

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

Jean-Michel 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

Software audio sequencers are generally considered to be digital versions of the traditional tools that are used in a recording studio : tape recorders, mixing desks, effect racks. . .

Most of the existing software metaphors follow this very closely, with concepts of tracks, buses, linear time, which are a skeuomorphic reinterpretation of the multi-track tape recorder [1]. At the other side of the music creation spectrum, entirely interaction-oriented tools, like Max/MSP, Pure Data, Csound, or SuperCollider, allow to create musical works in programming-oriented environments. In-between, one can find tools with limited interaction capabilities but full-fledged audio sequencing support, like Ableton Live, or Bitwig Studio. The interaction lies in the triggering of loops and the ability to change the speed on the fly but is mostly separate from the "traditional" sequencer.

In this paper, we present a new tree-based approach to interactive audio sequencing. By exposing the LibAudioStream audio engine to the interactive control sequencer i-score, we provide an audio sequencing software that allows to author music in a timeline with the possibility to arbitrarily nest sounds and effects, trigger sounds interactively while keeping the logical coherency wanted by the composer, and arrange audio effects in a temporal graph.

We will first present the existing works in advanced audio sequencing and workstations, and give a brief presentation of both i-score and the LibAudioStream. Then, the new objects introduced in order to integrate these software together, allowing for rich audio routing capabilities, will be explained. Finally, examples of usage in the graphical interface of i-score will be provided.

## 2. EXISTING WORKS

- Audiographe, etc. : Jamoma Audiograph, Logic - Ableton Live, Bitwig - Reaper - DRILE - Trackers (FastTracker, ...) - Reprendre article Bell

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

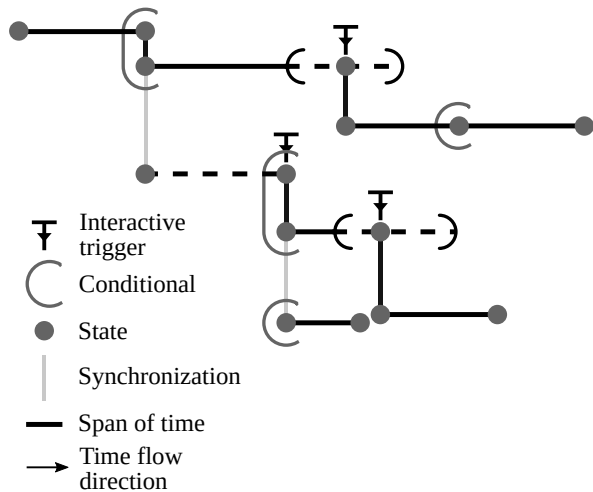
### 3.1 Description i-score

i-score is a sequencer for parameters.

Its main use is to communicate and orchestrate other software in a timely manner, through the OSC protocol. The software can send automations, OSC messages at a given point in time, and call arbitrary JavaScript functions, in a sequenced environment. It supports arbitrary nesting : a score can be embedded in another recursively. This is similar to the notion of group tracks in many other sequencers, but there is no limit of depth. Besides, there is no notion of "track" per se; rather, the composer works with temporal intervals which contains arbitrary data that can be provided by plug-ins.

There are multiple possibilities of interactivity in i-score : trigger points, conditions, mappings.

- Interactive triggers allow to block and synchronize the score until a specific event happens. For instance, when an OSC parameter fulfills a condition, such as `/a/b >= 3.14`, then a part of the score can continue.
- Conditions allow to execute or disable part of the score according to a boolean condition. It makes if-then-else or switch-case programming construct easy to implement in a temporal way.
- Mappings allow to map an input parameter to an output parameter, with a transfer function applied to the input.



**Figure 1.** Part of an i-score scenario, showcasing the temporal syntax used. A full horizontal line means that the time must not be interrupted, while a dashed horizontal line means that the time of this Constraint can be interrupted to proceed to the following parts of the score according to an interactive event.

A span of time in i-score might have a fixed or indefinite duration; we refer to this span as a Time Constraint since it imposes both a logical and temporal order to the elements before and after it. An example of the temporal syntax of i-score is presented in fig. 1.

This span of time may contain data by the form of processes : automations, mappings, but also loops and scenarios; a scenario is the process that allows nesting.

### 3.2 Description LibAudioStream

-i donner smantique des flux. -i players -i donner exemple flux stream avec effet faust.

## 4. AUDIO AUTHORIZING 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. This is done in accordance with the principle of least astonishment [3] for the composer : in most audio software, the notion of grouping implies that the grouped sounds will be mixed together and routed to a single audio bus.

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 matching 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.

The Send audiostream by itself is a passthrough : it just pulls the stream it owns. It possesses the same definition : same length, same number of channels... The Return audiostream, constructed with a Send stream, will make a copy of the data in the send stream and allow it to be used by the processing chain it is part of. This means that a sound source can be sent to sound effects in parallel, for instance.

The Return stream is infinite in length : to allow for long-lasting audio effects like reverb queues or delays, we suppose that we can pull the data from the Send stream at any point in time. If a Return stream tries to fetch the data of a Send stream that has not started yet, or that has already finished, a buffer of silence is provided instead.

