Daria: A New Framework for Composing, Rehearsing and Performing Mixed Media Music

Guillermo Senna[1] and Juan Nava Aroza[2] *

[1] Universidad Nacional de Rosario
[2] Universidad Nacional de Rosario

Abstract. In this paper we present a new modular software framework for composing, rehearsing and performing mixed media music. By combining and extending existing open-source software we were able to synchronize the playback of the free Musescore music notation editor with three VST audio effects exported using the Csound frontend Cabbage. The JACK Audio Connection Kit sound server was used to provide a common clock and a shared virtual timeline to which each component could adhere to and follow. Moreover, data contained on the musical score was used to control the relative position of specific Csound events within the aforementioned timeline. We will explain the nature of the plugins that were built and briefly identify the five new Csound opcodes that the development process required. We will also comment on a generic programming pattern that could be used to create new compatible VST audio effects and instruments. Finally, we will conclude by mentioning what other related software exists that can interact out-of-the-box with our framework, how instrument players and computer performers can simulate the performance experience while practicing their corresponding parts at home and what our future plans for this software ecosystem are.

Keywords: Mixed media music, Musescore, Csound, Cabbage, JACK.

1 Introduction

The aim of this paper is to present a new modular framework for composing, rehearsing and performing mixed media music. The pedagogical method presented by Lluán et al. [1] inspired us to build a new open-source alternative to their proposed system. By modifying and extending the free Musescore music notation editor [2], as well as by designing a series of VST effects with the Csound frontend Cabbage [3,4], we were able to build a more open and flexible substitute. The JACK Audio Connection Kit sound server [5] was used for providing a common clock and a shared virtual timeline and the software Carla [6] was chosen as our preferred VST host. As a result every component of this project is not only open-source, but also multiplatform.

* We would like to gratefully thank Professor Claudio Lluán for his continued support and guidance.

A short musical fragment and three new VST audio effects were created for demonstrating the capabilities of our system. Although not yet available for its first stable release, all the code that pertains to this project, including forked and modified third-party repositories, is now publicly available [7].

The synchronization between the music notation editor and Csound was achieved by developing a new Musescore plugin and a single VST (the *Conductor*) for controlling the transport part of the audio server. The two other VST audio effects mentioned earlier will serve us to showcase a common programming pattern for triggering Csound events that are required to be kept in sync with the shared timeline.

2 A Brief Overview of the Plugins

2.1 The Musescore Plugin

A new Musescore plugin was developed using the QML markup language. When called, this plugin first inspects the musical score and stores the time cues where each new measure starts. The resulting collection of values is sent to the *Conductor* using the OSC protocol.

For testing purposes, we used the musical score represented in Figure 1. The synchronization between the music notation editor and the *Conductor* plugin is considered essential for this system to work. Consequently, this communication should not be omitted while working on new musical pieces.

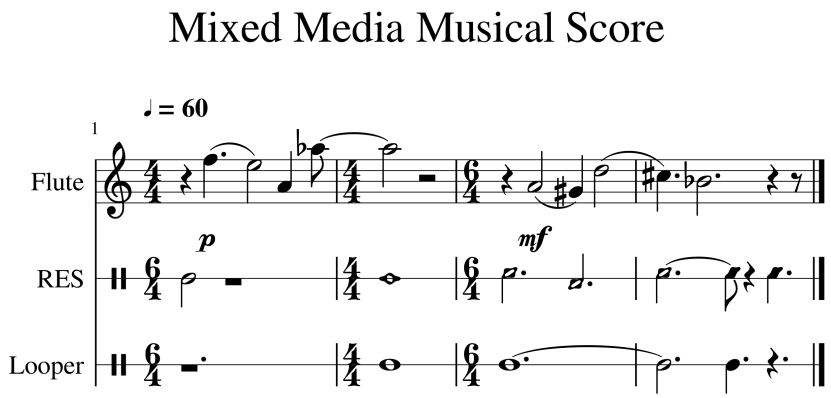

Fig. 1. A short musical fragment used to test our system.

A Javascript function is also included in the QML plugin for the composer and/or computer performer to parse, extract and send specific data to other

listening VSTs. In our particular scenario the QML plugin was programmed to transmit two additional arrays, both of which were constructed from information contained inside the musical score. While the first array was sent to the *Random Electronic Sounds (RES)* VST, the second one was directed towards the *Looper* VST.

