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

Jean-Michaël Celerier
LaBRI, Blue Yeti
Univ. Bordeaux, LaBRI, UMR 5800,
F-33400 Talence, France.
Blue Yeti, F-17110 France.
jcelerie@labri.fr

Myriam Desainte-Catherine
LaBRI, CNRS
Univ. Bordeaux, LaBRI, UMR 5800,
F-33400 Talence, France.
CNRS, LaBRI, UMR 5800,
F-33400 Talence, France.
INRIA, F-33400 Talence, France.
myriam@labri.fr

Jean-Michel Couturier
Blue Yeti, F-17110 France.
jmc@blueyeti.fr

ABSTRACT

The field of digital music authoring provides a wealth of creative environments in which music can be created and authored: patchers, programming languages, and multi-track sequencers. By combining the I-SCORE interactive sequencer to the LIBAUDIOSTREAM audio engine, a new music software able to represent and play rich interactive audio sequences is introduced. We present new stream expressions compatible with the LIBAUDIOSTREAM, and use them to create an interactive audio graph: hierarchical stream and send - return streams. This allows to create branching and arbitrarily nested musical scores, in an OSC-centric environment. Three examples of interactive musical scores are presented: the recreation of a traditional multi-track sequencer, an interactive musical score, and a temporal effect graph.

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, etc.

Most of the existing software follow this paradigm very closely, with concepts of tracks, buses, linear time, which are a skeuomorphic reinterpretation of the multi-track tape recorder [1]. On the other side of the music creation spectrum, we find entirely interaction-oriented tools, such as MAX/MSP (Cycling 74'), PUREDATA, CSOUND, or SUPERCOLLIDER. They allow the user to create musical works in programming-oriented environments. In-between are tools with limited interaction capabilities but full-fledged audio sequencing support, such as 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" sequencers integrated in these software packages.

In this paper, we present a graphical and hierarchical approach to interactive audio sequencing. We integrate the LIBAUDIOSTREAM audio engine in the interactive control sequencer I-SCORE. This takes form as an audio sequencing software package that allows the user to author music on a time-line with the possibility to arbitrarily nest sounds and effects and trigger sounds interactively while keeping the logical coherency specified by the composer. An extension is introduced to arrange audio effects in an interactive temporal graph. For instance, instead of simply applying a chain of effects to an audio track, it is possible to apply temporal sequences of effects: an effect would be enabled for ten seconds, then, if an external condition becomes true, another effect would be applied until the musician chooses to stop it.

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 packages together, allowing for rich audio routing capabilities, will be explained. Finally, three examples of usage in the graphical interface of I-SCORE will be provided: a recreation of a standard multi-track player, an interactive score, and an effect graph applied to a sound.

2. EXISTING WORKS

Outside of the traditional audio sequencer realm, there are multiple occurrences of graphical environments aiming to provide some level of interactivity.

Möllenkamp presents in [2] the common paradigms used when creating music on a computer: score-based with MUSIC and CSOUND [3], patch-based with MAX/MSP (Cycling 74') or PUREDATA [4], programming-based with SUPERCOLLIDER [5] and many of the other music-oriented programming languages, trackers such as FASTTRACKER which were used to program the music in early console-based video games, and multitrack-like such as Steinberg CUBASE, Avid PRO TOOLS. Ableton LIVE and Bitwig STUDIO are given their own category thanks to the ability to compose clips of sound interactively.

DRILE [6] is a virtual reality music creation software package. Loops are manipulated and bound together in a 3D environment, through instrumental interaction. Hierarchy

is achieved by representing the loops in a tree structure.

KYMA [7] is a hybrid software and hardware environment for sound composition. It offers multiple pre-made facilities for sound creation such as multi-dimensional preset interpolation, sound composition by addition and mutation, or sequential and parallel sound composition on a time-line.

