Rethinking the audio workstation : tree-based sequencing with i-score and the LibAudioStream

Jean-Michal Celerier
Affiliation1
author1@smcnetwork.org

Myriam Desainte-Catherine
Affiliation2
author2@smcnetwork.org

Stphane Letz
Affiliation3
author3@smcnetwork.org

ABSTRACT

Place your abstract at the top left column on the first page. Please write about 150–200 words that specifically highlight the purpose of your work, its context, and provide a brief synopsis of your results. Avoid equations in this part.

1. INTRODUCTION

Progression du papier -¿ dfinitions : audio sequencing, workstations, etc. -¿ poser le problme -¿ dfinitions et prsentation des outils -¿ extensions de la libaudiostream -¿ traduction de i-score en expression libaudiostream

2. EXISTING WORKS

- Audiographe, etc. : Jamoma Audiograph, Logic - Ableton Live - Reaper

3. CONTEXT

In this section, we will present the two tools that are used to achieve rich audio sequencing : i-score and the LibAudioStream. i-score is an interactive sequencer which allows to position events in time, and gives the possibility to introduce interaction points and conditions in the score. The detailed execution semantics are given in [?].

The LibAudioStream provides the ability to write rich audio expression by creating and combining streams. The notion of symbolic date, introduced in allows to start and stop the execution of streams at an arbitrary date.

cite

3.1 Description i-score

- interactivit -¿ mapping and JS -¿ donner example scnario i-score

3.2 Description LibAudioStream

-¿ donner smantique des flux. -¿ players -¿ donner example flux stream avec effet faust.

4. AUDIO AUTHORING IN I-SCORE

5. PRESENTATION OF THE AUDIO SYSTEM

In this section, we will explain the audio routing features offered by the software.

First, we introduce new audio streams that allow a LibAudioStream expression to encapsulate the execution of a virtual audio player, in order to allow for hierarchy.

We make the choice to allow for hierarchy by mixing the played streams together.

explique
pourquoi

Then, we present the concept of audio buses integrated to the LibAudioStream, with two special Send and Return streams.

5.1 Group audio stream

In order to be able to apply hierarchical effects on the streams, we have to introduce a way to use hierarchy in the LibAudioStream.

Two elements are necessary :

- A particular audio player that will be able to sequence the starting and stopping of interactive sounds. Such players already exist in the LibAudioStream but are tailored for direct output to the soundcard.
- A way to introduce the Renderer into the stream system, so that it is able to be pulled like it would be by a real soundcard.

We introduce two new objects in the LibAudioStream :

- A Group player. This is a player whose processing function has to be called manually. Timekeeping supposes that it will be pulled in accordance at the clock rate and samplerate of the soundcard.
- A Group audiostream. This particular audiostream, of infinite length, allows to introduce a Group player in a series of chained streams. For instance : (mix (fx aPlayer an

The execution time of the hierarchized objects will be relative to the start time of the Group audiostream. The sound of the objects will be mixed together.

5.2 Send and return audio streams

In order to be able to create rich effect graphs, we introduce another couple of objects working in tandem.

The Send audiostream - Flux infinis - ncessit de tirer les flux; il est possible de faire un adapteur ?

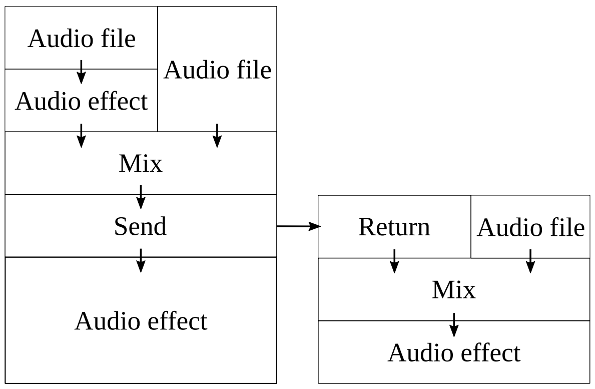

Figure 1. Possible mixing with the Send and Return objects. An arrow from A to B means that B pulls the audio data from A.

