A NOTATION SYSTEM FOR DISTRIBUTED MEDIA ART

Jean-Michaël Celerier
Concordia University / ossia.io
Montréal, Canada
jeanmichael.celerier@gmail.com

Akané Levy
ossia.io
Talence, France
levy.akane@gmail.com

ABSTRACT

Interactive, intermedia scores are scoring systems which enable composers to specify temporal evolutions and variations of such multimedia systems. We introduce a visual notation for distribution of the interpretation and execution of such scores over a computer network: both the temporal organization of the score and the multimedia data can be distributed through a set of scoring primitives.

For instance, we will show how one can write and execute a score which specifies: part A plays on a first group of computers, followed by part B on a second group of computers, while a parallel part C containing a synchronized video effect is being played on a separate machine with dedicated video hardware. Based on the pre-existing scoring environment *ossia score*, we cover how the extensions interact with the existing score execution algorithm, and present a set of small distributed scores which enables varied behaviours to be defined by the composer from our proposed set of nine temporal and three dataflow primitives for score distribution.

1. INTRODUCTION

Interactive scores are scoring systems which encode interactivity and the set of variations that can take place during the performance of said scores. Intermedia scores are scoring systems which encode not only the actions of musical instruments, but also of video systems, OSC controls, etc. *ossia score* [3, 4] is both an interactive, intermedia scoring language, and the software implementation of said language as an open-source score editor and interpreter.

The present research covers multiple ongoing parallel tracks for the distribution of interactive, intermedia scores, which converged in an implementation in *ossia score*. Simply put, we are interested in all the possible ways in which one can "distribute" an interactive, intermedia score over a computer network, and in how visual scoring languages can be defined to enable composers to write such a score, with a unique visual representation conveying the distribution semantics desired by the composer.

A fundamental example of distribution is the classical orchestra: orchestral scores can specify staves for violins, flutes…which are then distributed to an arbitrary number of performers which will all have their own personal interpretation of the score.

In the case of intermedia scores, it is common for computers and their strict internal clocks to be at the core of the performance: interactive, intermedia artworks can range from pieces that do not incorporate any human element during the performance to intricate collaborations between humans and machines. The goal of this research is to devise a notation to encode distributed semantics inside *ossia score*, which will allow performance to occur from a single document shared over the network, and provide an implementation of the distributed performance of said notation.

Distributed works in media art have existed for a long time: laptop orchestras [14] are a famous instance of musical ensembles revolving around a group of human performers being conducted and themselves using their laptops as musical instruments. Said laptops can optionally be synchronized, for instance to keep a shared musical beat across the piece. The question of conducting in the face of computer networks is discussed by Smallwood in [13]:

> Thus, possibilities for coordination, message-passing, group control, quantization, tempo, dynamics and so on are on the table for all composers working with PLOrk. Should these tasks be given to a conductor? Should the conductor be human, or should it be a program operating over the network? Or should there be both kinds of conductor?

Scoring systems for laptop orchestra pieces are as far as the author could find, bespoke and generally customized for each individual piece. Other systems that embody distributed composition and performance often focus on distributing scores that are closer to traditional western sheet music, such as *Quintet.Net* introduced by Hajdu in [8] and based on Max/MSP. Likewise, *Indra* [1] is another Max-based system for networked live performance with real-time compositional aspects. Other systems focus more on graphical or alternative notations: *Decibel ScorePlayer* [9] "allows for network-synchronised scrolling of proportional colour music scores on multiple iPads". More recently, Drawsocket [7] by Gottfried and Hajdu focused on live generation of graphical scores in the web browser from Max/MSP patches and a NodeJS communication layer, with the main aim being conduction of human performers through the distribution of the visual notation over multiple networked displays.