AUDIOMULCH [8] is an environment for live music performance, which also provides preset space exploration thanks to the Metasurface concept. Cantabile PERFORMER [1] is also an environment geared towards live performance, with the ability to trigger sounds, and a temporal ordering. It is closer to the cue metaphor than the sequencer metaphor.

Mobile and web applications are being increasingly used to create music, but their are often embedded in a bigger score or framework and act more as an instrument than in other systems. An interesting example of a web-based sequencer is JAMON [9] which allows multiple users to collaboratively and interactively author music by drawing within a web page interface. A deeper overview of the collaborative music authoring environments is given in [10].

Finally, modern video game music engines such as FMOD and AudioKinetic WWISE allow some level of interactivity, i.e when an event occurs in a video game, a sound will be played. Automation of parameters is possible, and these environments are geared towards three-dimensional positioning of sound and sound effects such as reverb, echo.

For low-level audio engines, one of the predominant methods is the audiograph. Prime examples are Jamoma AUDIOGRAPH [11] and INTEGRA FRAMEWORK [12]. Audio processing is thought of as a graph of audio nodes, where the output of a node can go to the input of one or multiple other nodes. Audio workstations such as Magix SAMPLITUDE (with the flexible plug-in routing) and Apple LOGIC PRO (with the Environment) provide access to the underlying audio graph.

3. CONTEXT

In this section, we will present the two tools that we used to achieve audio sequencing: I-SCORE and the LIBAUDIOSTREAM. I-SCORE is an interactive sequencer for parameters, which allows one to position events in time, and introduce interaction points and conditions throughout the score. The detailed execution semantics are given in [13].

The LIBAUDIOSTREAM [14] provides the functionality allowing the authoring of audio expressions through creation and combination of streams. The notion of symbolic date, introduced in an extension of the library, allows the user to start and stop the execution of streams at a time and date not known until the performance.

The goal of this work is to bind the audio capabilities of the LIBAUDIOSTREAM with the I-SCORE execution engine and graphical interface, in order to allow the creation of rich hierarchic and interactive musical pieces.

[1] https://www.cantabilesoftware.com/

3.1 Presentation of i-score

The original goal of I-SCORE is to communicate and coordinate other software in a synchronized manner, through the OSC protocol. The software can be used to send automations, cue-like 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 without a depth limit. Besides, there is no notion of "track" per se; rather, the composer works with temporal intervals which contain arbitrary data that can be provided by plug-ins.

Multiple methods of interactivity are available in I-SCORE: trigger points, conditions, mappings, speed control.

- Interactive triggers are used 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 enable the execution or disabling of part of the score according to a boolean condition. This makes if-then-else or switch-case programming constructs easy to implement in a temporal way.
- Mappings allow the user to map an input parameter to an output parameter, with a transfer function applied to the input.
- The execution speed of hierarchical elements can be controlled during the execution.

A span of time in I-SCORE might have a fixed or indefinite duration; we refer to this span as a Time Constraint (TC) since it imposes both a logical and temporal order to the elements before and after it.

A TC may contain data in the form of processes: automations, mappings, but also loops and scenarios; a scenario is the process that allows nesting. When the TC stops, all its processes are killed recursively.

TCs are linked together with Time Nodes, which allow for synchronization and branching of multiple streams of time.

An example of the temporal syntax of I-SCORE is presented in fig. 1. It is for instance used by Arias and Dubnov in [15] to construct a musical environment adapted to improvisation by segmenting pre-recorded audio phrases, to allow constrained improvisation according to high-level musical structures. The resulting generated structures bear similarity with the session concept in Ableton LIVE: one can loop dynamically over particular sections of a sound file.

3.2 Presentation of the LibAudioStream

The LIBAUDIOSTREAM [14], developed at GRAME, is a C++ library allowing the user to recreate the constructs commonly found in multi-track sequencers directly from code; it also handles communication with the sound card hardware via the common audio APIs found on desktop operating systems.

