Koryphaîos: A Patchworked Compositional Environment for Distributed Music Systems

Aliénor Golvet, Luciano L Barbosa, Etienne Démoulin, Benjamin Matuszewski

Koryphaîos

A Patchworked Compositional Environment for Distributed Music Systems

Aliénor Golvet
STMS Ircam-CNRS-Sorbonne Université
Paris, France
alienor.golvet@ircam.fr

Luciano L. Barbosa
Composer
Paris, France
lucianoleitebarbosa@gmail.com

Etienne Démoulin
Ircam, Production Department
Paris, France
Etienne.Demoulin@ircam.fr

Benjamin Matuszewski
STMS Ircam-CNRS-Sorbonne Université
Paris, France
benjamin.matuszewski@ircam.fr

ABSTRACT

In this paper, we present *Koryphaîos*, an environment that aims to facilitate composition in distributed music systems. This environment, rather than a monolithic and all-in-one solution, proposes to favor interoperability between our own platform (i.e. *soundworks*) and existing tools widely used by contemporary music composers and musicians. More precisely, our aim was to design an environment where the composer could use the tools they already know and master (e.g. the *Bach* library for *Max/MSP*)), to create musical pieces rendered on web-based distributed systems composed of multiple devices such as smartphones or nano-computers. First, we present our design methodology and overall architecture of the proposed environment. Second, we showcase some of the musical possibilities that it currently offers. Finally, we describe a novel and low-level extension of our framework aimed at facilitating communication and interoperability between our web-based framework and existing computer music software (i.e. *Max/MSP*, *Ableton Live*). The *Koryphaîos* environment is open-source[1] and released under the BSD-3-Clause license.

CCS Concepts

•Applied computing → Sound and music computing; Performing arts; •Human-centered computing → Interactive systems and tools;

Keywords

Web Audio, Distributed Music Systems, Authoring Tools

1. INTRODUCTION

The past decade has seen important developments of distributed systems dedicated to artistic and music practices. Indeed, the development of Web APIs, such as *WebSocket*[2] and the *Web Audio API*[3], together with the recent spread of handheld devices and embedded hardware (e.g. Raspberry Pi), has opened new possibilities for designers, researchers and artists to engage, use and manipulate the network as a primary material into their work [6, 21].

However, despite the growing number of artwork, concerts and installations that have been proposed along the years, it can be argued that such approaches and compositional techniques are far from reaching widespread adoption. We postulate that one of the possible reasons for this state of affairs is the lack of high-level and ready to use composition environments that 1) lean on existing composer's skills and practices, and 2) take into account the specificities (e.g. network, number of devices) of distributed systems. Indeed, while on the one hand we have frameworks dedicated to build distributed music systems [4, 15] that lack high-level tools oriented toward composition, on the other hand we have software and libraries dedicated to composition [3, 8] that are not primarily oriented toward the specificities of the Web platform and of network-based approaches.

Composing for distributed systems therefore remains a difficult task that generally ends up with ad-hoc systems and idiosyncratic solutions. The difficulties one must face are twofold: 1) tackle complex design, architectural and development questions and 2) reduce the unbounded creative possibilities afforded by the system to define a creative space that can be artistically manipulated. In this regard, we consider with Magnusson that "In new musical instruments created with general and diverse building blocks, the rationale for creating high-level constraints is primarily to engender an identity, a musical world that is simple, intuitive, and direct" [14]. To put in other words, when dealing with computer music environments, creativity arises from a set of carefully designed constraints and affordances encoded in the software that maps a defined space for musical expression.

In this paper, we propose to approach this problem by creating bridges between existing tools (i.e. *Max/MSP* and *soundworks*) and to improve their interoperability and ease of use in a common patchworked workbench, rather than proposing an integrated and

[1] https://github.com/ircam-ismm/koryphaios

[2] https://tools.ietf.org/html/rfc6455

[3] https://www.w3.org/TR/webaudio/

Web Audio Conference WAC-2022, December 6–8, 2022, Cannes, France.

monolithic solution. Indeed, contemporary music composers being generally familiar with the *Max/MSP* environment, we decided to build upon their existing practices and skills to foster the possibilities of our Web-based distributed music frameworks. Additionally, such an approach aims to put back the tools of composition and creation in the hands of the composers rather than relying on a developper as an intermediary agent, therefore leaving more time and cognitive space for the creative process rather than on solving technical issues.

