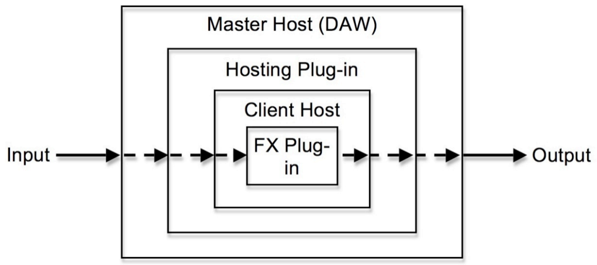

Figure 3. Architecture of a Hosting Plug-in

First, there is the "Master Host" which in this case is a DAW in which the "Hosting Plug-in" will be used. It is possible to place the Hosting Plug-in into any available plug-in slot (insert or aux bus) within the Master Host, as if it were any ordinary effect plug-in. Then within the Hosting Plug-in a local host is created that is known as the "Client Host". The Client Host then hosts the inserted "FX Plug-in", which can be "inserted" at a point in the Hosting Plug-in's signal flow. Audio content is passed from the Master Host to the Hosting Plug-in, which performs the required processing on the audio, before it is then passed to the FX Plug-in, via the Client Host. When the FX Plug-in has performed its given operation the audio is passed back to the Hosting Plug-in, again, via the Client Host. The Hosting Plug-in can then perform any further necessary processing. Finally, the processed audio is then passed back to the Master Host for any further manipulation and output.

2.1. AU Hosting Plug-in

The Core Audio framework under Mac OS X that contains AUs has extensive Application Programming Interfaces (APIs) for creating both hosts and plug-ins [1], [2]. The hosting plug-in architecture is based on the Core Audio "pull" model, which allows audio supplied to the Hosting Plug-in to be "pulled" through the FX Plug-in, via the Client Host [1]. This was achieved by creating an `AUGraph` (a signal chain of connected AUs) that connects the required FX Plug-in AU to a `GenericOutput` AU. An *input render callback* function was then written, which is called whenever the `AUGraph` needs input data. The callback simply copies data to be passed to the FX Plug-in into the designated buffer. The audio "pull" is then initiated by iteratively calling the `AudioUnitRender` function of the `GenericOutput`. This render is called through the graph's units until the *input render callback* is reached. The data is then processed by the `AudioUnitRender` of the FX Plug-in AU, and in turn passed on to the

`AudioUnitRender` of the `GenericOutput` AU. When this is completed, the data is ready for any subsequent processing required in the Hosting Plug-in. This system is shown diagrammatically in **Figure 4**.

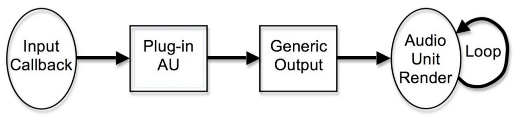

Figure 4. AU FX Plug-in's Audio Render Mechanism

To reduce the processing overhead of the AU Hosting Plug-in to a minimum, pointers to the data are supplied rather than copying data to or from the `AUGraph`. This takes the processing overhead of the AU Hosting system to virtually nothing.

2.2. AU Hosting Plug-in Channels

Core Audio uses a kernel system to manage operations that are performed across multiple channels in Audio Units [2]. However, in order to "pull" the audio data through the Hosting Plug-in's AUGraph, as explained in Section 2.1, it was necessary to remove the kernel system and override the `ProcessBufferList` method in the underlying `AUEffectBase` class [2]. This is because the `AUGraph` renders all the channels at once and there is no way to separate these out within the kernel. Using the `ProcessBufferList` method, all the "pulled" data can then be copied (using pointers) into the appropriate `AudioBufferList`. The disadvantage of removing the kernel system is that it then means the plug-ins channels must be managed manually.

It was decided that in the first instance the AU Hosting Plug-in architecture would be created to operate in three different channel modes. The three supported channel configurations are the most widely used in music production and are: mono->mono, mono->stereo and stereo->stereo. This can be easily expanded in future if required.