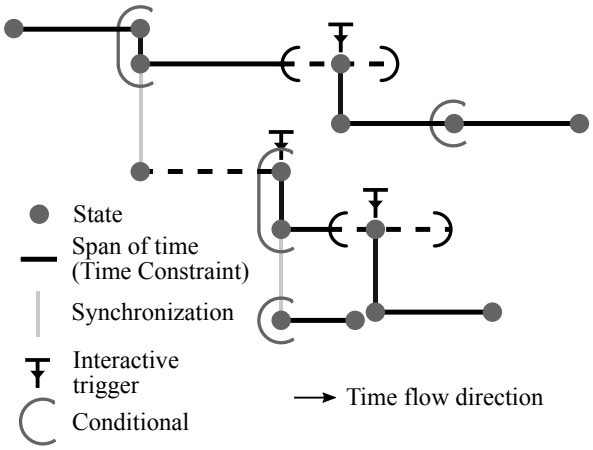

Figure 1. Part of an I-SCORE scenario, showcasing the temporal syntax used. A full horizontal line means that the time must not be interrupted, while a dashed horizontal line means that the time of this Constraint can be interrupted to continue to the next part of the score according to an external event

One of the advantages of using a library instead of a graphical interface is that it provides scripting capabilities to the composer and makes algorithmic music composition easier. It has been used with success in OPENMUSIC [16].

Audio sounds and transformations are modeled via streams. The following operations are applied to these streams: serial and parallel composition, mixing, and multi-channel operations. Streams are bound together in order to construct complex audio expressions. For instance, two sound files can be mixed together with a Mix stream expression:

```
auto sound = MakeMixSound(
   MakeReadSound("a.wav"),
   MakeReadSound("b.wav"),
   0.75);
```

A stream can then be played via an audio player, with audio sample accuracy:

```
StartSound(audioplayer, sound, date);
```

The play date must not necessarily be known in advance thanks to the notion of symbolic date [2] . Finally, FAUST [17] audio effects can be applied to the streams.

4. PROPOSED 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, in order to follow the principle of least astonishment [18]: in most audio software, the notion of grouping implies that the grouped sounds will be mixed together and routed to a single audio bus.

[2] Symbolic dates in the LibAudioStream are dates that can be set dynamically at run time.

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

Finally, we exhibit the translation of I-SCORE structures in LIBAUDIOSTREAM expressions, which required the creation of a dependency graph between audio nodes.

4.1 Group audio stream

In order to be able to apply hierarchical effects on the streams, and handle interactivity in recursive groups, we have to introduce a way to use sound hierarchy in the LIBAUDIOSTREAM.

Our method employs two elements:

- 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 sound card
- A way to reintroduce the player into the stream system, so that it is able to be pulled at regular intervals like it would be by a hardware sound card while being mixed or modified by subsequent stream operators.

We introduce 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 with the clock rate and sample rate of the sound card.
- A Group audiostream. This particular audiostream, of infinite length, allows the introduction of a Group player in a series of chained streams and takes care of having the Player process its buffers regularly.
- A finite loop audiostream. This stream loops over its content after a given duration.

The execution time of the nested objects will be relative to the start time of the Group audiostream.

4.2 Send and return audio streams

In order to be able to create temporal effect graphs, we introduced another couple of objects.

The Send audiostream by itself is a pass-through: it just pulls the stream it is applied to. It possesses the same definition: same length, same number of channels. The Return audiostream, constructed with a Send stream, makes a copy of the data in the Send stream and allows it to be used by the processing chain it is part of. For instance, this means that a single sound source can be sent to two effect chains in parallel.

The Return stream is unlimited 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.

The Send stream must itself be pulled regularly by being played as a sound, either directly or by a stream that would encapsulate it.

An example of such an audiostream graph is presented in fig. 2.