In the present paper, we will argue for a basic set of visual primitives for writing such scores in the hope of foster-

ing collaboration and a common understanding of the ways performers, be they human or machine, can be synchronized (or not) in such artworks. First, semantics for distribution of both temporal interactions and specifications will be presented, then we will introduce a core primitive for symbolizing the various ways exchange and sharing of multimedia data can be done in a distributed score. Extensions to *ossia score*'s visual language to encode these distribution semantics into the scoring language will be presented, and various examples of usage will be discussed. The implementation is entirely self-contained in *ossia score*. Due to its current use of WebSockets for communication, it works transparently across both desktop apps and the beta version of the WebAssembly port of the software.

2. AN OSSIA SCORE PRIMER

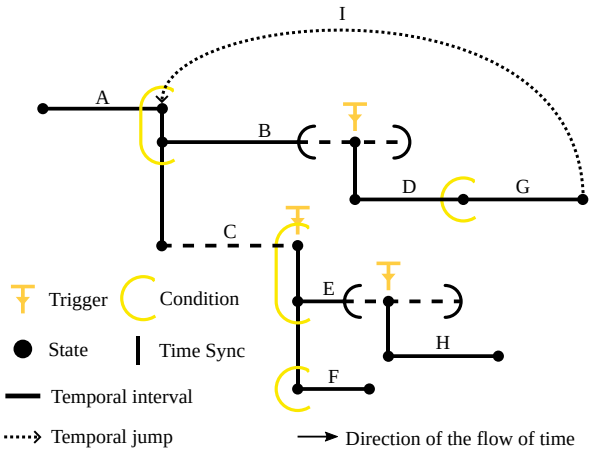

Figure 1: Syntactic elements of *ossia score*'s visual language.

Dubbed "interactive sequencer for the intermedia arts", *ossia score* is a system which combines both non-linear time-lines and the data-flow paradigm to allow artists to create interactive multimedia artworks, musical pieces, museum installations, etc. It supports audio and video playback and effects, as well as communication and control over protocols such as OSC, OSCQuery, WebSockets, Art-Net or even raw serial port communication.

The visual language is showcased in fig. 1. Here is a short description of the syntax elements and their semantics: *time intervals* are represented with horizontal lines proportional to their durations. A full horizontal line means that the time must not be interrupted, while a dashed horizontal line means that the time of the interval can be interrupted to continue to the next part of the score according to an external event. The time during which such interruption can occur can be bounded by the user: these bounds are visible as the small black parentheses around the dashes. Start and end points of a group of intervals can be synchronized. On these synchronizations points, an optional trigger can be used to specify that the synchronization is dependent on an external event, such as an OSC message. Conditions allow to selectively disable parts of the score depending on the value of an external control at the moment execution reaches that point: an OSC parameter, a MIDI CC…

In fig. 1, execution occurs as follows: the interval A runs for a fixed duration. When it ends, a condition is evaluated: if it is false, the branch which contains B will not run. Else, after some time, the flow of time in B reaches a flexible area centred on an interaction point, also called a trigger. If an interaction happens, B stops and D starts. If there is none, D starts when the max bound of B is reached by the flow of time in B. Just like after A, an instantaneous condition will make G execute or not execute. In all cases, C started executing after A. C expects an interaction, without time-outs. If the interaction happens, the two instantaneous conditions which follow C are evaluated: the truth value of each will decide of the execution of E and F. Finally, after G, execution circles back to the end of A thanks to the temporal jump I; the condition will be reevaluated, at least C will start again, and B depending on the value of the condition. Besides this, the system allows varying each interval's execution speed independently during the performance: even if all the external conditions evaluate to the same value at the same time, it is possible to have two very different executions for the same score.

In itself, this only represents an abstract "flow of time": the score language needs to have a way to specify concrete behaviours, such as sounds or videos being played, OSC messages being sent, etc. This is done at two levels: *states* can contain a bundle of messages that are sent at a specific point in time, and *intervals* contain *processes* that generate, filter and output data: automations, MIDI, sound or video file players, VST effects and instruments, PureData patches, JavaScript scripts, video effect processors, etc. Processes can be linked together in a dataflow graph akin to the usual patching or coding environments: PureData [12]…, or communicate directly with the external environment: OSC-like addresses can be specified for their inputs and outputs ports to address the external environment of the software, visualized in 2.

