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

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. . .

Most of the existing software 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 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. 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 a 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 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

Outside of the traditional audio sequencer realm, we can find multiple occurences of graphical environments aiming to provide some level of interactivity.

Möllenkamp presents in [2] the commons paradigms for creating music on a computer : Score-based with MUSIC and Csound, patch-based with Max/MSP or PureData, programming-based with SuperCollider and many of the other music creation languages, music trackers such as FastTracker which were used to make the music in old video game consoles, and multitrack-like such as Cubase, Pro Tools. Ableton Live and Bitwig Studio are given their own category thanks to the ability to compose clips of sound interactively.

Drile [3] is a virtual reality music software : loops are manipulated and bound together in a 3D environment. Hierarchy is achieved by representing the loops in a tree structure.

Mobile and web applications are also a way that is more and more used to create music, but their are often embedded in a bigger score or framework and act more as an instrument than other systems. An interesting example of web-based sequencer is JamOn [4] which allows multiple persons to author music interactively in collaboration by drawing in a web page.

Finally, modern video game music engines such as FMOD and Audiokinetic wWise allow some level of interactivity : when some event occurs in a video game, then 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 audio engines, one of the predominant metaphors is the audiograph. Prime examples are Jamoma AudioGraph [5] and Integra Framework [6]. Audio processing is thought of as a graph of audio nodes, where the output of a node can go to the input of another node.

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 for parameters, 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 [7].

The LibAudioStream [8] provides the ability to author audio expressions 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.

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.

3.1 Presentation of i-score

The main use of i-score is to communicate and orchestrate other software in a timely manner, through the OSC protocol. The software can 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 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.

Multiple possibilities of interactivity are provided in i-score : trigger points, conditions, mappings, speed control.

- 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.
- 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 since it imposes both a logical and temporal order to the elements before and after it.

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.

Time Constraints are linked together with Time Nodes, which allows for synchronization and branching of multiple streams of time. An example of the temporal syntax of i-score is presented in fig. 1.

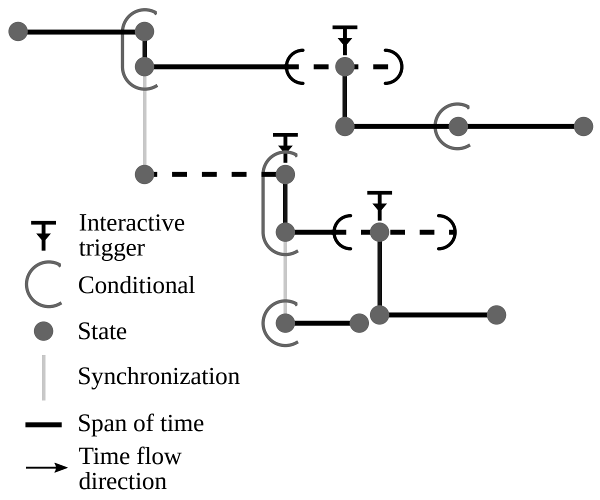

Figure 1. Part of an i-score scenario, showcasing the temporal syntax used. A full horizontal line means that the time must not be interrupted, while a dashed horizontal line means that the time of this Constraint can be interrupted to proceed to the following parts of the score according to an external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

3.2 Presentation of the LibAudioStream

The LibAudioStream [8], developed at GRAME, is a C++ library allowing to recreate the constructs commonly found in multi-track sequencers directly from code; it also handles communication with the soundcard hardware via the common audio APIs found on desktop operating systems.

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 [9].