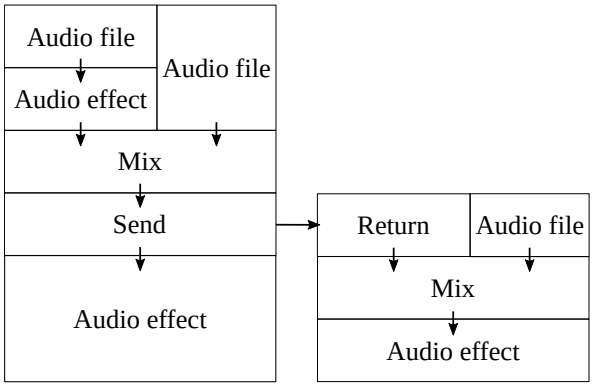

Figure 2. An example of audio stream composition with the Send and Return objects. An arrow from A to B means that B pulls the audio data from A.

4.3 Audio processes

We provide multiple audio processes in I-SCORE, that map to the existing LIBAUDIOSTREAM structures.

- Effect chain process: register multiple audio effects one after the other. For instance:
  Equalizer → *Distortion* → *Reverb*.
  Currently only FAUST effects or instruments are supported. Interfaces are provided to allow the extension to other audio plug-in formats.
- Input process: allows the introduction of the audio input of a sound card to the stream.
- Sound file: reads a sound file from the file system.
- Explicit send and return processes for manual routing.
- Mixing process: it exposes a matrix which allows to adjust the percentage of each sound-generating process going to each input process, send, and parent.

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 achieved by sending OSC messages to a remote software package. The OSC messages tree is modeled as an object tree. We present the loaded effect plug-ins to this object tree, so that automations and mappings can control audio effects and audio routing volume.

A screen capture of a TC with processes is given in fig. 3.

Figure 3. An example of a TC loaded with audio processes in I-SCORE. Selecting a particular process shows a complete widget for editing the relevant parameters. On a single TC, there can be only a single Mixing process (the table at the bottom), but there is no limit to the amount of other processes i.e there can be multiple sound files, etc.

4.4 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 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 associated with their output Send, for example e.g. an audio file reader, hierarchical elements, etc.
- Edges are the connections going from a send to a return, or from an element to the element it is mixed in.

The graph, implemented with the Boost Graph Library [19] can then be used to check for acyclicity. The user will be notified if that is not the case.

We provide here the method to build the graph.

Vertices are created recursively from the TCs in I-SCORE: an I-SCORE document is entirely contained in a top-level TC.

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 cases, we create a Group vertex to model the process. Edges are to be added from each stream in the hierarchical time-line, to the group stream.

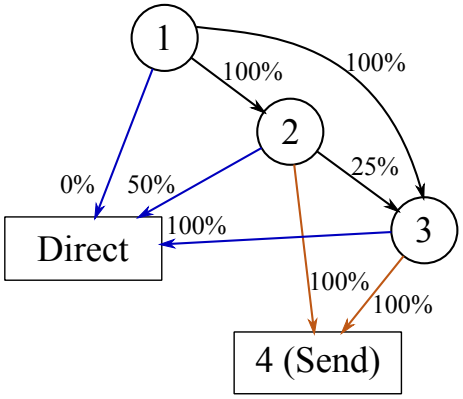

Figure 4. Translation of the TC of fig. 3 in a dependency graph. The edges in black represent the intra-Constraint connections. The edges in blue (resp. orange) represent a connection to a visible output of the Constraint. The percentages represent the level of mixing of the stream. *Direct* corresponds to the signal that will be sent at the upper level of hierarchy.

If the process is a send or a return, we create a corresponding vertex. Then, we create inner Sends for all the streams and a vertex for the Constraint itself.

Once all the vertices are created, the edges are added as mentioned before.

As mentioned before, there is an ordering between nodes of the graph: the parent-most vertex has to be pulled before the others to reflect the causality.

Inside a TC, causality also has to be enforced. Since a mixing matrix is provided, we have to ensure that an effect bus cannot be routed back into itself creating a loop. To prevent this at the user interface level, we disable by default the mixing of audio effect chains into each other. In fig. 4, we show the resulting graph for a TC.

