RackFX: A Cloud-Based Solution for Analog Signal Processing

Sean Peuquet
Ludic Sound
Denver, CO, United States
seanpeuquet@ludicsound.com

David Jones
RackFX
Berthoud, CO, United States
drj@rackfx.com

ABSTRACT

This paper introduces the RackFX platform as an alternative to using digital plugins that emulate analog hardware devices (plugin modeling) as a way to incorporate analog sound characteristics into computer-based music production. RackFX technology provides digital access to actual analog hardware devices. Given existing technological, social, and perceptual tensions between digital audio workstation-based effects plugins and outboard analog processing units, the RackFX platform provides a cloud-based solution. The platform is presented as a way for a community of internet users to interface directly with off-site analog hardware for the purposes of signal processing. A technical overview of the platform is provided, which outlines user experience and various server and on-site robotic processes that ultimately support the return of an analog-effected digital audio file to the user for each job processing request. Specifics regarding robotic control of analog devices and how the digital audio is handled and routed through the system on the device-side (on-site) is paid particular attention. Finally, the social implications of using the platform are discussed, regarding the cultivation of a community of users with unprecedented and affordable access to analog gear, as a new way to leverage digital technology toward the democratization of music production tools and techniques.

1. INTRODUCTION

The fetishization of analog audio recording, production, and reproduction technology shows no sign of abating. In the large and ever expanding field of music technology, analog hardware continues to be associated with musically desirable psychoacoustic descriptors, most notably 'warmth'. While the term warmth is strongly correlated to the acoustic phenomena of harmonic distortion and high frequency roll-off, the idiosyncratic production and (inter)subjective perception of analog warmth poses interesting problems for the computer musician.

Digital music technology replicates, processes, and stores audio exactly according to its software programming and the hardware limitations of the computer the code runs on. Which is to say, digital music is (barring any hardware stability issues) deterministic–from the moment directly after analog to digital conversion until the moment the signal is converted back to analog for sound reinforcement. Modeling the effect of psychoacoustic warmth using digital signal processing (DSP) techniques thus poses a hierarchical problem regarding the accurate representation of sound; no longer is the mere capture and digital representation of an analog signal at issue, but rather the problem concerns the capture and representation of how that analog signal was produced, which necessarily entails some degree of indeterminacy. As Karjalainen and Pakarinen decribe, "virtual analog modeling seems straighforward but is found demanding due to the nonlinearities and parametric variation in the analog domain." [1] The desired perceptual excess of an analog processed signal, it's warmth, is largely a direct result of the physical components of the analog system, their unpredictability and imperfection. In this respect, the modeling of analog effects (warmth correlates) using DSP is also closely related to synthesis, specifically physical modeling synthesis.

While the modeling approach has led to great successes and a burgeouning marketplace for software instruments and analog modeled plugins alike [2], there remains both a precision problem and a perception problem regarding the refinement and accuracy of our models. The question remains: what physical interactions are necessary to model and to what degree of accuracy– sufficient to overcome the just noticable difference (JND) in respect to some analog reference point? Despite Julius O. Smith's 1996 pronoucement (regarding synthesis) that, "we appear to be approaching *parity* between real and virtual acoustic instruments," [3] we are twenty years on and it appears that the *lack* of parity is increasily what structures both the popular discourse and commecial reality of music recording and production. The cello has yet to be fully replaced, in the same way that people who actually have access to a vintage Fairchild 670 would claim that all attempts to emulate the device as a digital plugin have failed. So despite the ease and accessibility of plugin emulators, actual vintage analog hardware processing units remain the gold-standard.

Counter to the prevailing trend of digitally modeling analog processes that yield the sensuous qualities of sonic warmth, the authors have sought to simply digitize access to the analog components and processing itself. The RackFX platform is essentially a "straight from the horse's mouth approach" to analog signal processing. While the idea of enabling distributed access to physical acoustic resources is not without precedent (see the Silophone [4] or the the MIT Responsive Environments Group's "Patchwerk" web interface to Joe Paradiso's massive modular synth [5]), the

RackFX platform is a uniquely scaleable and flexible solution with potentially longer-term consequences and implications. The technology was conceived of by David Jones, and developed by Jones and Sean Peuquet across much of 2015. The platform will be live for beta users starting May 14, 2016. Across the rest of this paper, the RackFX platform will be presented as both a technological solution and a paradigm shift regarding issues of access, affordability, and quality that govern the viability of signal processing using analog hardware devices.