The *soundworks* framework [15, 16] is a web-based full stack framework that aims to facilitate the prototyping and creation of networked multimedia applications. The main objectives of the framework are threefold: 1) help the scaffolding of applications, 2) provide a simple way to maintain a coherent and distributed state of the application in real-time, 3) provide a possibility of extension by the implementation of a plugin system. However, while *soundworks* has successfully been used in a variety of applications and artistic pieces, the framework does not provide any dedicated interface for computer assisted composition, which required us to program ad-hoc solutions, therefore impeding the composers' agentivity. Additionally, *soundworks* lacked a simple and easy to use solution to inter-operate with existing tools using protocols widely used in computer music software programs such as OSC. The application and underlying component described in this paper represents a first step to fill these gaps.

In this project, we also considered it important to approach our question from different perspectives within a heterogeneous team composed of persons with multiple backgrounds, skills and activities. Finally, while we recognize that the environment as described here, necessarily embodies some aesthetic and compositional perspectives of one single composer (i.e. our co-author L. Barbosa), we hope the genericity and extensibility of the proposed system could prove to be interesting for other composers and artists as well.

After a short review of the related works and a presentation of our design methodology (Sections 2 and 3), we will describe in Section 4 the design choices and overall architecture of our environment. Then, we will showcase in Section 5 some of the artistic and musical possibilities it unfolds. Finally, we will describe in Section 6 a novel component–that lies at the heart of the proposed environment–dedicated to facilitating communications between *soundworks* and *Max/MSP*.

2. RELATED WORKS

In this section we present several tools dedicated to computer-assisted composition that have been proposed over the years. We then present the choice we made amongst these software for our own application.

A number of dedicated software and tools (e.g. *Bach* and *Cage* [3], MaxScore [12]) have been created, often with the help of, or by composers themselves, to manipulate symbolic musical data and scores. For example, *OpenMusic*[4] [8] has been developed at IRCAM since the end of the 1990s. It uses a graphical interface and offers a large range of functionalities for algorithmic composition and usage of digital signal processing. [1].

Another example is *ossia score*[5]. Born from the *i-score* software which has been developed since the late 1990s at LABRI [5]. The software focuses on the sequencing of multimedia events and on the construction of interactive scenarios. It benefits from the embedding of multiple programming languages and its support for a large number of communication protocols (OSC, websocket, etc…) [9].

More recently, the *Bach* package for *Max/MSP*[6] has been proposed by Agostini et. al [3]. *Bach* (and its brother package *Cage* [2]) provides various objects, including graphical interfaces, made for performing low and high-level operations on lists of musical data. *Bach* has been heavily inspired by *OpenMusic* and both environments share a lot of functionalities, but while the latter is more advanced and provides more possibilities and processing power in some contexts, *Bach* benefits from the ability to operate with other elements in the *Max/MSP* environment.

Out of all these options we decided to work with the *Bach* library. We wanted to create a tool as accessible as possible, and a large number of composers are already familiar with *Max/MSP* and use it in their works. Also, as one of our goals was to develop a more fluid and user-friendly communication interface between the two software, we think the architecture developed in our application could serve as an interesting model that could be declined to other *Max/MSP* packages (e.g. score following, MuBu [17], …), either in combination with *Bach* or not.

3. DESIGN METHODOLOGY

We choose to inscribe our methodological approach in the framework of *Meta-Design* [10, 11]. We indeed consider composition, and more generally artistic creation, as fundamentally ill-defined problems in which use cases cannot be fully anticipated at design time. In addition, we consider with Fischer and Giaccardi that "if systems cannot be modified to support new practices, users will be locked into old patterns of use". For these reasons, we believe such an approach could prove to be useful for the design of creative and interactive applications in which novelty, exploration and serendipity are very important aspects. Therefore our goal is not to propose a rigid solution to the problem of networked music composition but rather to develop a design space to unfold a wide range of novel approaches and solutions.