These processes are rarely represented on paper notation. The software implements this visual language along with many such processes; the temporal scoring language is itself defined as such a process, called *Scenario*, and as such can be nested recursively to implement grouping and hierarchy: scenarios describe the organization of intervals, which themselves contain processes.

3. DISTRIBUTING SCORES

A first prototype for distribution in *ossia score* had been introduced in [2]. We recapitulate here the core ideas: the distribution of interactive scores is a semantic that allows defining on which hardware specific parts of scores are going to run. For instance: a computer handles the audio processing of a performance, while another processes video feeds. Through simple modifications to the software model of *ossia score*, this enables varied networked behaviours to occur. Our implementation uses a client-server architecture: one of the machines acts as a server and ensures the coherency of the network session. More interesting peer-to-peer implementations would be possible through distributed consensus algorithms such as Paxos[10], but in

Figure 2: A screenshot of *ossia score*, showing the main system for authoring scores with the visual language of fig. 1. On the left, a tree view shows the OSC addresses to which the score can speak. On the right, an inspector allows to edit the properties of selected elements. The score is in the center.

practice the current implementation has performed sufficiently well for the needs of the scores written with it so far. The success of the client-server architecture in *Quintet.Net* shows that this is a viable long-term approach.

3.1 Abstracting over hardware with groups

The core idea is the addition of a notion of groups to the software model. Groups are the interface and abstraction between physical hardware and the semantic level of the score: we do not want to annotate IP addresses representing a specific computer in scores for obvious maintainability and readability reasons. The way groups are used by the composer is simply by assigning objects of the score to a group defined as part of the score's metadata: a part of the score can be assigned to a group named `audio`, another part to a group named `osc_controls` for instance. These parts can play in parallel, one after each other, or more generally in any temporal arrangement made possible by the scoring semantics given in Section 2. Then, when the performance happens, all the computers supposed to partake in the execution of the score connect to the session, and choose which groups they want to join. During playback, a given client computer will execute only the elements of the score which are assigned to any of its groups. This means that for prototyping, it is trivial to run the entire score on a single computer, simply by assigning the unique score instance to every group defined in the score.

Due to the hierarchic model of *ossia score* processes, where a scenario is itself a process, entire parts of the score can easily be distributed to various groups, simply by changing a property on a parent node of this hierarchical process tree. The group property is inherited across the child objects: if an interval is assigned to a group `A`, all its children processes are recursively assigned to that group too unless one explicitly selects another group. Likewise, all the other properties mentioned in the sections below propagate hierarchically until an element of the score with properties explicitly set by the composer is encountered.

3.2 Distribution of interaction

Remember that *ossia score* allows defining interaction points in the score: in the non-distributed case, this means for instance that someone can write a score where a sound plays until a physical sensor is activated, then a light flash occurs. The interaction point will contain an expression such as: `device:/sensor/1 > 100`. The question we ask is: what are the possible semantics for the distribution of the evaluation of such an expression over the network – and most importantly, what does it enable intermedia composers to do. The main idea is that we can leverage the multiplicity of computer clients to represent consensus in the score: for instance, by assigning the interaction point to a group, we require that all the participants to the group validate the expression for enacting forward progress past the interaction point in the score; client machines outside of the group simply don't participate in the expression evaluation and just follow the result defined by the group evaluating the expression.

We consider two axes for defining the temporal relationships between how the synchronization of expression resolution is achieved across a group of clients executing a given scenario. The temporal properties to balance are:

- Latency: how fast an individual machine reacts to the resolution of an interactive expression on the network.
- Respect of the temporal order: how precisely the overall execution of the score on the network matches the specification given by the composer.

These two properties are at odds: for instance, if in the case given above, different groups and hardware execute sound playback and light control, is it tolerable for the aesthetics desired by the composer that the computers of the first group still play back sound for a few milliseconds if that means that the light flash starts closer to the sensor parameter change? Only the composer can answer this question.