2. TECHNICAL OVERVIEW

2.1 User Web-App Experience

The RackFX platform begins with a community of internet users. By creating an account and logging into the RackFX web-app (http://rackfx.com), each user is presented with a "dashboard" listing previously completed processing requests or 'jobs' and a drop area for uploading a digital audio file in .wav or .aif format (sampling rate and bit depth of the uploaded file are entirely flexible). See Figure 1 below for a screenshot of the current dashboard interface layout for any given user.

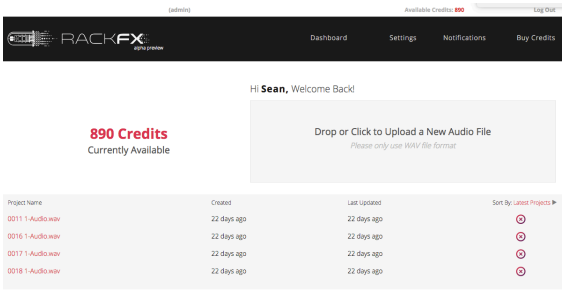

Figure 1. RackFX user dashboard

Once the user has uploaded a .wav file, she will be taken to a page displaying the waveform of the uploaded file, with an opportunity to play it back. At this point the user can decide to "add processing" (see Figure 2).

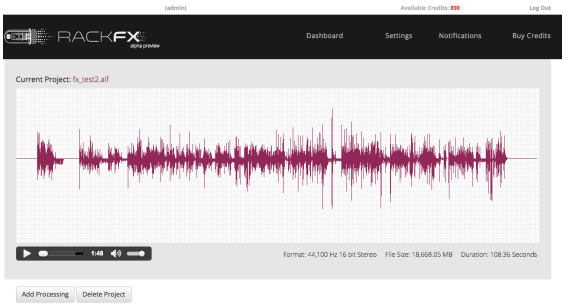

Figure 2. RackFX uploaded audio file waveform

Once the user decides to add processing to the uploaded digital audio file, she will be presented with a page detailing the devices that are currently hooked up to the RackFX system and listing which devices are currently online or active (see Figure 3).

The user selects an available analog device and is taken to a device-specific page to set parameters for how that device will process their signal. Parameters are unique to each device and so the available sliders on the web interface reflect what is available given the particular configuration of hardware knobs, sliders, buttons, etc. The user sets the desired parameters and then clicks "process audio" (see Figure 4).

Figure 3. RackFX uploaded audio file waveform

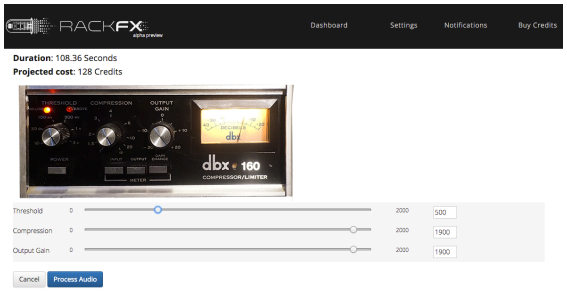

Figure 4. RackFX uploaded audio file waveform

Once the process audio button has been clicked, the user is taken back to a page showing their uploaded waveform (audio file to be processed) with a message dialog pane to the bottom right of the waveform reporting the status of the audio processing. The first step in processing is to add the processing job to a queue. (see Figure 5). The queue is a node.js application running on the RackFX server that handles the scheduling of all requests for processing received from the internet.

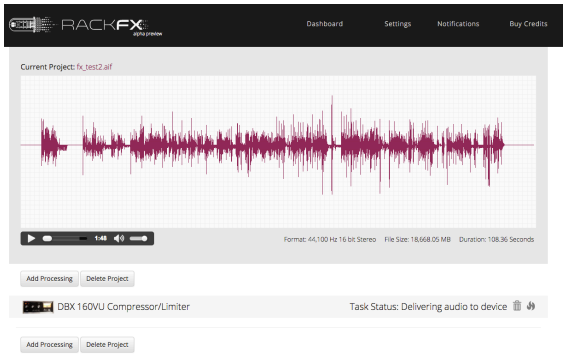

Figure 5. RackFX uploaded audio file waveform

Once the user's file has been processed and uploaded to

the server as a new file, having waited only for the device to become available (if another user is currently active) and for the audio to process in realtime, the RackFX current project page will update, show the new file's waveform, and provide a download link to the analog-effected digital file.