In our view, *Meta-Design* could be summarized by three important characteristics. First, as it can't be completely designed prior to use, the application must be designed to evolve, and moreover to co-evolve with its users. Second, the application should support and provide a learning path from simple user to expert user and ultimately to co-designer of the application. Finally, this process takes place in a model called *Seed-Evolving Growth-Reseeding (SER)*, in which software development is performed during the *Seed* and *Reseeding* phases, while the *Evolving Growth* phase is dedicated at observing and documenting how users adopt and appropriate the application. While the application we present in this paper is still in its early stage, we expect it to provide both a *Seed* and some support for a first *Evolving Growth* phase, potentially providing material for a future *Reseeding* phase.

To tackle these objectives, the development of our application has been guided from the start together with one of our co-author, the composer Luciano Leite Barbosa. We deliberately ignored all questions of interaction and participation except the one of the composers facing such distributed systems, therefore considering the system as a mere audio projection tool. In this frame, one of our first objectives was to re-create and re-implement a piece composed

[4] https://openmusic-project.github.io/

[5] https://ossia.io/

[6] https://www.bachproject.net/

by Luciano in 2018, *Color Fields* for accordion, smartphones and electronics (cf. Fig. 1), into a more interactive and versatile compositional workbench. We also asked him to develop a pool of examples with a variety of compositional techniques as well as to propose new features he thought he could need to further simplify his compositional process for distributed music performance. We iteratively developed the first version of *Koryphaîos* with the intent to make all these examples and use-cases fully working.
Alongside these goals, we also asked Luciano to test every version of our application and to deliberately push it to the limits. Indeed, as Tahiroğlu et al. note, "Musicians often use musical instruments in ways that the original designers never intended, probing for hidden affordances" [20]. We also regularly organized test sessions in the studio to test the application under more realistic conditions with a larger number of mobile phones. These test sessions were not only useful to detect technical issues but were also opportunities to discuss with Luciano about new features or modifications. It allowed us to readjust the course of development to incorporate unanticipated elements, which pushed us to design our software architecture in terms of modularity and flexibility to foster rapid testing and addition of new features.

Figure 1: Premiere of *Color Fields* by Jean-Étienne Sotty at the *CENTQUATRE-PARIS*, 2018.

Another design goal was to provide an environment that hides some low-level aspects (e.g. networking, message routing) to the users, but still provide several entry points at its domain level (e.g. audio synthesis, mapping). As such, a large part of *Koryphaîos* is conceived with the idea that it could provide a "a background against which situated cases, coming up later, can be interpreted" [11], an application that is able to translate the creative endeavor of its users in the language of a network of mobile devices. This approach represents an attempt to lower the technical wall that exists between composers and the network of sound-producing mobile devices, to facilitate the process of co-adaptivity between them. Objectives of modularity and openness guarantee that the network will adapt to the many idiosyncrasies of its users but in return, we hope that the relationship with the network (with its specific capacity to question traditional music boundaries [6]) created through *Koryphaîos* could influence composers to reinvent their practice.

4. DESIGN OVERVIEW

Guided by these objectives, we developed a *soundworks*-based application for composing distributed music pieces using the *Bach* library in *Max/MSP*. As an overview, *Koryphaîos* is built around a local network of devices, at the center of which lies a *Node.js* server to which *Max/MSP* and the mobile devices can connect. The *Node.js* server receives the score information from *Bach* and *Max/MSP* though OSC and dispatches this information to the connected Web client through WebSocket channels (cf Fig. 2).

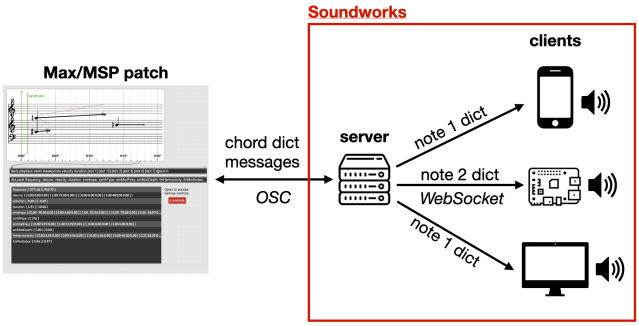

Figure 2: Diagram of the communication between the different parts of *Koryphaîos*.

