Audio Engineering Society

Convention e-Brief 656

Presented at the 151st Convention
2021 October, Online

Developing plugins for your ears

Gary Spittle, Michael Lee, and Weiming Li

Sonical, San Mateo, California, USA

Correspondence should be addressed to Gary Spittle (gary.spittle@sonical.ai)

ABSTRACT

We present a new intuitive development platform that allows algorithm developers to put plugins in our ears. The growing number of advanced audio processing plugins developed for DAWs is enabling highly creative sound experiences. We explain how plugins for DAWs can be easily ported to small embedded processors used in ear worn products and other audio devices. This includes signal processing targeting low latency, low power, high compute and large memory plugins. We describe an open development platform to bring machine learning based algorithms directly to the end user. This will also give plugin developers access to data streams from additional sensors and multichannel audio data beyond stereo music streaming. The next generation of hearables for gaming, music, movies, AR/VR will require processing techniques currently only available to professionals in studios. These new development tools allow algorithms to be created such that end users can select, download and control plugins to unlock innovation that fits their individual needs and personal preferences.

1. Introduction

Today's headphones and other ear worn devices such as hearing aids and true wireless Bluetooth earbuds, are becoming significantly more feature rich. The compute capabilities of other devices we regularly interact with, such as laptops, gaming consoles, mobile phones and tablets, provide a strong increasing trend in capabilities and performance. This has enabled enormous creativity in software and user experiences. The same level of efficient compute will soon be available in hearable devices which serves to unlock a new wave of innovation.

Our ear devices will do significantly more than be used for listening to music or making phone calls. Users will download applications and plugins that will be customized for the different use cases we experience during a day. This will include spatial rendering with head tracking. It will also enable the processing of ambient sounds in a way that is personalized and based on each individual user's preferences. It will bring new processing algorithms for gaming and movie content. It will combine the real world with multiple virtual scenes from many different sound sources and devices.

The community of audio algorithm developers have been unable to deliver their plugins directly to consumers, until now. They typically require permission from end device manufacturers or are limited to DAW platforms. User demand for new technology is converging with our appetite for immediate downloadable applications, just like we do with our mobile phones and tablets and other smart devices.

We are witnessing a platform transition where plugin developers will be able to deliver their algorithms directly to the end user's earphones. This will allow a user to select plugins for specific use cases during the day to enhance the way they interact with the real and virtual worlds, bringing them together into personalised experiences.

This requires a new set of development tools. We are creating an embedded platform that brings advanced processing capabilities to wearable devices to enable plugin developers direct access to consumers. A complete hardware and software stack provides the foundation of a development environment that will enable a rapid migration of professional plugins to consumer applications. The platform will present software APIs that map onto traditional plugin development frameworks such as VST and JUCE. An embedded operating system, specifically designed for low latency audio processing for low power applications, will provide a simple host framework. The framework will allow multiple plugins to run in a single platform, fully controlled by the user.

The graphical user interface can be accessed and controlled by the user from their existing devices, such as a laptop, mobile phone or tablet. Here they will select the plugin developer's GUI that is connected to the embedded code running in the wearable device.

We discuss how the simple migration of existing algorithms will provide a baseline of experiences to build on for more advanced wearable products. This will include multiple data streams including local sensors and multi-channel audio. We explain how the end user will access plugins and manage their experiences during a day. No longer will our hearable devices be fixed in their functions and performance. We will not be relying on end device manufacturers to create a feature set that meets our individual needs. Instead, we will create our own menu of plugins that we can select and activate, as needed, during the day.

2. Why is it important

Many impressive audio processing algorithms have been developed for specific applications such as music compression, audio streaming, speech enhancements, noise reduction and spatial rendering, to name just a few. Now there is a growing interest in machine learning based algorithms with some astonishing results. It is clear that some of these processing modules are best placed in the cloud, others need to run on devices we carry with us and yet some are most effective if running in, or on, our ears. Due to latency and data streaming limitations, it makes sense for many algorithms to process data locally in the ear device.

Our ears are used for a lot more than listening to music. The development system will provide a framework that supports stereo audio processing and rendering. There are also a number of extensions of a typical headphone system that can be implemented to evaluate other types of audio processing algorithms. For example, this can include the capture and processing of the sounds around us to provide augmented hearing. This could bring enhancements and intelligent control of noise to everyone, not just those that have hearing loss. The EarOS framework allows for multiple microphone and sensor data streams to be passed to the plugin for processing. This allows for low latency processing of binaural signals and also the analysis and monitoring of sounds that the user is exposed to.

This raises a number of technical challenges. Plugins are generally developed on a computer. Some systems allow plugins and algorithms to be ported to an embedded platform or small single board computer, such as Elk [1] for plugins on hardware and OpenMHA [2] for hearing algorithms on a BeagleBone Black [3]. Although these solutions provide a number of advantages over a laptop for real world testing it doesn't represent a real device that an end user will purchase.