As can be seen in Figure 1, we decided to include this data inside two distinct Musescore instruments. By following this approach we were able to represent the start and end time, as well as a few other parameters required by the Csound instruments, simply by using traditional music notation.

2.2 The *Conductor* VST

The *Conductor* is a VST audio effect plugin exported from Cabbage. It allows the electronic performer (or the acoustic instrument player in case of practice sessions and rehearsals) to rewind, stop, start, pause and advance the playback cursor; modify and visually keep track of a countdown timeline; be aware of the transport position of JACK in relation to the measures and beats contained in the musical score; and check the total duration of the piece as well as the time of the current playback position.

Fig. 2. The *Conductor* VST.

The construction of this VST demanded the development of two new Csound opcodes: `jackquery`, for controlling the JACK transport; and `floorarray`, for conducting a binary search among the array containing the time cues. The Cabbage source code also had to be modified in order to dynamically display bar lines and numbers inside its *image* widget [8].

2.3 The *Random Electronic Sounds (RES)* VST

The *RES* is a VST audio effect plugin that plays back randomly selected audio files while keeping them in sync with the shared clock provided by JACK. After receiving an array of numeric values from the Musescore plugin, the aforementioned files are laid out on a virtual timeline that corresponds to the rhythmic figures written on the musical score. Moreover, when the user repositions the transport through the *Conductor* (or by means of an external application) the *RES* updates its internal state to reflect the newly assumed position.

We believe the programming pattern used for achieving this kind of functionality to be relevant to composers and/or computer performers while they try to design new VSTs that are meant to be used with this framework. In consequence, we will further explain this technique in 2.4.

The Graphical User Interface (GUI) of this plugin is composed of a button to scan (or later, rescan) the folder containing the audio files; a virtual LED for indicating the status of the plugin; a bypass button for disabling the effect; a rotary knob for adjusting the pan position; a vertical slider for controlling the output level; and a virtual VU Meter to check the levels present at the output.

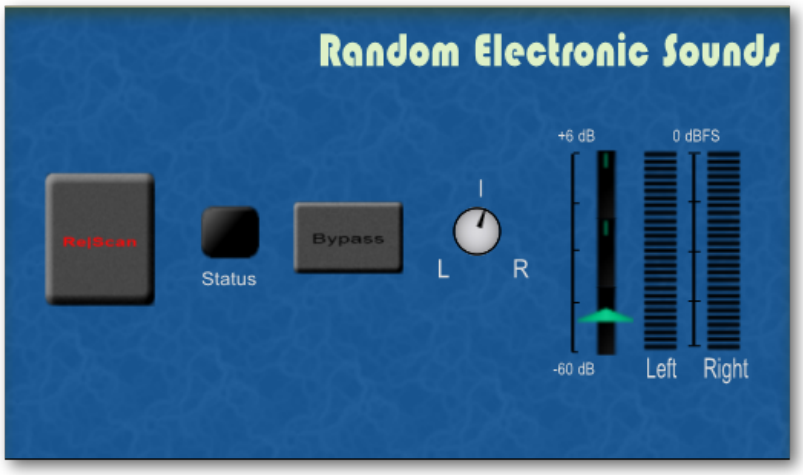

Fig. 3. The *Random Electronic Sounds (RES)* VST.

The *RES* VST required the development of three new Csound opcodes: `filewalker`, a recursive file iterator; `flnarray`, for querying the lengths of a collection of audio files; and `erandom`, for randomly choosing files depending on a list of parameters received by the plugin.

2.4 The *Looper* VST

The *Looper* is conceptually the simplest of the VSTs presented in this paper. After initialization, it expects to receive an array of time cues through an UDP port. The first pair of values triggers a recording instrument, while the remaining subsets of the array activate different instances of a same instrument that plays back what has been previously recorded.

Its GUI consists of two vertical sliders for controlling the gain of the input present at channel 1 and channel 2; a mono VU Meter for each input; two rotary sliders for modifying the pan position of each channel; a color-coded LED for checking the status of the plugin; a bypass button; a vertical slider for adjusting the level of the output; and a stereo VU Meter for controlling the levels present at the output.