Audio sounds and transformations are modeled by streams; an algebra with the expected operations is applied to these streams :serial and parallel composition, mixing, and multichannel 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 = MakeMixStream(
    MakeAudioStream("a.wav"),
    MakeAudioStream("b.wav"),
    1.);
```

A stream can then be played through 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. Finally, Faust [10] 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. This is done in accordance with the principle of least astonishment [11] 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.

Finally, we exhibit the translation of i-score structures in LibAudioStream expressions, which requires 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, we have to introduce a way to use 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 soundcard.
- 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 real soundcard while being mixed or modified by subsequent stream operators.

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 and takes care of having the Player process its buffers regularly.

The execution time of the hierarchized 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 introduce another couple of objects.

The Send audiostream by itself is a passthrough : it just pulls the stream it is applied to. It posseses 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 single sound source can be sent to two effect chains 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.

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.

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

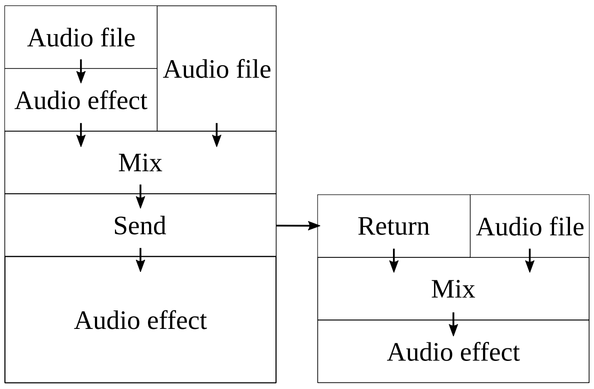

Figure 2. An example of audio stream composition with the Send and Return objects. An arrow from A to B means that B pulls the audio data from A.

4.3 Audio processes

We provide multiple audio processes in i-score, that map to the LibAudioStream structures.

- Effect chain process : register multiple audio effects one after the other. For instance :
  Equalizer → *Distortion* → *Reverb*.
  Currently only Faust effects are supported.
- Generator process : a sound generator, like a synthesizer. Currently only Faust instruments are supported.
- Sound file : reads a sound file from the filesystem.
- 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. Hence, the OSC message tree is modeled as an object tree. We present the loaded effect plug-ins to this object tree, so that automations and mappings are able to control audio effects and audio routing volume.

An example is given in fig. 3.

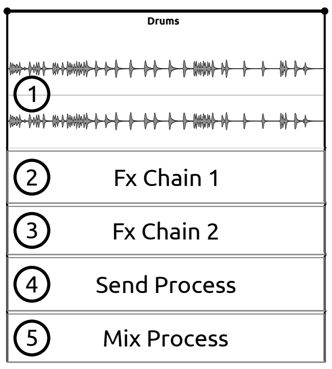

Figure 3. An example of a Time Constraint loaded with audio processes in i-score. Selecting a particular process shows a complete widget for editing the relevant parameters. On a single Time Constraint, there can be only a single Mixing process, but there is no limit to the amount of other processes : there can be multiple sound files, etc.

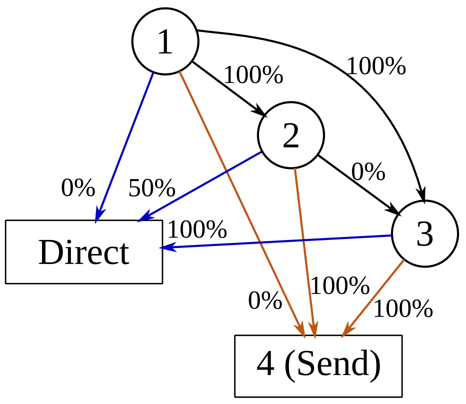