The proposed distribution system enables fine-tuning of the synchronization mechanism by leveraging the pre-existing asynchronous infrastructure in *ossia score*: interactive trigger points. Two cases are possible for interactive trigger points assigned to a group: they can be time-compensated – that is, when all the computers validate the trigger, the network engine will try to define a date in the future at which every computer must actually execute the trigger point, so that this happens simultaneously from the point of view of an external observer looking at all the computers. Or, execution can also happen on an as-fast-as-possible basis, which can be helpful when the artistic trade-off between latency and synchronization tips in favour of the shortest reaction times.

- Asynchronous versus synchronous: in the asynchronous case, the synchronization semantics of trigger points are not respected across the network. The triggering algorithm is:

1. Obtention of a consensus on the value of the expression across the network on the server machine.
2. The server notifies all the clients, which react as soon as they get a message.

This means that if the latency between a client executing a process that precedes the trigger point and the server is greater than the latency between the server and a client that executes a process following the trigger point, to an external observer, an external observer will see both timelines overlap for a short duration, which would be impossible in a non-distributed execution of the score.

In the synchronous case, the semantics of trigger points are respected across the network: the following intervals cannot start before the previous intervals have ended on all clients. The triggering algorithm is:

1. Obtention of a consensus on the value of the expression across the network on the server machine.
2. The server notifies all the clients that the trigger has started executing.
3. When the clients finish executing the intervals that precede the trigger point, they notify the server.
4. The server notifies all the clients that the trigger has finished executing.
5. The clients start executing the following intervals.

This of course increases latency, but ensures that the temporal semantics of the score are respected globally: the score executes on the network in the same order as it would on a single machine, except with greater delays.

- Uncompensated versus compensated: in the uncompensated case, messages are processed as soon as they arrive. In the compensated case, the server tries to derive a timestamp in the future where all clients are supposed to have received a message according to their respective latency with regards to the server, in order to make sure that from an external observer point-of-view, everything happens simultaneously.

  For instance, consider a situation with three machines: one server at time t_0 with latency 0 with regards to itself, one with a latency of 50 milliseconds and one with a latency of 100 milliseconds, the server will send to the entire network session messages that request execution at a minimum of $t_0 + 100$ms. Note that we implicitly assume that the machine's wall clocks are synchronized through an external mechanism such as NTP or PTP.

3.3 Polyphony

Processes of the score can operate either in `free` or `shared` mode. Free means that the properties and execution of a given process is independent across machines; shared means that the process's state will be synchronized across all the machines that execute it. For instance: a scenario playing a sound could play it at two different speeds and have entirely different conditions resolutions on multiple clients in the free case, while in the shared case, all the clients executing it will be kept in sync for this specific scenario's execution: condition resolution and interval speed adjustments will be synchronized across the network. Likewise, for processes with controls, this means that a change of control during execution will not be synchronized across machines in the `free` mode, to enable for instance multiple performers to interact with a virtual musical instrument each in a different way on their respective computer. In contrast, in the `shared` mode, the controls will be synchronized across all the machines.

For instance, for processes such as sound generators or filters, this simply means that a process in `free` mode can execute with different state for its user-interface controls across the session. In `shared` mode, a change of control from a client is applied to all the other machines in the network.

Likewise, in `free` mode, a scenario process will be able to execute with different interval speeds, conditions and interactions in all the clients executing it. In `shared` mode, the interval speeds, expression resolutions, will be synchronized across the network through the mechanisms described above.

4. DISTRIBUTING DATA

An open question which had been evoked in [2] was the distribution of the live, run-time data of the score: the original implementation did not consider the transfer of things such as OSC parameters, sound or video streams: it only focused on the distribution of the temporal semantics, and users had to define the transmission of controls through separate systems such as NetJACK[11].