In the following section we present the application from a design perspective. We start by presenting the composition interface available as a *Max/MSP* patch. We then present how the application produces music out of an array of mobile devices by detailing the communication process over the server and the custom audio engine we developed using the *Web Audio API*. Next, we present a variety of functionalities the application provides for monitoring and control to facilitate its usage both in studio and in concert situations. Finally, we detail several features aimed at fostering its appropriation and customization by users, potentially opening doors for future evolutions.

4.1 An Interface for Composition

From the point of view of the composer, our application is primarily seen as an interface for composing distributed music performances through a *Max/MSP* patch (Fig. 3). The core element of the patch is a `bach.roll` object provided by the *Bach* library. A `bach.roll` is presented as an interactive score sheet which you can edit to place musical notes. Each note can be associated with arbitrary metadata (i.e. using "slots") of different types (e.g. enveloppe breakpoints, modulation parameters, text, filenames). As the score contained in the `bach.roll` object is played in real-time, its output notes are collected by *Koryphaîos*' `kp.to_soundworks` object and formatted as a *Max/MSP* dictionary. The user can freely define which data is to be collected from the `bach.roll` object and to which parameters in *Koryphaîos* they are mapped to by sending a list of parameters to one of the outlet of the `kp.to_soundworks` object. The dictionary then created contains all the desired data for sound synthesis over different synthesizers (e.g. AM, FM) developed using the *Web Audio API*, for instance: synthesizer to use, frequency, velocity, duration, envelopes, synthesizer parameters, etc.

We designed *Koryphaîos*'s *Max/MSP* objects so that lower-lever coding from the composer can be avoided. Any note data sent out by the `bach.roll` object is automatically formatted to be consumed by the rest of the application.

4.2 The Audience as a Speaker Array

Each note information is sent and parsed from the *Max/MSP* patch to the *Node.js* soundworks server through the generic `soundworks.shared-state` object built on top of OSC and presented in Section 6. Upon reception on the server-side, the notes are tagged with a synchronized timestamp and dispatched to all connected clients for rendering using Web Audio synthesizers. By default, the application currently includes different dispatch strategies: **sendAll** (all notes received are sent to all connected clients at

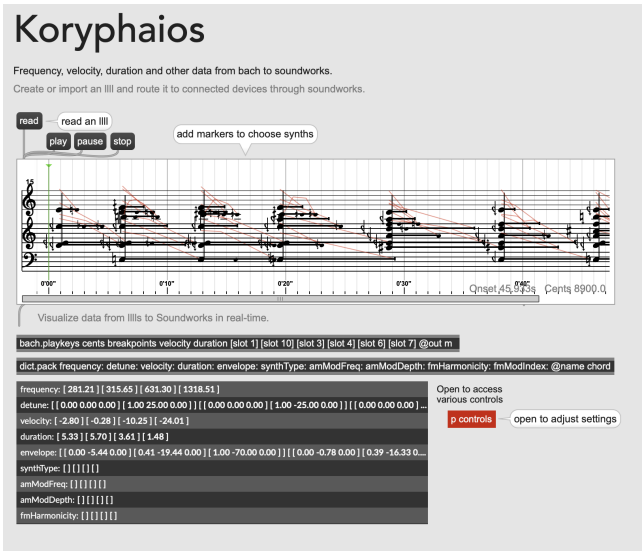

Figure 3: Example of the main composer interface in *Max/MSP* using the *Bach* library in *Koryphaîos*.

the same time), **randomSpread** (the n notes of a chord are split between n random groups of clients of the same size), **randomPoint** (any incoming chord is sent to a single randomly-chosen client).

Upon reception of the time-tagged note by the client, the latter creates an instance of the specified synthesizer (predefined or user-defined), and schedule its rendering using a synchronized scheduler created thanks to the `@ircam/sync`[7] library, which achieves clock synchronization up to 5 ms [13]. By default, the application proposes 5 types of generic synthesizers: a basic sine synth, an AM synth, a FM synth, an audio buffer player and a granular synth.

The synthesizer instances are finally piped through master buses for balance and volume controls.