2.3. Selecting FX Plug-ins

To let the user choose which FX Plug-in will be placed into the insert slot it is necessary to build a list of all the available plug-ins. This must be done dynamically when the Hosting Plug-in is initialized as new plug-ins may be installed on the system at any time. Also, as discussed in Section 2.2, the Hosting Plug-in will be able to operate in three different channel configurations. Therefore, when the list of available AUs is built it is important to determine that all the FX Plug-ins in the list can operate in the current channel configuration.

2.4. AU Hosting Plug-in Factory Presets

A mechanism for managing Factory Presets was also incorporated. At this stage only two presets were added, but can be easily expanded in the future.

3. PLUG-IN HOSTING DELAY EFFECT

In order to test the validity of the architecture a "proof of concept" was performed to verify the approach to hosting other plug-ins within an effects plug-in. A delay effect was chosen here as it offered the possibility of incorporating multiple FX Plug-ins.

3.1. Delay Structure

The chosen implementation for the delay is based on a single delay tap [3], [4]. This is constructed from a single digital delay line with a level controllable feedback path. The output of the delay tap is then mixed with the input signal using a feed forward network to give a wet/dry mix. Although in the first instance the Plug-In Hosting Delay Effect was created with just a single delay tap, the code has been written such that additional taps can be added easily.

The implementation of the digital delay line used in the delay tap is based on a circular buffer, where a separate "read" and "write" pointer are used so that the data in the buffer does not actually need to be moved [3]. Although this gives a very efficient implementation for the delay line it can result in "zipper noise" when the delay time is continually changed in real-time. This is because the "read" pointer will move to a new location. One way to resolve this is to use a fractional delay line, where it is possible to obtain fractional-length delays [4]. This is achieved by using an interpolator to calculate the output sample that lies between two samples. Several interpolation algorithms have been proposed for audio applications with different computational intensities and performance characteristics [5]. Computationally a linear interpolator will have the lowest impact, and so is used here, but it does produce a low-pass filtering effect that will remove some of the high frequencies from the audio content [4].

Although the interpolator removes "zipper noise" it does not stop an audible jump when the read pointer's position is step changed. Others have previously used a "crossfade" system [3], where two delay buffers are used. When the delay time is changed the read pointer for one of the buffers is moved to the new location and the other is left at the original location. A crossfade is then performed between the outputs of the two delay buffers. This results in a smooth transition from the "original" delay audio to the "new" delay audio. However, there is a cost, as it requires an additional delay buffer the same size as the original.

3.2. Plug-In Delay Effect

The plug-in delay effect was then created so that two plug-in slots were available. One in series with the delay line and other in the feedback path. In this way, other effects can be added to the delayed signal, just the fed back signal or both signals. The structure of the plug-in delay effect is shown in **Figure 5**. The two plug-in slots have been created by generating a separate `AUGraph` for each. The first plug-in slot, in the delay path, was fairly straightforward to achieve and uses exactly the same principle as defined in Section 2.1. That is, when the render method for the Hosting Plug-in is called, the same amount of data as defined for the Hosting Plug-in's render is "pulled" through the FX Plug-ins `AUGraph`.