When the graph is verified acyclic, we perform the stream creation by iterating over the list of topologically sorted vertices.

4.5 Stream creation

In this section we will detail the stream creation for particular elements.

4.5.1 Scenario

An I-SCORE scenario is an arrangement of temporal structures, as shown in fig. 1; it is an independent time-line. Since the execution and duration of these structures can change at run-time due to interactivity and hierarchy, it is not meaningful to directly use the tools provided by the LIBAUDIOSTREAM: sequence stream, parallel stream, mix stream. We instead use the Group player to organize our elements in time.

The creation of the Scenario stream is done as follows:

1. A Group player is created.
2. For each Time Node in the scenario, a symbolic date is generated.
3. For each TC 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. In order to enforce sample-accuracy whenever possible, if the i-score structures have a fixed date, we preset this date to its computed value. If there is no interactivity involved, a sound directly following on from another will begin to play from the audio sample past the end of the first one. As soon as a sound's execution time is fixed, an algorithm checks for all the following sounds whose date could also be fixed.

4.5.2 Loop

Due to their interactive nature, loops in I-SCORE can be entirely different from one iteration to another. They are more similar to imperative programming `do-while` constructs, than audio sequencer loops. This prevents us from directly using the LIBAUDIOSTREAM's loop stream, since it expects a looping sound of finite duration. Instead, if the loop is interactive, we wrap the loop pattern TC's audiostream in a Group player, reset the stream and start it again upon looping. If the loop is not interactive, we can reset it at a fixed interval of time with the fixed loop stream introduced earlier. This allows for sample accurate hierarchical looping with I-SCORE's process semantics.

4.5.3 Time Constraint

As explained earlier, a TC is a process container. Such processes can be the sound processes presented in section 4.3, and the control processes such as automation, etc.

The creation of the Constraint audio stream is done as follows:

1. For each sound-generating process, a stream and a send are created.
2. For each effect chain, the effects are instantiated and an effect stream is created with a mix of the returns of the elements to which this effect applies. A send is also created.
3. The mixing matrix is used to create mix audio streams from the sends and returns, which are routed either in the user-created sends, or in the stream corresponding to the TC. A time-stretching audio stream is inserted before the send: it is linked to the execution speed of the TC in I-SCORE which can vary interactively.

4.5.4 A note on real-time performance

Since a real-time audio input is provided, we ought to be able to use the system as a multi-effect, so as to introduce the lowest possible latency. The time-stretching effect itself may impose a latency high enough to make playback through the system impossible. Similarly, Sends and Returns must operate at the same playback speed; else, the Return will either fetch not enough data, or skip data from the Send it is listening to.

To solve this, when creating the graph, the parents of each Input, Send, and Return nodes are recursively marked with a flag to indicate real-time processing. The TCs with this flag will not be able to be time-stretched, and will only be affected by the latency due to the effects manually introduced by the composer.

5. EXAMPLES

We present in this part three examples of usage of the presented system.

5.1 Recreation of a multi-track sequencer

The first example, in fig.5, is a recreation of the multi-track audio sequencer metaphor, with the primitives presented in this paper.

This score has three tracks, **Guitar**, **Bass**, and **Drums**, which are implemented with three TCs. Each TC has a Sound process and an Effect process; the Mixing process is hidden for clarity. The bass track is a looping one-note sound. Automations are applied either at the "track" level, as for the drums, or at the "clip" level, as for the guitar **outro** clip. However, in the model there is no actual difference between track and clip, it is solely a particular organization of the score.

5.2 Interactive scenario

The second example, in fig.6, gives an overview of the interactive possibilities when creating a song.