2.2 The RackFX Platform Behind the Scenes

On the server-side for any RackFX processing job, the web-app proxy receives all internet requests and forwards them into a web-app cluster written in node.js. The web-app cluster interfaces with Amazon S3 for storage and a MySQL database handling all site data. The queue software, a light weight file-based JavaScript Object Notation (JSON) queuing service, runs on the web-app cluster. When a job is submitted for processing, the queue schedules it, waits for the targeted analog device to become available and then passes each job, when ready, to be processed one at a time. The queue passes the job specifics to a Messaging Application Programming Interface (MAPI) cluster (also written in node.js), which messages the Machine Device Controller (MDC) software running on an OSX machine in close physical proximity to the actual analog hardware units.

At this point, the processing request has moved from the server (web-app) to the actual device-side of the processing system. The MDC (also written in node.js) orchestrates the actual processing of the (still) digital audio in the following order: (1) identify the device selected for processing; (2) switch on electrical power to the given analog hardware unit and an arduino interfacing with the device using device-specific robotic components; (3) pass device parameters to the arduino (using Johnny-Five, a JavaScript robotics module for node.js); (4) wait for the robotics to physically interact with the analog hardware device and set all parameters; and (5) spawn a Cycling74 Max application (that we call "max-io") that handles the realtime digital audio playback and capture of the analog-effected signal. Once the roundtrip signal i/o is complete, max-io tells the MDC that the new file has been written, the MDC cleans up: uploads the new file to the server, signals completion, and shuts all on-site devices down. A visual overview of the whole RackFX system, including major components and signal flow, is shown in Figure 6.

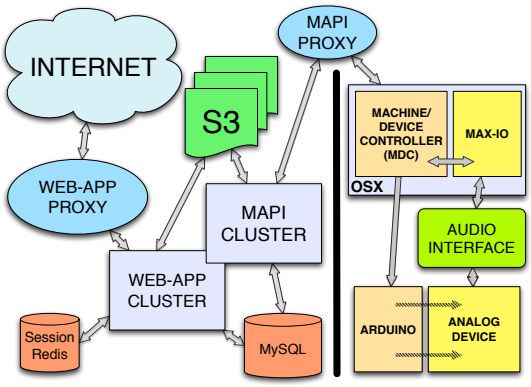

Figure 6. RackFX system overview

3. ROBOTIC DEVICE INTERFACE SPECIFICS

In order to maximize the automatization of the interaction between the user (client-side) and the analog device (server-side), RackFX aims to outfit each analog device with custom robotics that physically interact with the device's particular control panel. Various models of stepper motors, acctuators, and sensors combine to create each hardware interface between machine and device. These robotic components are controlled using a dedicated arduino board for each device (see Figure 7).

Figure 7. Robotic hardware assembly mounted on a Fender Princeton Reverb amp

Each arduino is loaded with the Advanced Firmata firmware (a protocol for communicating with microcontrollers from software on a host computer) and addressed by its host computer using the JavaScript client library Johnny-Five (J5), a robotics and Internet of Things programming framework. The node.js MDC loads the J5 module to enable communication between the MDC and each device. When the MDC goes to process a job (the next job in the queue), the program identifies the arduino associated with the specified analog device, instructs all stepper motors to reset (one at a time) by (over-)turning all knobs counter-clockwise a full rotation to ensure the analog device potentiometers are set to zero. The MDC then instructs each stepper to turn a certain number of steps commensurate to the parameter setting specified by the user through the web-app. (Maximum and minimum step values are tuned in relation to each physical parameter setting for each device, as part of the configuration process.) After a short delay to ensure all parameters are set, the MDC communicates with max-io to commence audio processing.

4. AUDIO HANDLING SPECIFICS

The Max applictation, named max-io in this project, handles all of the digital audio playback, routing in and out of the outboard analog device, and the digital re-capture of the processed audio. Max-io is designed to be as transparent as possible regarding the digital source and returned object for each processing job. Furthermore, communication between the Machine Device Controller (MDC), which orchestrates the processing of each job, and max-io follows a very specific messaging protocol using Open Sound Control (OSC) messages.