We introduce here a new set of processes in *ossia score* which leverage the underlying networking session to enable scores to embed media data transfer semantics through user-friendly objects. These processes are called Netpit: Message Pit, Audio Pit and Video Pit. Right now, their implementation focuses on ease of implementation for the sake of prototyping and experimenting with writing scores and as such uses WebSockets for data transfer instead of WebRTC [1] . The final implementation aims to use WebRTC as it is the only real-time audio-video protocol supported by web browsers: there is no other choice if we aim for our system to be fully functional in the web.

These processes have a combining semantic. They have one input port, and one output port. Their operating rule is: the data that a Netpit process instance receives as input is combined with all the clients executing this process with a user-chosen function: summing or concatenating channels for audio, taking the mean for real-valued control sig-

[1] An implementation using WebRTC, more optimized for audio and video transfer, has been started but is stuck on a bug in the underlying GStreamer library implenting the WebRTC protocol: `https://gitlab.freedesktop.org/gstreamer/gstreamer/-/issues/1261`

nals, applying a blend for videos, etc. For instance, if two clients execute a given Netpit audio process at some point during the execution of a score, with each their custom microphone input set in the input port, both clients will by default get a sum of both microphone signals as output of the process. There is as of now no latency compensation in the system.

This, combined with the notion of group, enables to transparently define a set of distributed behaviours: users do not have to fiddle with defining explicit distribution semantics such as one would do in environments such as Max/MSP, by using custom network send / receive objects for each signal they want to transmit; here, the objects scale automatically to the network topology used and allow he desired distribution semantics inside the dataflow graph engine of *ossia score*.

5. VISUAL LANGUAGE EXTENSIONS

To represent the multiple states of synchronization of the elements of *ossia score*, we propose to introduce a set of alternative symbols to denote the various network-related semantics. Table 1 presents the matrix of possibilities. The idea behind this prototypical design language is to represent the concepts described in sections 3 and 4 as follows:

Inactive elements (e.g. those that are not in a group that a given *ossia score* instance executes) are in grey, while active elements are in colour.

The *free–shared* dichotomy is represented by a connecting line: free elements aren't tied together, shared elements are.

We chose to not represent the *uncompensated–compensated* axis yet as it would double the amount of symbols necessary while only providing marginal benefits in local networks with very short latencies ; the toggle is however still available from the user interface.

The *asynchronous–synchronous* dichotomy straightens the round shapes to evoke the stricter constraints of synchronization related to non-synchronization. Frutiger says about the square in [6],

> (…) a symbolic object, boundaried property, also a dwelling place with the feeling of floor, ceiling, walls, protection, etc. (p.43)

The notion of readiness which matches well with our *free* and *uncompensated* and *asynchronous* semantics is also associated with the circle by the same author:

> The most used figures are 1 and 0, two signs that have highly differentiated forms: the straight line and the circle. Once again we come across the binary principle: 1 = notch, cut, hardness (two visible stroke endings); 0 = emptiness, readiness (no beginning or ending). (p.211)

The most restrictive and synchronized case should be the same as the original, non-networked version, as a network with only one computer can be considered as fully synchronized with itself. It would be of course possible to argue the opposite: that a network with a single score instance running is equivalent to the *free* case. However, using the original symbols in the *free* case would likely have implied that the *shared* cases would have to bear additional visual elements to denote the stronger semantics synchronization semantics that they carry: experiments in this direction made the visuals too heavy and harder to read. In short, we opted for deconstructing the existing elements instead of constructing new ones, with the sole exception of the condition becoming "straighter": debate is ongoing about changing its appearance to the squared version in the non-networked mode of *ossia score*.

These visual language extensions are still ongoing a prototyping and experimentation process in *ossia score* and may still evolve in terms of representation, with the goal being making the transmission of our distribution semantics as obvious as possible to the users.

6. IMPLEMENTATION

The system discussed here is implemented in C++, using simple messages over WebSockets through the Qt WebSockets implementation. This enables the system to function over both desktop, embedded, and tentatively through the web with the WebAssembly port of Qt, with a single codebase. The only limitation is that the web version cannot act as a server for the session as web browsers do not allow web page to open network ports.