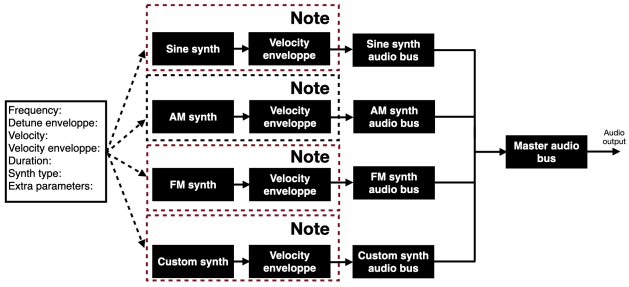

Figure 4: Graph of the audio path within the application. Upon reception of the score information, a `Note` object is instantiated, containing a synthesizer instance and a velocity enveloppe. The `Note` is connected to the corresponding synthesizer's bus which is connected to the master bus. Finally the output of the master bus is sent to the audioContext's destination.

4.3 Contexts, Control and Feedback

The application has been designed to be used both in the studio and in concert situations. To fulfill the multiple and sometimes contradictory requirements of these different contexts, we decided to provide multiple access to the same functionality as well as complementary information from different entry points.

For example, alongside the *Max/MSP* interface, we developed a browser-based *controller* interface. This interface is composed of different parts useful both for monitoring and control:

- A text box that logs any incoming note on the server, which is useful to monitor the proper functioning of the application both in working and in concert situations.
- Buttons to switch between available dispatch strategies for the incoming notes on the fly.
- Master and synthesizer-specific bus controls containing each a mute button and a volume slider (cf. Fig. 5). The Master also exposes two sliders for controlling the frequencies of a low-pass filter and a high-pass filter. All these controls are also available in the *Max/MSP* patch and their visual display is synchronized over the network. To simplify the control and use of these different interfaces in concert situation, we also implemented possibilities of control over a MIDI device either in the *Max/MSP* patch (using the built-in MIDI map assignment) or directly in the browser using a MIDI map assignment interface developed using the Web MIDI API.

Finally, we implemented a *concert mode* that provides a series of interfaces that guide the public through the performance:

1. Upon connection to the web page of the applications, the participant is brought to a volume test page in which they are asked to set the volume of their phone to a comfortable level. While mainly technical, this step can also be considered and used by the composer as a real introductory part of the piece.
2. This step is followed by a waiting screen showing the names of the piece and of the composer, prompting them to wait for the performance to begin.
3. Upon starting the performance in the controller interface, participants are automatically brought to the main playing interface in which the audio engine is connected to their device's output and reception of notes from the server is activated. We also included a simple visualization that displays the current energy of the sound produced by the participant's device through a full-screen animated gray scale.
4. Once the performance has ended, participants are brought to the end page thanking them for their participation.

4.4 Appropriation and Evolutionary Growth

To support the evolution of the application in the hands of its users, *Koryphaîos* provides various possibilities for customization, inclusion of user-made components and sharing of information.

Koryphaîos allows advanced users familiar with *JavaScript* to program custom implementations of several levels of the application. Thanks to *soundworks*' *scripting* plugin[8] , user-made scripts can be created and modified on the fly, without having to restart the application or the network. Since user-made scripts are stored locally as a single file they can also be easily shared among users. At the time of writing, the application only supports user-made synthesizers and dispatch strategies but we plan to make the widest range of technical aspects customizable including the visualization animation when playing sound in the performance screen and the

[7] https://github.com/ircam-ismm/sync

[8] https://github.com/collective-soundworks/soundworks-plugin-scripting

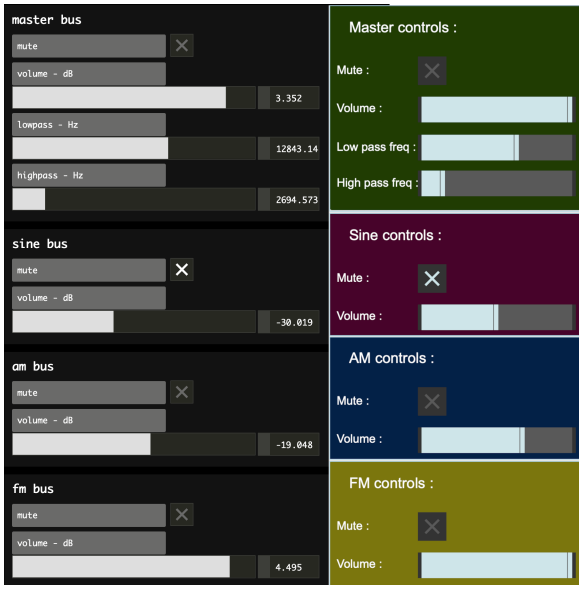

Figure 5: A part of the controller interface: audio bus controls in the browser (left) and in *Max/MSP* (right)