The score behaves as follows: for a few seconds, **intro** will play. Then, if an external event happens, like a foot switch being pressed, multiple things may happen:

- In all cases, the **eqcontrol** part will play, and automate a value of a global effect.
- If a first condition is true (**case1**), then **case1.B** will start playing immediately, and **case1.A** will start playing after a slight delay. If another external event happens, **case1.A** will stop playing immediately.
- If a second condition is true, at the same time, **case2** will start playing.
- After **eqcontrol** finishes, a hierarchical scenario **outro** is played, which contains two sounds and a parameter mapping.

If no external event happens, after some time, when reaching the end of the triggerable zone delimited by the dashed line, the triggering occurs regardless of user input.

5.3 Temporal effect graph

This last example, in fig. 7 shows how to arrange not sound, but sound processing temporally. In this case, we have a sound playing, which is routed in the send process. Then, a hierarchical scenario with multiple TCs is used. Two TCs have return processes connected to the previously created send. Automations are applied to parameters of these effects.

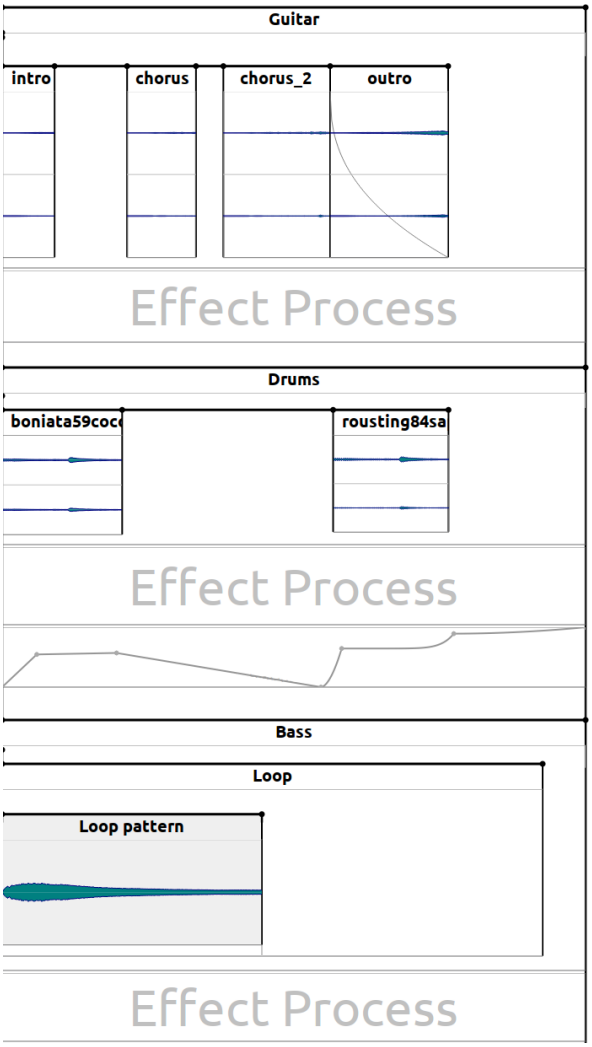

Figure 5. Multi-track sequencing.

Second effect will be triggered after an external event happens. By using loops, effects, and TCs with infinite durations, this same mechanism would allow one to simulate a guitar pedal board with switchable effects, and to create temporal transitions between the various sounds.