The latency of simple operations such as a control change is overwhelmingly defined by the network's implicit latency, plus the local latency between the network thread of the software, and the audio or video thread which will end up turning the network message into an observable behaviour, which is at most the duration of an audio buffer or screen frame. Since our current implementation is based on a client-server architecture, the network communication latency is always the latency between the originating client and the server, plus the latency between the server and the other clients that will receive the message.

For the synchronized operations discussed in Section 3.2, the latency will increase due to multiple round-trips between the involved clients and server: the overall latency will always be bounded by the latency of the slowest client as the server waits for all the clients in a group to trigger the advancement of the score.

The system has been tested so far with up to 8 instances on a local network, over Wi-fi and Gigabit Ethernet, and with two instances over the internet. Our naive video and audio transfer implementations however are only useable with correct results on a local network so far, and would require using specific codecs with optimizations (such as GPU encoding / decoding for video) making them suitable for use over the internet.

7. DISTRIBUTION EXAMPLES

We cover in this section a few examples of usage of the new data distribution primitives and distributed scores.

Element	Free		Shared			
			Uncompensated		Compensated	
	Inactive	Active	Inactive	Active	Inactive	Active
Triggers						
Conditions						
Processes						

Table 1: Proposed visual syntax for the distribution specification of *ossia score*'s elements. The icons for processes are displayed in their header, and are also used for intervals.

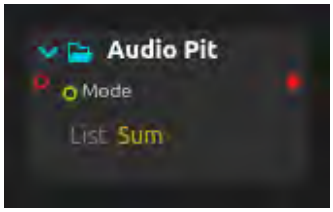

Figure 3: Distribution of a mixed audio stream across machines: this is the entire score.

7.1 Sending data between machines

In this example, we want to showcase the simplest possible example of distributing data: there are two machines, both want to hear a mix of all the musical instruments playing on the network: for instance the scenario is the classic networked band rehearsal. Fig. 3 shows how simple it is: this behaviour is the one that the "Audio Pit" object will provide by default. All the machines can be in the same default group, `all`. They will all execute the process: its input is set to the address `audio:/in/main` which represents the default input of the sound card of the computer. The process will pull the audio input, and sends it to all the clients which then each perform the downmix with the data they received [2] .

7.2 Combining control data across a group of players

In this example, we wish to combine MIDI CCs, gamepad inputs or even GUI widgets which would be controlled by musicians on cheap hardware, in order to generate audio on a computer with a powerful sound card. Fig. 4 shows how such a setup can be achieved.

The groups are defined as follows: there is a `players` group, and an `audio` group. The "Knob" process is associated with the first group: only the clients in this group will execute this process. The second group is associated to the entire bottom interval: it applies recursively to the processes within. During execution, all the clients which registered themselves in the `players` group will have their "Knob" send an input message to the "Message Pit" process, which by default is assigned to the `all` group, which everyone is part of by default. Then, the score instance which registered itself in the `audio` group will receive a list containing all the input frequencies of the clients. It applies it to a simple polyphonic synthesizer and effect chain and renders it on a sound system.

[2] Our current research prototype implementation uses a central server; a production implementation using WebRTC would be P2P

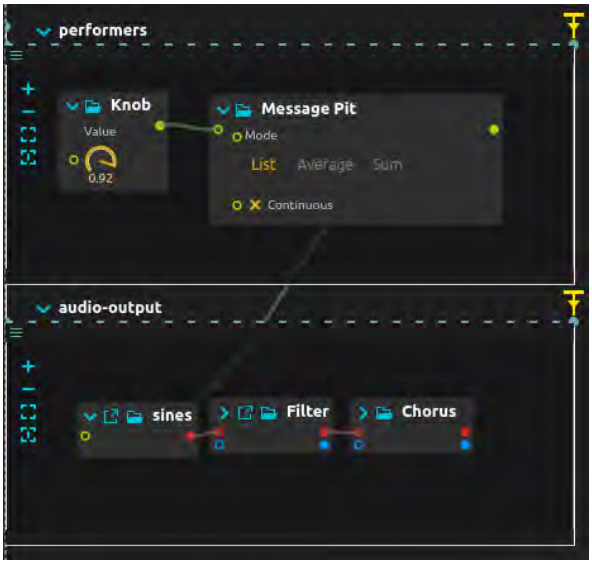

Figure 4: Clients-to-server communication.