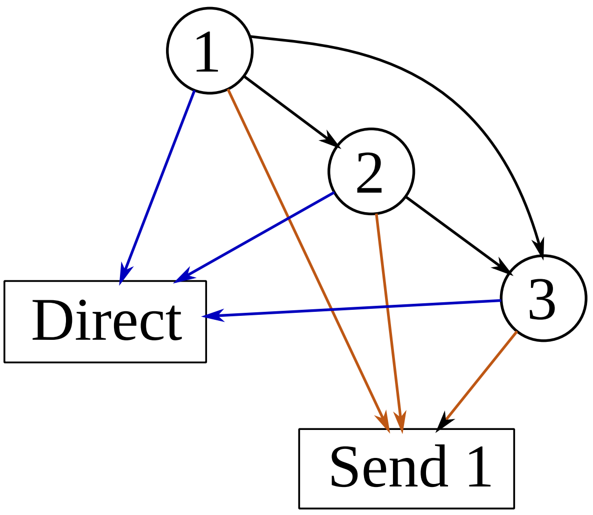

Figure 2. Translation of the Time Constraint of fig. TODO in a graph. The edges in black represent the intra-Constraint connections. The edges in blue (resp. orange) represent a connection to a visible output of the Constraint.

5.3 Stream graph

One problem caused by the presence of routing is that it is possible to have a data loop : if a Send is fed its own data through a Return, the output would be garbage data : the Return would be asked to read the currently requested output from the Send which has not been written yet.

To prevent this, we create a graph where :

- Vertices are the sound generating elements : audio file reader, etc.
- Edges are the connections going from a send to a return.

CITE

The graph, implemented with the Boost.Graphlibrary can then be used to guarantee the acyclicity, and return an user error if that is not the case.

We provide here the algorithm to build the graph.

Disambiguate the user-created sends and the sends required for the model

Vertices are created recursively from the Time Constraints in i-score. First, we iterate through all the processes of the given constraint. If the process is hierarchical (Scenario, Loop), then we call the algorithm recursively on the process. In the case of the Scenario, it means that we call recursively on all its Constraints; in the case of the Loop, we call recursively on its loop pattern Constraint. In both case, we create a Group vertice to model the process. Then, we create inner Sends for all the streams and a vertice for the Constraint itself.

As can be seen, there can be an ordering between nodes of the graph : the parentmost vertice has to be pulled before the others to reflect the causality.

Comparer Ableton ou il est possible pour chaque piste de choisir son entre et

Inside a Time Constraint, causality also has to be enforced. Since a mixing bus is provided, we have to ensure that an effect bus cannot be routed in itself in a loop. To prevent this at the user interface level, the vertical order of effect chains is used : if Effect Chain 1 comes before Effect Chain 2, then Effect Chain 1 can be routed into Effect Chain 2, but not the contrary.

5.4 Stream creation

-¿ dire qu'on connecte chaque StartStream et chaque StopStream un timenode.

5.4.1 Scenario

5.4.2 Loop

5.4.3 Time Constraint

-¿ donner smantique de flux des Stream Group, Send, Return.

- Comme la dure de chaque contrainte peut varier avec le ralentissement, on utilise principalement des dates symboliques -¿ Processus audio dans i-score : - FX =¿ Supports FaUST. - Instrument. - Send. - Return. - Mixing. - Hirarchie profondeur arbitraire. - Automations : exporte les paramtres dans le modle d'objet

Faire graphe pour une Time Constraint et donner un exemple avec effets appliqus sur scnario. Expliquer graphe hirarchique de dpendances : penser au cas ou un a un return dans une hirarchie puis un send un niveau suprieur; il faut faire le grpahe de A Z et s'assurer qu'il ne soit pas cyclique

1er cas : Un son avec une piste d'effets.

2eme cas : scnario hirarchique, boucle

Cas de la boucle avec un coup A, un coup B selon la condition ? -¿ excution d'un timenode doit reset le flux.

Piste send / return : permet de maintenir les queues de reverb.

5.5 Routing, multi-channels, etc.

-¿ mettre maquettes track mix

6. EXAMPLE

6.1 UI

-¿ capture d'cran Faire vue scnario et sa traduction en graphe de routage

6.2 Temporal effect graphs

7. CONCLUSION

-¿ lackluster areas : - MIDI support (but OSC) - no musical time information : first aimed for artists, but improvements could be the waiting of triggering on some measure of time. - ”play from anywhere” - audio input ? - correction de latence ? - pour l’instant pas d’optimisations dans des cas simples (e.g. pas besoin de mixage)

Acknowledgments

ANRT, Blue Yeti, Magali

8. REFERENCES

[1] A. Someone, B. Someone, and C. Someone, “The title of the conf. paper,” in *Proc. Int. Conf. Sound and Music Computing*, Porto, 2009, pp. 213–218.

[2] X. Someone and Y. Someone, *The Title of the Book*. Springer-Verlag, 2010.

[3] A. Someone, B. Someone, and C. Someone, “The title of the journal paper,” in *J. New Music Research*, 2008, pp. 111–222.