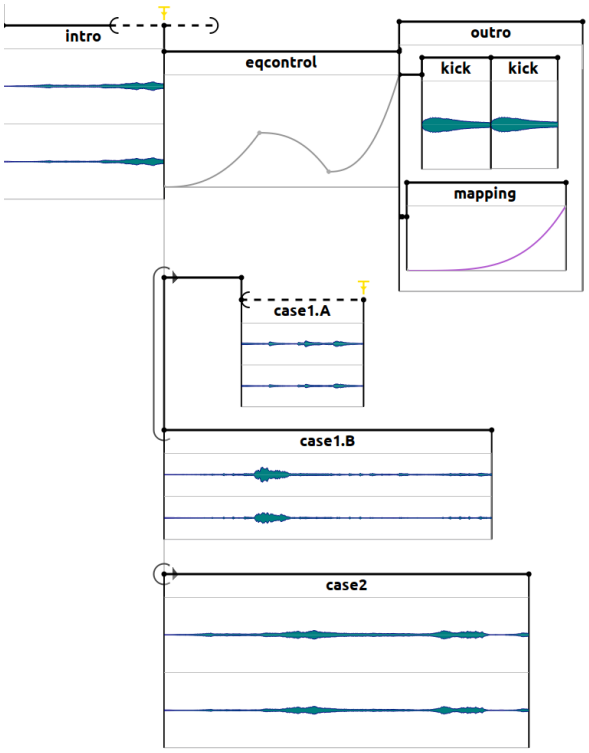

Figure 6. An interactive musical score.

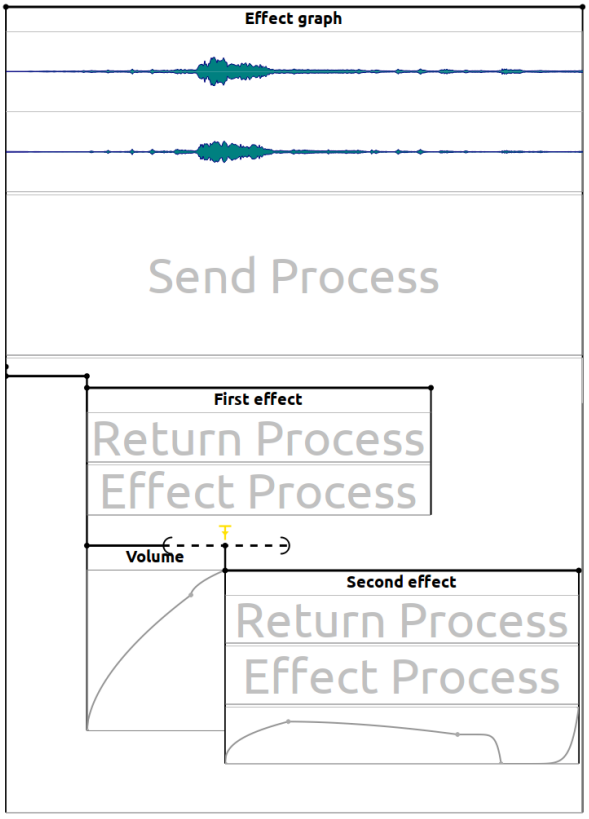

Figure 7. Temporal effect graph applied to a sound.

6. CONCLUSION

We presented a computer system for creating interactive music, which extends the audio sequencer metaphor. New kind of streams enabling hierarchy and audiograph-like behavior are introduced to the LIBAUDIOSTREAM, which is then binded to the I-SCORE primitives for specifying and scoring time and interaction. Three examples present the various musical possibilities that are offered through this system.

However, there are currently some key differences to more traditional musical environments: for one, musical notation and concepts are absent from the system. All durations are expressed in seconds or milliseconds, instead of beats or any subdivision as they would in other environments. A possible extension to the I-SCORE execution engine would be to take into account beats for triggering, which would allow the user to synchronize multiple hierarchical loops to a beat and may be useful for some genres of music, such as electronica or rock.

Likewise, the system mostly handles audio and OSC data; MIDI is implemented at a primitive level. Another unhandled question is effect delay compensation: sometimes, audio algorithms must introduce multiple frames of latency in their processing chain, for instance because they have to accumulate a certain amount of data. This is not taken into account here, hence seemingly synchronized sounds may desynchronize themselves if this latency is not accounted for.

Finally, in many cases optimizations could be achieved to reduce the amount of data structures being created. For instance, when a single sound file is in a TC, a simpler stream expression could be created.

The next steps for this research includes these points, work on sound spatialization, and interactive edition: modifying the score while it is already playing.