sound to be played during the testing phase in concert mode.
Script creation and edition is made possible by using text editors available in the controller interface in the browser (cf. fig 6). Upon creation of the script, any user-made component is treated as any other component in the application. For instance, a user-made synth can be called up by its name in the `bach.roll` object and a dedicated audio bus with GUI in the controller interface is dynamically created. As explained before, the mappings between the `bach.roll` slots and the user-defined synths' parameters can also be defined at runtime in the *Max/MSP* patch by sending specific messages to the `kp.to_soundworks` object.

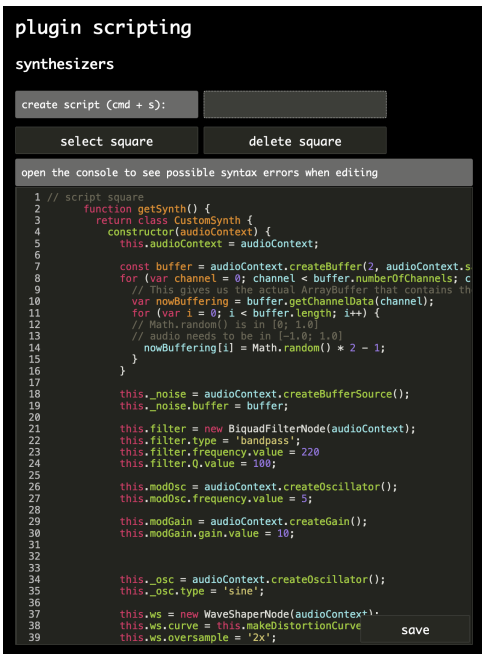

Figure 6: A user-made synthesizer in the scripting interface in the browser.

This process of appropriation by users also extends to more "social" aspects surrounding the application. To this end, we also created an information repository in the form of a wiki on the github repository of the application. It already contains documentation and tutorials on several aspects of *Koryphaîos* as well as the description of the example patches developed by Luciano L. Barbosa (cf section 5). We hope this knowledge base will grow and develop as users may share their own components, musical examples and ideas.

5. MUSICAL EXAMPLES

In this section, we present several musical examples created within *Bach* that showcase the compositional possibilities opened by the application.
First, we present two simple case-studies created during the design and development of the application. Second, we describe the first sketch of a sound installation, *Refraction* and third, we present *Color Fields*, a piece composed in 2018 and rewritten using *Koryphaîos*. All these musical examples have been created by our co-author Luciano L. Barbosa, and explore distributed synthesis techniques that expand the familiar notion of additive synthesis by taking into account the spatialization of each frequency. As a sound can be created or recreated through several sources (e.g. smartphones or other connected devices), the possibilities of distribution of frequencies through devices are therefore numerous, including random distribution, single or multiple frequencies per device, organization of devices into groups, etc. The resulting sound has an intrinsic immersive quality, as a high number of sound sources are used and these sources can be easily spread in the concert space.

5.1 Case Studies

A first example of the possibilities of *Koryphaîos*, leveraging also on the *Cage* library for *Max/MSP*, is to read a sound analysis and resynthesize sounds directly with smartphones. In the following example (see Fig. 7), the object `bach.readsdif` is used to read an sdif file of a sound analysis, which is displayed in the `bach.roll`. A number of symbolic transformations, such as time stretching (using the `cage.timestretch` object) or frequency shifting (using the `cage.fshift` object) can be applied to this resynthesis. All these transformations can be tested, rendered and listened to in real-time through connected devices.

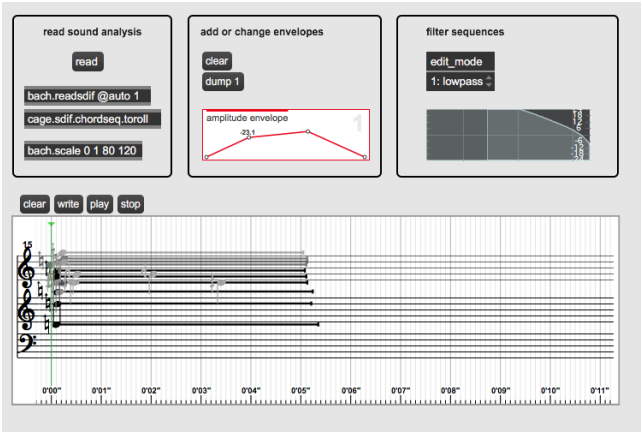

Figure 7: *Max/MSP* example patch of additive resynthesis of sound analysis using *Koryphaîos*.