Figure 4. Translation of the Time Constraint of fig. 3 in a dependency graph. The edges in black represent the intra-Constraint connections. The edges in blue (resp. orange) represent a connection to a visible output of the Constraint. The percentages represent the level of mixing of the stream. *Direct*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

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 to their output send : 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 [12] can then be used to check for acyclicity, and return an user error if that is not the case. As 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.

We provide here the method to build the graph :

Vertices are created recursively from the Time Constraints in i-score : an i-score document is entirely contained in a top-level Time Constraint.

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. Edges are to be added from each stream in the hierarchical timeline, to the group stream.

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

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

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

Inside a Time Constraint, causality also has to be enforced. Since a mixing matrix 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; this is also visible on the fig. 4.

Hence, the graph is topologically sorted, which allows for the creation of the streams afterwards.

4.5 Stream creation

We detail in this section 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's a timeline of its own. Since the duration of these structures can change at run-time due to interactivity, it is not meaningful to use the tools provided by the LibAudioStream : sequence stream, parallel stream, mix stream. We instead use the audio player concept 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 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.

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 traditional programming language `do-while` constructs, than audio sequencer loops. This prevents us from using the LibAudioStream's loop stream, since it expects a looping sound whose duration will not change from an iteration to another. Instead, we wrap the loop pattern Time Constraint's audiostream in a Group player, reset the stream and start it again upon looping.

4.5.3 Time Constraint

As explained earlier, a Time Constraint 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 Time Constraint. A time-stretching audio stream is inserted before the send : it is linked to the execution speed of the time constraint in i-score which can vary interactively.

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 multitrack audio sequencer metaphor, with the primitives presented in this paper.

This score has three tracks, **Guitar**, **Bass**, and **Drums**, which are implemented with three Time Constraints. Each Time Constraint 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.

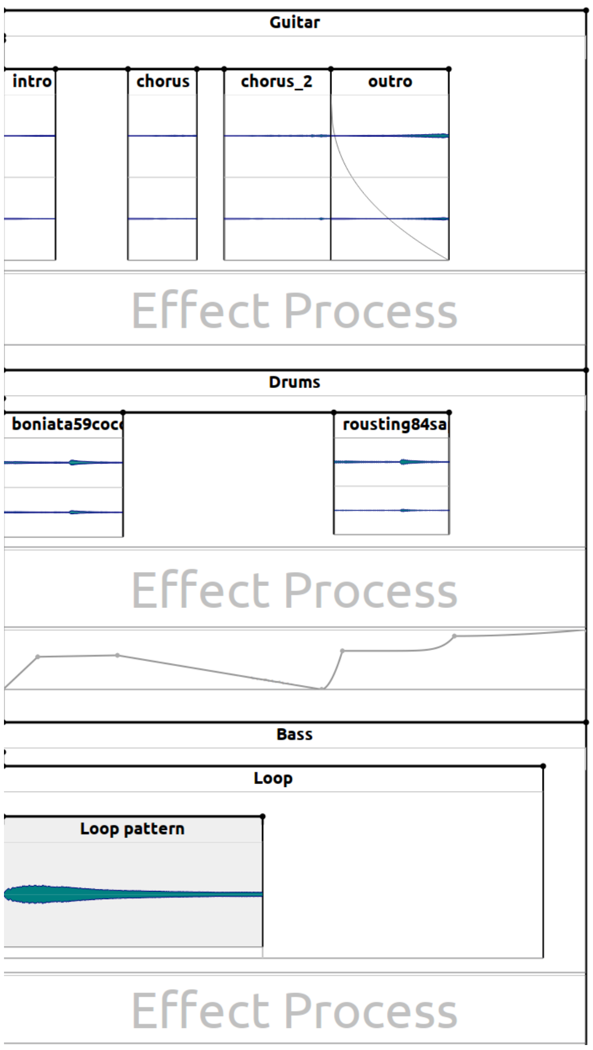

Figure 5. Multi-track sequencing.