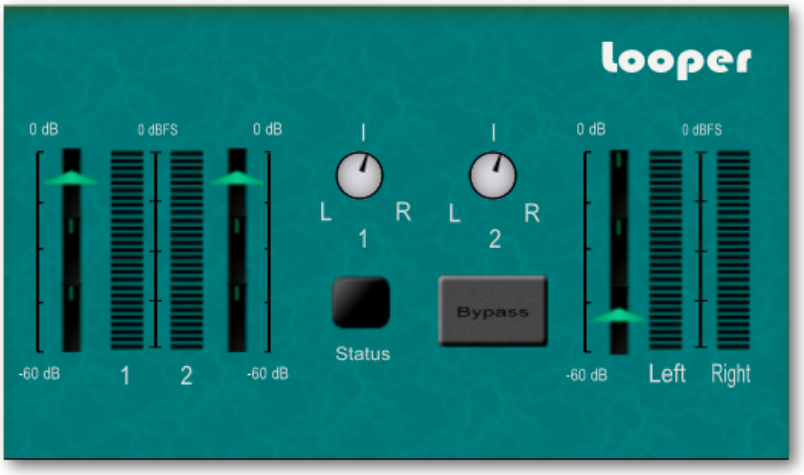

Fig. 4. The *Looper* VST.

As previously stated, a particular programming pattern was used in the *Looper* -as well as it was in the *RES*- that can be further utilized while developing new plugins. By using this technique we can control the triggering process of any Csound instrument that is thought to be kept in sync with the transport.

Conceptually, the design pattern can be explained in terms of a chain of instruments that are sequentially called upon. In this chain, the first instrument -from here on the *listener*- is prepared to receive an array of values through the OSC protocol. Upon success, the *listener* calls a new instance of another instrument (the *watchdog*) for every note written on the relevant instrumental part of the musical score.

Each *watchdog* will be in charge of keeping track of the external time in terms of samples and seconds, updating the corresponding Csound control vari-

ables and channels throughout the performance. Later, whenever the instantaneous time position of the transport falls in between the correct time interval, a *performer* instrument will be triggered. It is important to note that an automatically re-triggering process will also occur whenever a repositioning of the transport has taken place.

The *performer* is the last component of our chain and also the Csound instrument in charge of handling the audio processing task we are ultimately trying to accomplish. This means that the sole purpose of the other two instruments is to relay the information coming from the notation editor and also to enforce a synchronism between the performer and the external transport, but by encapsulating each task in a different Csound instrument we ensure a design pattern that can be generically applied to any new VST effect.

3 Conclusions

The framework presented in this paper provides a novel alternative for producing mixed media music. Being modular by nature, all Cabbage effects and instruments already created by the community can be (asynchronously) used and new ones could also be developed to perform in sync with the musical score.

By using MIDI and/or audio automation tracks -both functions provided natively by Carla- the acoustic instrument player and/or the electronic performer can both individually study the musical piece, transforming a practice session at home into a full rehearsal. Moreover, software like Xjadeo [9] could be used to perform live mixed media music perfectly in sync with the playback of a video.

In the near future we expect to test this new framework in an educational environment. We hope that this kind of live experience will enable us to gather feedback and this, in turn, will subsequently help us improve the system to meet the demands of future mixed media music composers.

References

1. Lluán, C. et al.: A theoretical and aesthetics approach to the study and practice of mixed Electroacoustic Music: a pedagogical proposal. In: Electroacoustic Music Studies Network EMS 09: Heritage and Future. Buenos Aires (2009)
2. Musescore website, `https://musescore.org`
3. Walsh, R.: Developing Csound Plugins with Cabbage. In: Ways Ahead: Proceedings of the First International Csound Conference, pp. 64–82. Cambridge (2013)
4. Cabbage website, `http://cabbageaudio.com`
5. JACK Audio Connection Kit website, `http://jackaudio.org`
6. Carla Github repository, `https://github.com/falkTX/Carla`
7. Daria Github repository, `https://github.com/gsenna/Daria`
8. Cabbage Docs: the image widget, `http://cabbageaudio.com/docs/image`
9. Xjadeo website, `http://xjadeo.sourceforge.net`