Max-io requires the MDC to provide seven parameters for each job to ensure successful completion. They are as follows: (1) number of channels for the audio signal, (2) file path to the digital input file, (3) file path for the analog-effected output file, (4) sample format (int8 up to float32), (5) Output file FX tail duration in milliseconds, (6) roundtrip audio latency compensation in milliseconds, and (7) which audio interface output and input channel(s) to use (i.e. which channel is physically routed to the appropriate analog device). Given the successful reception of each of the above parameters, max-io loads the input file into RAM, and allocates the appropriate memory (given the FX tail and sample format) to record the return output file. When all is set, max-io plays back the specificed digital audio routed to the appropriate [dac] channel, while simlutaneously starting recording on the appropriate incoming [adc] channel. Neither the amplitude of the outgoing digital signal, nor the amplitude of the incoming analog signal is adjusted. When playback is complete, the audio buffer containing the analog-effected audio is trimmed, given the latency compensation parameter, and the file is writen to disk with correct specified sample format.

Furthermore, given the specifics of the system– the software and hardware resources of the PC and the audio interface hardware connected to it–the MDC can adjust DSP parameters for each job by interfacing with max-io. For instance, different interfaces may be selected, along with different signal vector sizes and sampling rates. This flexibility and customizability built into the ground floor of the RackFX system makes it possible to potentially run this automated system on a variety of machines with different limitations interfacing with different audio gear.

5. FUTURE DEVELOPMENTS

Future development using the RackFX platform is focused on not only extending the device offerings for the analog processing of any given job, but also providing users with the ability to preview the analog audio effect, ideally by routing a small portion of audio through the device to test the current parameter settings before processing the whole file. User ability to interact with the web GUI such that they may turn the appropriate virtual knobs and preview the effects of different parameter settings is highly desirable and would make the platform even more useful to the non-expert engineer or musician looking to experience the possibilities of analog processing.

Furthermore, while the on-site facilities supporting the RackFX platform are steadily growing in the number of available analog devices, it is also possible for the RackFX platform to be backed by a distributed network of device providers– partners existing in multiple physical locations sharing their own analog devices, making their devices available to internet users through the RackFX web-app. The notion of scalability here is particularly interesting and encouraging because once affiliates are provided with the necessary software (MDC + max-io) and the robotics hardware to mount onto their particular analog device(s), the RackFX platform could grow to allow individuals and professional studios alike to share their analog hardware resources.

6. CONCLUSIONS

The RackFX platform allows users to access analog equipment through an easy to use web site. This platform allows users to use audio processing equipment through cloud-based technology and robotics. And in the future, studios and individuals can bring their devices to the community and become a RackFX partner, bringing analog processing capability to users around the world through our easy-to-use custom framework.

Ultimately, RackFX represents an opportunity for musicians and audio producers to engage in the analog processing of sound through the web. In the past, low-budget musicians, video producers, music producers and podcasters had to rely on increasingly expensive digital plugins that attempt to emulate analog signal processing devices, or they had to invest in cheap analog gear with low-quality components in an attempt to achieve the sound qualities they associate with high-budget studio analog gear. Now users can have access to this high-end equipment through the RackFX platform.

As a digital music solution, the RackFX project simply refuses to pick sides in the analog versus digital signal processing debate. While our commitment to achieving ever more refined in-the-box DSP techniques and analog device emulations will continue, we should not be dogmatic here; we should not think that *parity* between the digital and analog world is either necessary or desireable. Nor should we eschew what digital tools have afforded in the name of maintaining limited access to analog processing units– resulting in analog fetishization to an even greater degree, given such a scarce resource. By leveraging a host of digital technologies, including cloud computing, real-time digital audio manipulation, and robotics, the RackFX platform provides an alternative path: make analog devices accessible through the web to empower all musicians, regardless of budget. At the very least, our psychoacoustic value judgements regarding the 'warmth' and 'presence' of analog processing effects will be put to the test now that analog gear is no longer cloistered. Ideally, a platform like RackFX will help advance our ability to hear.

7. REFERENCES

[1] M. Karjalainen and J. Pakarinen, "Wave Digital Simulation of a Vacuum-Tube Amplifier," *ICASSP*, vol. 2, pp. 153–156, 2006.

[2] "Waves Audio Analog Models Plugins," http://www.waves.com/plugins/analog-models, accessed: 2-29-2016.

[3] J. O. Smith III, "Physical Modeling Synthesis Update," *Computer Music Journal*, vol. 20, no. 2, pp. 44–56, 1996.

[4] "Silophone project," http://www.silophone.net/eng/about/desc.html, accessed: 4-20-2016.

[5] "Patchwork: Control a Massive Modular Synthesizer," http://synth.media.mit.edu/patchwerk/, accessed: 4-20-2016.