More complex and generative processes can also be handled by *Koryphaîos*. The example shown in Fig. 8 shows the possibilities of

using generative material in *Bach* and sending it to *soundworks* in real-time for audio rendering. This simple generative patch creates a new sequence when the cursor arrives at the marker `generate seq`, using random values to create a new harmonic sequence.

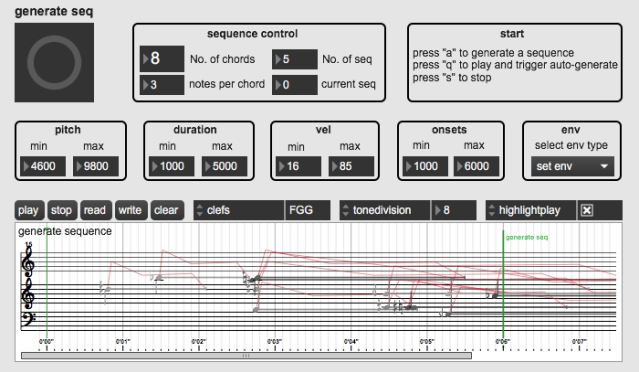

Figure 8: *Max/MSP* example patch of generative music using *Koryphaîos*.

5.2 Refraction

The sound installation *Refraction* is an example of a piece created directly using *Koryphaîos*. The piece is an installation for smartphones that are spread out in the performance room and that includes the participation of the audience. Its main compositional materials are additive synthesis and FM synthesis, with occasional use of AM synthesis, where the sonic result can be envisioned as a distributed synthesis technique in which the rendering of each component is distributed in space amongst devices.

The installation consists of three `bach.roll` objects that are linked to one another through markers, playing independently and overlapping at times. During the compositional process, the frequency materials were input freely on the `bach.roll` and *Koryphaîos* allowed immediate feedback of the resulting sound. Each note on the `bach.roll` (see Fig. 9) contains basic data such as pitch, velocity and duration. The *slots* are used to carry additional data including amplitude envelopes, amplitude modulation values for modulating frequency and tremolo depth, frequency modulation values for harmonicity and modulation indices, and type of synthesizer (e.g. sine wave, am or fm) to be used.

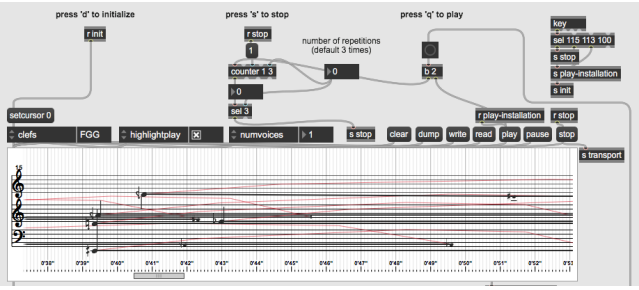

Figure 9: Patch of the *Refraction* installation.

5.3 Color Fields

Color Fields, dedicated to Jean-Étienne Sotty, was composed in 2018 during the *Cursus* program at IRCAM[9]. It was written for XAMP microtonal accordion [19], smartphones, electronics and included audience participation through the use of mobile devices. The electronics were conceived to be diffused mainly through the audience's smartphones, with the aim of spreading the sound throughout the hall and having audience members participate in the sound production of the work. This feature allowed interesting possibilities of sound masses and harmonic blend between the soloist on stage and the sound coming from the devices of the audience.

The piece used additive synthesis as its main compositional material, and each frequency was assigned to a single device from the audience, randomly distributed among the smartphones of connected audience members. The frequency material was created through a number of processes carried out in the *OpenMusic* software, such as sound analysis and transformation of the resulting data, and exported to *Bach*. Other composition techniques used in the piece included free manipulation of harmonies directly within the `bach.roll` object.