7.3 Duplicating an input

This example is a sort of contraposition of example 7.2. A single machine produces an input, which is going to be broadcast. A set of machines will synthesize either sounds or visuals from this input, depending on their groups. The group organization is actually almost the same as example 7.2: the only difference is that a `video` group is added, in order to distribute audio and video rendering on different hardware. Fig. 5 shows the score. Note that the system has no way to specify the number of participants, which is purely a property of the actual performance that will take place: a performance could use one client, two audio machines and two video machines, while the next performance could actually average the input of four clients, and only use a single audio and video output clients, without chang-

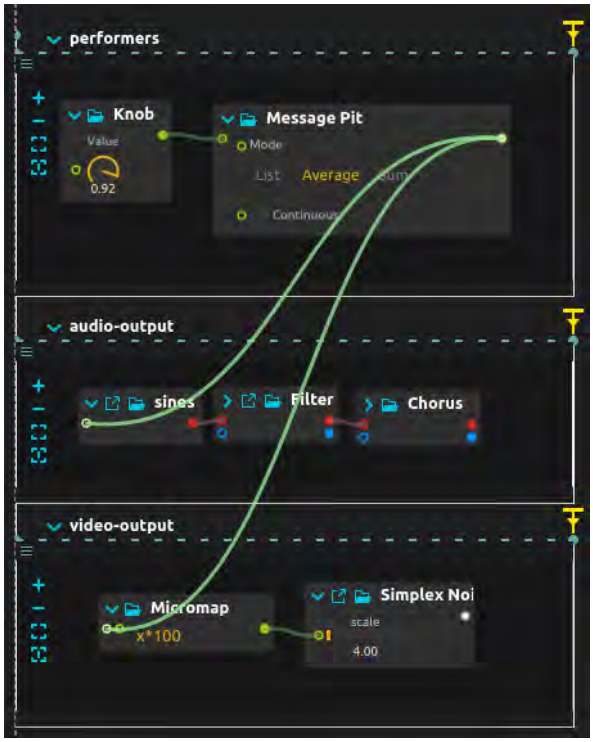

Figure 5: A single client's input data will be broadcast to all the machines in the audio and video groups.

Figure 6: Excerpt of the score demoed over the internet at SMC2022 – this was before the introduction of the visual language.

ing anything to the score. Conceptually, the score specifies only sections and not individual players, just like scores for orchestras or choirs usually does not specify an exact number of second violins or singers.

7.4 Score for SMC2022

A successful prototype demonstration happened for the 2022 Sound & Music Computing Conference (fig. 6), located in Saint-Étienne (France). The score was distributed between this location and the home of the author in Peyriac-Minervois, 357 km away and featured both interactive triggering and distribution of OSC data. Audio and video transmission had not been implemented yet and have so far only been tested on local networks.

This score was before the introduction of the visual language, it featured multiple groups and controls, as well as a shared evolution of the interactive timeline over the network: the automations at the top would for the first part all be synchronized together as the main scenario was in "shared" mode. Then, a sub-scenario *D* was in free mode: inside this hierarchical level, the trigger's triggering time would be independent across machines.

7.5 Polyphony, sharing and visual language

This example (fig. 7) presents some of the visual language extensions introduced in Section 5 The root scenario is shared: execution speed and triggers will be synchronized across the entire network (the scenario is assigned to the special "all" group which encompasses all clients). Then, its score is as follows: intervals *A* and *B* are assigned to respective groups of the same name. Both contain an audio generator. The generator of group A is shared: if any client changes one of the control, this is passed on all the other machines which execute it. The generator of group B is free: every client can change its controls independently. Clients that are neither part of *A* nor *B* will not execute any of these sound processes. Whenever the interactive trigger point between A and the Video interval is triggered, all the machines start executing the video effects (GLSL shaders) locally. The Scenario which contains *GFX1* and *GFX2* is in `free` mode: all the machines can execute it independently. That means for instance that the speed of execution of *GFX1* and *GFX2* can differ across all clients, and that the interactive trigger point between both isn't synchronized across the network. The output of the video generator processes is connected to an *Echo Trace* process in `shared` mode: its controls will be shared across the network. Finally, in the video effect chain, the *VVMotionBlur* video effect is in free mode: its controls are also independent across the network.