Other platforms use the mobile phone as the processing host (UTDallas [4]) or have created development boards based around ARM processors (Tympan [5]). The intent of the EarOS is to allow these solutions to make the final step to the ear and embed their plugins into the ear worn device in preparation for user download. This will also provide a useful starting point for developers building on the open speech platform (OSP) as created by the University of California, San Diego [6].

Instead, we have identified the need for reference designs that contain all the electronics of a typical device in a representative form factor. This enables plugin developers to build algorithms for a real target device that closely resembles what a manufacturer is

likely to build, or where the plugin developer could design and 3D print their own form factor.

3. The solution

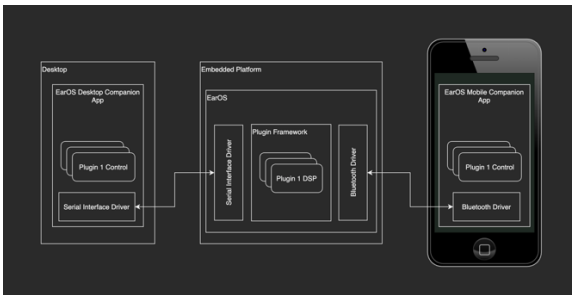

Figure 1: Sonical system with the EarOS

1. The concept

We have created a wireless audio platform that can enable all phases of algorithm development from laboratory research to end user deployment. The platform is a small form factor, battery powered, wireless multichannel audio device and accompanying software. The system, using its real-time OS and plugin framework, can be easily configured and dynamically updated via a mobile companion app enabling fast iterations during experimentation phases. As previously mentioned, the goal is for this platform to become widely available amongst mainstream consumer headphone related products.

2. System description

The system is comprised of three main components: a wireless audio embedded system, the EarOS and audio plugin framework, and a mobile companion app for controlling the embedded system. Algorithm researchers and plugin developers will typically produce and deploy two separate artifacts: the compiled plugin to be hosted by EarOS on the embedded device and a compiled software component hosted by the companion app framework. The embedded system includes both serial and Bluetooth interfaces for either wired or wireless remote control.

3. EarOS

The EarOS is a multi-tasking embedded real-time OS with audio centric features with primary design goals to maximize processor utilization, minimize resource constraints caused by fixed allocation schemes, and enable add-on and post deployment flexibility. To achieve this, the underlying system implements a task scheduling and data delivery mechanism, basic file management, a hardware abstraction layer, and a flexible and dynamic audio plugin framework to realize complex signal graphs. The system supports down to 1mSec latency and sample rates ranging from 8kHz to 96kHz without restrictions on either dimension. Further, logging facilities in the OS enable metrics and debug information to be gathered for data analysis.

4. Plugin framework

One of the primary goals of our initiative is to provide plugin developers and researchers with easy-to-use tools to deploy and experiment with new algorithms. We kept the following objectives in mind:

- Make it easy to move existing algorithms and code to the Sonical platform
- Facilitate the development of mobile companion components that run both iOS and Android platforms
- Enable control of the embedded plugins over standard BLE services (GATT and MIDI)

As we expect many novel plugins to come from the digital music processing community, we specifically sought out an approach that would facilitate developers in bringing their algorithms to our platform. Within this community, many cross platform plugin tools exist with many based on the JUCE framework (others options include VST, JUCE, Distrho, and iPlug2) [7] [8] [9] [10]. Of these, JUCE can generate the broadest range of software components: VST (Win/OSX), AU (OSX/iOS), standalone apps (Win/OSX/iOS/Android/Linux (including headless)), and native Android activities (Android). Because our system separates the control surface from the DSP and requires generating both mobile and embedded components, none of these existing solutions are a direct match for our needs. However, we have taken inspiration from these cross-

platform approaches to fashion our own plugin development tool chain to produce the requisite embedded signal processing and companion app artifacts. SOUL, a recent initiative from JUCE, is similar in concept, but requires a runtime element on the embedded device [11]. Yet another system by Elk.Audio also seeks to facilitate bringing VST plugins to embedded platforms [1]. The Elk system which boasts over 500 ported VST plugins, is based on the RaspberryPi platform and takes advantage of the headless Linux JUCE exporter.

Underneath the hood, each plugin is instantiated as its own task. As with many other frameworks, the core functions required by developers are the initialization, process, and release functions. Each plugin will be able to save and restore its own state between power cycles. Audio and control data is passed between plugins using the internal messaging system enabling us to create complex signal graphs and control schemes. Plugins can receive messages via the serial interface or BLE paths. The signal chain is assembled by establishing links between the different plugins and the entire signal path can be assembled and disassembled dynamically without the need to pause or break the audio stream: everything still runs without interruption.

The EarOS employs both priorities and data flow systems to schedule plugin tasks. This enables the system to handle a mixed application environment naturally; for example, one with data coming from both sensors and audio subsystems.

5. Example application

As an example, we have deployed the EarOS on a prototype hardware development platform and modeled after selected opensource plugins implementations [12] [13]. The prototype hardware has the following elements:

- 6in/6out Maxim 98090 codecs
- Nordic nRF52840 BLE
- STM32H7 series processor
- GPIO drivers for handling button events and lighting
- Drivers for temperature, motion, and other common sensors

Communication between the hardware and other control surfaces can take place over a serial COM interface (desktop) or over BLE MIDI.