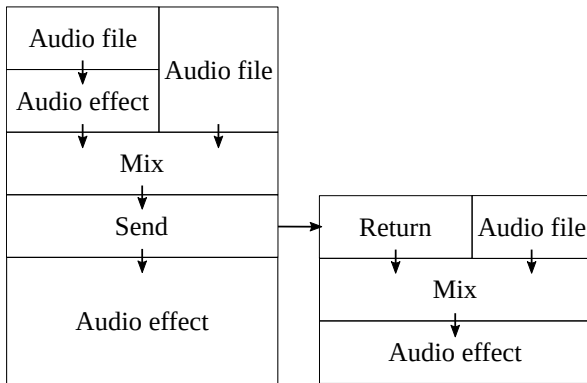
A byproduct of the imposed acyclicity of the graph, some level of parallel processing may be achieved, as long as causality is respected between nodes. For instance, if two Returns with each their own effect chain are connected to a Send, it is possible to compute these returns on different threads after the current buffer of the Send has been processed.

An important point is that the Send stream must itself be pulled regularly by being played as a sound, either directly or by a stream that would encapsulate it.

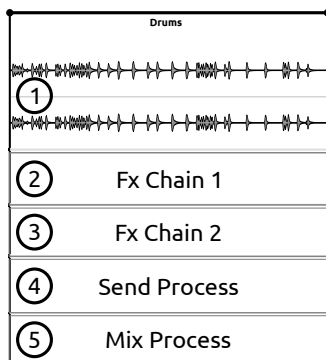
#### 5.3 Audio processes

We provide multiple audio processes in i-score, that map to lower-level audio streams.

- Effect chain process : register multiple audio effects one after the other. For instance : *Equalizer* → *Distortion* → *Reverb*. Currently only Faust effects are supported.



**Figure 2.** Possible mixing with the Send and Return objects. An arrow from A to B means that B pulls the audio data from A.



**Figure 3.** An example of a Time Constraint loaded with audio processes in i-score. Selecting a particular process shows a complete widget for editing the parameters.

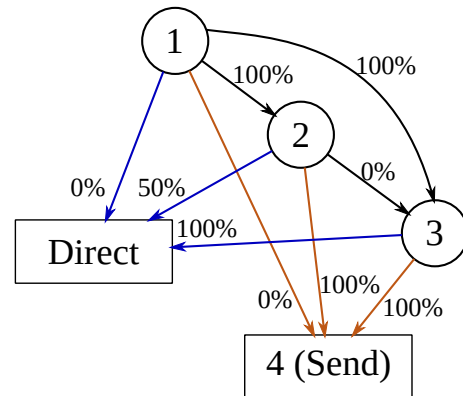
- Generator process : a sound generator, like a synthesizer. Currently only Faust instruments are supported.
- Sound file.
- Explicit send and return processes.
- Mixing process.

An important feature of audio workstations is the support for automation, that is, controlling the value of a parameter over time, generally with piecewise continuous functions. In i-score automation is generally achieved by sending OSC messages to a remote software; the OSC message tree is modeled as an object tree. We introduce the loaded effect plug-in to this object tree, so that automations and mappings are able to control audio effects and audio routing parameters such as volume.

An example is given in fig. 3.

## 5.4 Stream graph

One problem caused by the presence of routing is that it is possible to create a data loop : if a Send is directly or indirectly fed its own data through a Return, the output would be garbage data : the Return would be asked to read



**Figure 4.** Translation of the Time Constraint of fig. 3 in a dependency 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. The percentages represent the level of mixing of the stream. *Direct* corresponds to the signal that will be sent at the upper level of hierarchy.

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.

The graph, implemented with the Boost Graph Library [4] 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.

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 vertex to model the process. Then, we create inner Sends for all the streams and a vertex for the Constraint itself.

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

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.

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

Compare Ableton ou il est possible pour chaque piste de choisir

## 5.5 Stream creation

### 5.5.1 Scenario

An i-score scenario is an arrangement of temporal structures, as shown in fig. 1. Since the duration of these structures can change at run-time, it is not meaningful to use the tools provided by the LibAudioStream : sequence stream, parallel stream, mix stream.

1. A Group player is created.
2. For each Time Node in the scenario, a symbolic date is generated.
3. For each Time Constraint in the scenario, a stream is built; it is started and stopped at the symbolic date matching its start and end Time Nodes in the group player.

The Audio stream of this process is the group player.

### 5.5.2 Loop

Due to their interactive nature, loops in i-score can be entirely different from one iteration to another; they are more similar to traditional programming language `do-while` constructs, than audio sequencer loops.

- problème des boucles dans i-score : chaque itération peut être différente.

### 5.5.3 Time Constraint

audio-graphe donner sémantique de flux des Stream Group, Send, Return.

Comme la durée de chaque contrainte peut varier avec le temps, on utilise principalement des dates symboliques.

Faire graphe pour une Time Constraint et donner un exemple simple avec effets appliqués sur scénario. Expliquer graphe hiérarchique de dépendances : penser au cas où un a une relation dans une hiérarchie puis un send à un niveau supérieur; il faut faire le graphe de A à Z et s'assurer qu'il ne soit pas cyclique.

Cas : Un son avec une piste d'effets.

2ème cas : scénario hiérarchique, boucle

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

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

## 5.6 Routing, multi-channels, etc.

-> mettre maquettes track mix

## 6. EXAMPLE

### 6.1 UI

-> capture d'écran Faire vue scénario 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. Bell, E. Hein, and J. Ratcliffe, "Beyond skeuomorphism: The evolution of music production software user interface metaphors," 2015.
- [2] S. Letz *et al.*, "The libaudiostream library," 2012.
- [3] P. Seebach, "The cranky user: The principle of least astonishment," 2001.
- [4] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual, Portable Documents*. Pearson Education, 2001.