8. CONCLUSION

This work introduces a notation system for intermedia composers to author distributed behaviours in a simple graphical environment. The interactive system has been tested both in local networks and over the internet.

The remaining implementation goals are to make sure that the system works correctly from web browsers through *ossia score*'s WebAssembly port, and improving the data protocol implementations to use more optimized protocols than the current WebSockets-backed implementation, which is not well suited to streamed multimedia data such as audio and video. In particular, a pathway we would like to explore, is to automatically adapt the protocols used for au-

Figure 7: A score with various behaviours across various groups. This uses the visual language discussed in Section 5.

dio/video data transmission to the clients connected and the network situation: if the performance does not use web-based clients and is entirely situated on a local network, it would make sense to automatically use NewTek NDI for video transmission due to its efficiency [5]. In contrast, if the performance is done over internet, it would make more sense to use WebRTC for its P2P communication abilities, and support for various NAT-bypassing technologies which would enable its use across for instance academic institution firewalls. A tentative to use the Opus codec for transmitting audio content has also been done, but so far did not provide an acceptable quality / latency trade-off for musical applications.

References

[1] Drake Andersen. "INDRA: A Virtual Score Platform For Networked Musical Performance." In: *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR)*. Hamburg, Germany, 2021.

[2] Jean-Michaël Celerier, Myriam Desainte-Catherine, and Jean-Michel Couturier. "Exécution Répartie De Scénarios Interactifs." In: *Proceedings of the Journées d'Informatique Musicale (JIM)*. Paris, France, 2017.

[3] Jean-Michaël Celerier et al. "OSSIA: Towards a Unified Interface for Scoring Time and Interaction." In: *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR)*. Paris, France, 2015.

[4] Jean-Michaël et al. Celerier. "ossia score 3." In: *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR)*. Marseille, France, 2022.

[5] Andrew Cross et al. "IP Workflow and Scalable Video Performance." In: *SMPTE17: Embracing Connective Media*. SMPTE. 2017, pp. 1–12.

[6] Adrian Frutiger. *Signs and symbols, their design and meaning*. 1989.

[7] Rama Gottfried and Georg Hajdu. "Drawsocket: A browser based system for networked score display." In: *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR)*. Melbourne, Australia: Monash University. 2019, pp. 15–25.

[8] Georg Hajdu. "Quintet. net: An environment for composing and performing music on the Internet." In: *Leonardo* 38.1 (2005), pp. 23–30.

[9] Cat Hope and Lindsay R Vickery. "The Decibel Scoreplayer – A Digital Tool for Reading Graphic Notation." In: *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR)*. Paris, France, 2015.

[10] Leslie Lamport. "The Part-time Parliament." In: *ACM Transactions on Computer Systems* 16.2 (1998), pp. 133–169.

[11] Stéphane Letz, Nedko Arnaudov, and Romain Moret. "What's new in JACK2?" In: *Proceedings of the Linux Audio Conference (LAC)*. Utrecht, Netherlands, 2009.

[12] Miller Puckette et al. "Pure Data: Another Integrated Computer Music Environment." In: *Proceedings of the Second Intercollege Computer Music Concerts*. Tokyo, Japan, 1996.

[13] Scott Smallwood et al. "Composing for laptop orchestra." In: *Computer Music Journal* 32.1 (2008), pp. 9–25.

[14] Dan Trueman. "Why a laptop orchestra?" In: *Organised Sound* 12.2 (2007), pp. 171–179.