The example signal chain implemented on the demonstration hardware consist of a number of signal splitters (the internal channel count for this example is set to 6), a flexible 8 stage EQ section, fractional delay, and gain elements. The EQ implements a variety of different filter types (shelf, high/low pass, parametric, notch, and all/band pass filters). The fractional delay unit operates on the sample level and the gain elements include phase inversions. The entire signal graph operates with 1mSec latency and has enough headroom to accommodate additional plugins if needed.

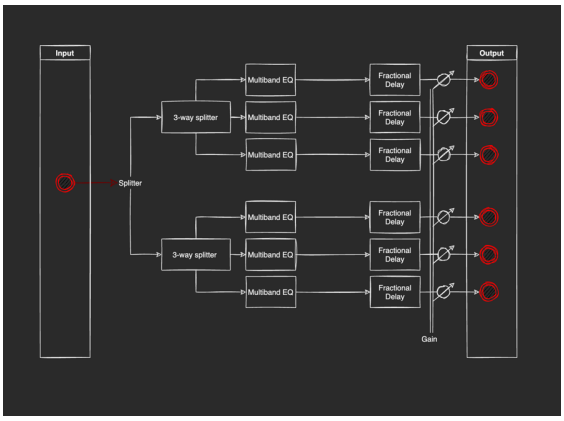

Figure 2: Example signal chain

The companion app is built using JUCE with OSX, Windows, iOS, and Android targets. The desktop applications use the serial interface while the mobile devices use BLE MIDI as the communications pipe. The embedded BLE midi parser is based on the midi-message-parser opensource repo [14].

Figure 3: Companion app screenshot

It is expected that any production application built for the platform will enable downloadable plugins via iOS AU or Android dynamic feature module mechanisms.

4. Conclusions

Ear devices with in-ear compute are expected to become an increasingly important part of our wearable world. In this paper we have presented our flexible digital audio solution. It seeks to deliver features absent from current platforms and can facilitate not only rapid prototyping of algorithms and contextual experimentation, but eventual deployment and monetization of audio plugins into the market. The solution combines a real-time OS with a dynamic plugin architecture with hardware that can be put into market ready form factors. Further, the platform can give researchers access to data streams from additional sensors to combine with multichannel audio data. Once deployed onto the device, plugins and algorithms can be manipulated by a mobile companion app allowing both researchers and users to experiment.

References

[1] Elk.Audio, "elk.audio," elk.audio, [Online]. Available: https://elk.audio/audio-os. [Accessed 25 August 2021].

[2] OpenMHA, "Open community platfrom for hearing aid algorithm research," openmha.org, [Online]. Available: https://openmha.org. [Accessed 25 August 2021].

[3] beagleboard.org, "BeagleBone Black - BeagleBoard.org," beagleboard.org, [Online]. Available: https://beagleboard.org/black. [Accessed 25 August 2021].

[4] University of Texas, Dallas, "Smartphone-Based Open Research Platform for Hearing Improvement Studies," utdallas.edu, [Online]. Available: https://labs.utdallas.edu/ssprl/hearing-aid-project/research-platform. [Accessed 25 August 2021].

[5] Tympan, "Tympan is Open Hearing," tympan.org, [Online]. Available: https://shop.tympan.org. [Accessed 25 August 2021].

[6] L. Pisha, J. Warchall, T. Zubatiy, S. Hamilton, C.-H. Lee, G. Chockalingam, P. Mercier, R. Gupta, B. Rao and H. Garudadri, "A Wearable, Extensible, Open-Source Platform for Hearing Healthcare Research," *IEEE Access,* vol. 7, pp. 162083 - 162101, 2019.

[7] Raw Material Software Ltd., "JUCE," juce.som, [Online]. Available: https://juce.com. [Accessed 25 August 2021].

[8] Steinberg, "3rd Party Developers," steinberg.net, [Online]. Available: https://steinberg.net/developers. [Accessed 25 August 2021].

[9] Distrho, "distrho," sourceforge.io, [Online]. Available: https://distrho.sourceforge.io. [Accessed 25 August 2021].

[10] iPlug2, "iPlug2 - C++ audio plug-in framework," github.io, [Online]. Available: https://iplug2.github.io. [Accessed 25 August 2021].

[11] Roli, "Soul - the future of audio coding," soul.dev, [Online]. Available: https://soul.dev. [Accessed 25 August 2021].

[12] J. Gil, "Audio-Effects," github.com, [Online]. Available: https://github.com/juandaglic/Audio-Effects. [Accessed 25 August 2021].

[13] D. Walz, "Frequalizer," github.com, [Online]. Available: https://github.com/ffAudio/Frequalizer. [Accessed 25 August 2021].

[14] N. Hill, "midi-message-parser," github.com, [Online]. Available: https://github.com/BinaryNate/midi-message-parser. [Accessed 25 August 2021].