Acknowledgments

This research was supported by the SCRIME (Studio de Création et de Recherche en Informatique et Musiques Expérimentales, `scrime.labri.fr`) which is funded by the French Culture Ministry. SCRIME is a GIS (Group of Interest in Science and Art) with University of Bordeaux, Bordeaux INP, Bordeaux City, CNRS (National Scientific Research Center), Région Nouvelle Aquitaine (Aquitaine Regional Council) and the DRAC (Regional Direction of Culture). This work was also supported by an ANRT CIFRE convention with the company Blue Yeti under funding 1181-2014. The authors wishes to thanks Stéphane Letz for his help with the LibAudioStream.

7. REFERENCES

[1] A. Bell, E. Hein, and J. Ratcliffe, "Beyond Skeuomorphism: The Evolution of Music Production Software User Interface Metaphors," *Journal on the Art of Record Production*, 2015.

[2] A. Möllenkamp, "Paradigms of Music Software Development," in *Proceedings of the 9th Conference on Interdisciplinary Musicology*, 2014.

[3] B. Vercoe and D. Ellis, "Real-time csound: Software synthesis with sensing and control," in *Proceedings of the International Computer Music Conference*, 1990, pp. 209–211.

[4] M. Puckette *et al.*, "Pure data: another integrated computer music environment," *Proceedings of the second intercollege computer music concerts*, pp. 37–41, 1996.

[5] J. McCartney, "Rethinking the computer music language: Supercollider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, 2002.

[6] F. Berthaut, M. Desainte-Catherine, and M. Hachet, "Drile: an immersive environment for hierarchical live-looping," in *New Interfaces for Musical Expression*, 2010, p. 192.

[7] C. Scaletti, "The kyma/platypus computer music workstation," *Computer Music Journal*, vol. 13, no. 2, pp. 23–38, 1989.

[8] R. Bencina, "The metasurface: applying natural neighbour interpolation to two-to-many mapping," in *New Interfaces for Musical Expression*, 2005, pp. 101–104.

[9] U. Rosselet and A. Renaud, "Jam On: a new interface for web-based collective music performance," in *New Interfaces for Musical Expression*, 2013, pp. 394–399.

[10] R. Fencott and N. Bryan-Kinns, "Computer musicking: HCI, CSCW and collaborative digital musical interaction," in *Music and Human-Computer Interaction*. Springer, 2013, pp. 189–205.

[11] T. Place, T. Lossius, and N. Peters, "The jamoma audio graph layer," in *Proceedings of the 13th International Conference on Digital Audio Effects*, 2010, pp. 69–76.

[12] J. Bullock and H. Frisk, "The integra framework for rapid modular audio application development," in *Proceedings of the International Computer Music Conference*, 2011.

[13] J.-M. Celerier, P. Baltazar, C. Bossut, N. Vuaille, J.-M. Couturier, and M. Desainte-Catherine, "OSSIA : Towards a Unified Interface for Scoring time and Interaction," in *Proceedings of the TENOR 2015 Conference*.

[14] S. Letz, "Spécification de l'extension LibAudioStream," Tech. Rep., Mar. 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00965269

[15] J. Arias, M. Desainte-Catherine, and S. Dubnov, "Automatic Construction of Interactive Machine Improvisation Scenarios from Audio Recordings," in *4th International Workshop on Musical Metacreation (MUME 2016)*, Paris, France, 2016.

[16] D. Bouche, J. Bresson, and S. Letz, "Programmation and Control of Faust Sound Processing in OpenMusic," in *Joint International Computer Music/Sound and Music Computing Conferences*, 2014.

[17] Y. Orlarey, D. Fober, and S. Letz, "Faust: an efficient functional approach to DSP programming," *New Computational Paradigms for Computer Music*, vol. 290, 2009.

[18] P. Seebach, "The cranky user: The Principle of Least Astonishment," in *IBM DeveloperWorks*, 2001.

[19] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual, Portable Documents*. Pearson Education, 2001.