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 footswitch 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 anyways.

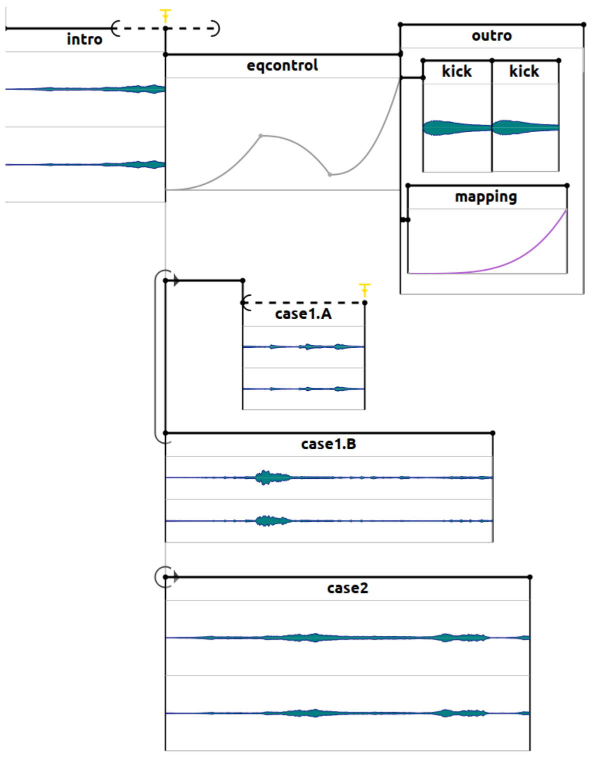

Figure 6. An interactive musical score.

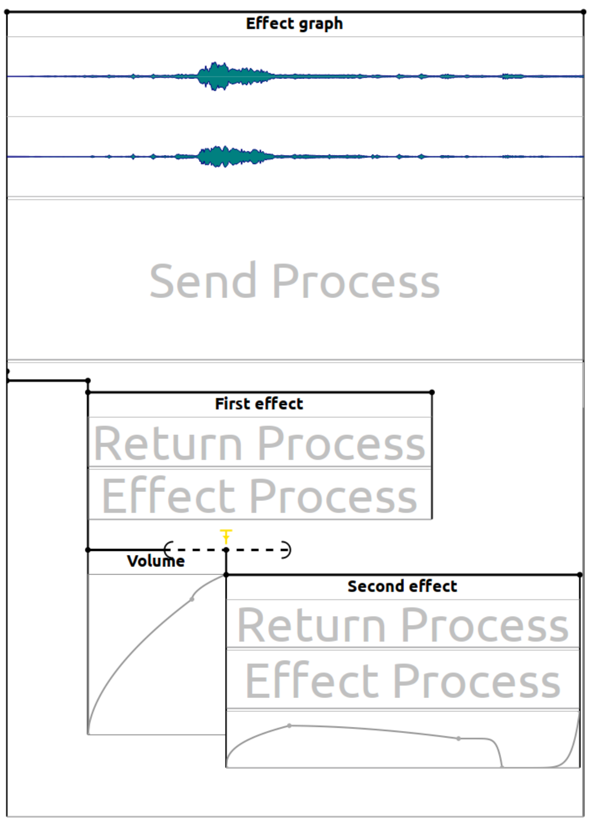

Figure 7. Temporal effect graph applied to a sound.

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 is used, and contains multiple Time Constraints, two of which have return processes connecte to the previously created send. Automations are applied to parameters of these effects.

Second effect will be triggered after an external event happens. By using loops, effects, and time constraints with infinite durations, this same mechanism would allow to simulate a guitar pedalboard with switchable effects, and the added possibility to create transitions between effects.

6. CONCLUSION

We presented a computer system for creating interactive music, inspired by but extending the audio sequencer metaphor. New kind of streams enabling hierarchy and audiograph-like behaviour 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 differences with more traditional musical environments : for one, musical notation and concepts are absent from the system. All the 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 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 only handles audio data, and not MIDI. 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 optimisations could be achieved to reduce the amount of data structures being created. For instance, when a single sound file is in a Time Constraint, a simpler stream expression could be created.

The next steps for this research includes these points, the handling of real-time audio input, and interactive edition : modifying the score while it is already playing.

Acknowledgments

Redacted.

7. REFERENCES

[1] A. Bell, E. Hein, and J. Ratcliffe, “Beyond skeuomorphism: The evolution of music production software user interface metaphors,” 2015.

[2] A. Möllenkamp, “Paradigms of music software development,” in *Proceedings of the 9th Conference on Interdisciplinary Musicology*, 2014.

[3] F. Berthaut, M. Desainte-Catherine, and M. Hachet, "Drile: an immersive environment for hierarchical live-looping," in *New Interface for Musical Expression*, 2010, pp. page–192.

[4] U. Rosselet and A. Renaud, "Jam on: a new interface for web-based collective music performance." in *NIME*, 2013, pp. 394–399.

[5] 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.

[6] J. Bullock and H. Frisk, *The integra framework for rapid modular audio application development*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2011.

[7] 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*.

[8] S. Letz *et al.*, "The libaudiostream library," 2012.

[9] 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 (ICMC-SMC)*, 2014.

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

[11] P. Seebach, "The cranky user: The principle of least astonishment," in *IBM DeveloperWorks*, 2001.

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