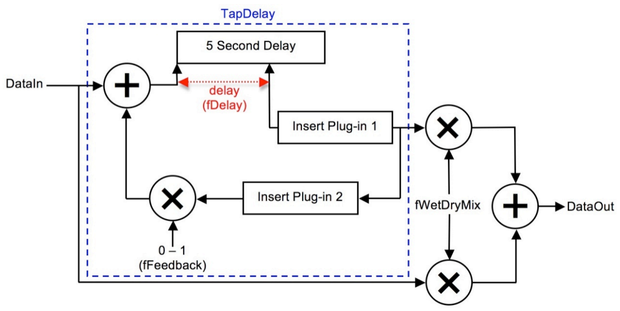

Figure 5. Plug-In Hosting Delay Effect

However, when the feedback plug-in slot was added this system had to be changed. The reason for this is that it is perfectly feasible that the delay time in samples (`fDelay`) could be less than the render block size (`iFrames`). In which case, all the required data would not be available in the delay line. This would result in incorrect audio being produced. To resolve this issue it was necessary to derive a mechanism where the render block size (`iFrames`) is compared to the current delay size (`fDelay`). If the delay size is less than the render block size then only a block of data equal to the delay size is rendered and the remaining block size is calculated. This process is iteratively repeated until there is no more data to be rendered. Using this technique there is always enough audio in the digital delay line to be rendered. However, it does result in the processing of the `AUGraphs` being executed in smaller blocks, which slightly increases the CPU load. Nonetheless the residual CPU load for the Hosting Plug-in, hosting no plug-ins, is less than 2%[3].

4. CONCLUSIONS

The development of the Plug-in Hosting Delay effect clearly demonstrates how audio plug-ins can be "inserted" within the developed effect. This opens up sound design avenues that although possible with existing DAWs and plug-ins, they are somewhat cumbersome and outside of most users' comfort zone. By integrating the Plug-in Hosting architecture into other plug-ins it then makes these accessible and easy for all users. This Plug-in Hosting Delay effect has been realized using the Apple Core Audio frameworks, resulting in an AU plug-in that has been successfully tested with a variety of DAWs using different configurations and was found to perform as expected. However, there are still a number of improvements that can be made to the plug-in. Possibly most importantly, it was originally hoped that the Plug-in Hosting Delay effect would be able to implement a "dynamic delay" effect where a compressor is placed in the feedback path of a delay. Although the designed Plug-in Hosting Delay effect does allow a compressor plug-in to be inserted into the feedback path, this then needs to be side-chained to the input audio. Core Audio does not supply a specific mechanism for implementing side-chain inputs, which means they have to be implemented as separate buses and then managed manually inside the Audio Unit. Implementing this is planned as further work.

Although "factory presets" have been created for the Plug-in Hosting Delay effect, these only allow the parameters in the Hosting Plug-in, and not the FX Plug-in, to be configured. This is because, to the Hosting Plug-in, the FX Plug-in is just a "black box" and it knows nothing about what this plug-in does or how it operates. Also for the same reason it will not be possible to automate FX Plug-in parameters in the Master Host. To resolve both of these problems "dummy" parameters could be created that are normally ignored by the Master Host. Then when an FX Plug-in is selected, its parameters are mapped to the available "dummy" parameters and these are then "published" so they become visible to the Master Host. If this is done then the parameter values can be included into "factory presets" and they can also be used in the Master Host's automation system.

"User presets" for the Hosting Plug-in are managed by the Master Host and can be saved and loaded as expected. However, there is currently no such system for the Client Host, which means that there is no way to have "user presets" for the FX Plug-ins. This means any work that the user does with an inserted plug-in cannot currently be saved. This procedure for "user presets" needs to be modified so that the presets are saved as part of the Hosting Plug-in presets and then integrated into the Client Host.

5. REFERENCES

[1] Apple Inc, *Core Audio Overview*, 2007.

[2] Apple Inc, *Audio Unit Programming Guide*, 2007.

[3] Smith, J. O. *Physical Audio Signal Processing: For Virtual Musical Instruments and Audio Effects*, 2010.

[4] Zölzer, U. (Editor), *DAFX:Digital Audio Effects*, 2002.

[5] Rocchesso, D. "Fractionally Addressed Delay Lines", *IEEE Transactions on Speech and Audio Processing*, Vol. 8, No. 6, November 2000, pp. 717–727.

[3]When run on a MacBook Pro with 2.6 GHz Intel Core 2 Duo and 4GB 667MHz DDR2 SDRAM.