In the first version of the piece, using ad-hoc OSC communications and protocol between *Bach* and *soundworks*, frequency and velocity values for each smartphone were hardcoded in advance in flat files directly read by the *soundworks* server. Each event in the main patch would trigger a *bach.roll* containing markers that controlled the start of each harmony stored in *soundworks*. Only the envelope values were sent from *Bach* to soundworks in real-time, handling the overall volume of the synthesis distributed through the devices of the audience. In order to organize the movement of different harmonic fields, the synthesis data was assigned to different groups of smartphones. Such architecture, with data spread between the *Max/MSP* patch and *soundworks*, was however difficult to test and change, making the compositional process slow, cumbersome and error prone.

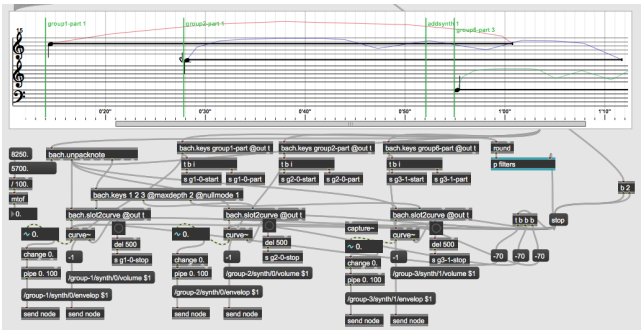

Figure 10: Concert patch of the first version of *Color Fields* in *Max/MSP*.

As discussed in Section 3, being able to recreate *Color Fields* (see the first patch version in Fig. 10) within *Koryphaîos* was one of our main goals from the beginning. As a result, we consider the new updated version of the piece rewritten using *Koryphaîos* to be both more efficient and versatile as the synthesis data and parameters are fully contained in the `bach.roll` and can therefore be manipulated in real-time and from a single place.

In this new version shown in Fig. 11, each *event* of the piece stores one or more `bach.rolls` that are in direct communication with *soundworks* through the `bach.roll`'s playout outlet. Each `bach.roll` connects to a `send` object that routes the data to a subpatch containing the *soundworks* component. Finally, *soundworks* receives values of frequency, velocity, duration and envelope and propagates them to its smartphone clients, which carry out the synthesis themselves. AM and FM synthesis parameters are included directly in the `bach.roll`, unlike the first version of the piece. The update both simplified the patch and increased its stability, validating also that *Koryphaîos* was able to handle the complex harmonic structures of *Color Fields*, such as dense chords and resynthesis.

[9] https://youtu.be/4GuYtPejijI

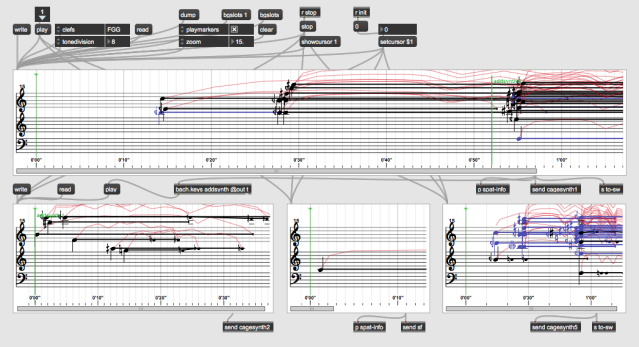

Figure 11: *Max/MSP* patch of the novel version of *Color Fields*, rewritten in *Koryphaîos*.

6. REUSABLE COMPONENT

In this section, we present the `soundworks.shared-state` component[10] for *Max/MSP*, a low-level module developed to facilitate and improve interoperability between *soundworks* and *Max/MSP*, and on top of which *Koryphaîos* has been built. First we present the technical details of the component and then describe some of its possible use-cases outside *Koryphaîos*.

6.1 soundworks.shared-state for Max/MSP

The `soundworks.shared-state` component has been developed to facilitate the creation of musical applications relying on both *Max/MSP* and *soundworks*. The main objective of this component is to provide an extension of the *soundworks*'s `SharedState` abstraction [15][11] in order to seamlessly synchronize states with *Max/MSP* `Dict` objects through OSC communication. As illustrated in Fig. 12, the *global* state, owned by the server, can be accessed and modified from both a *Max/MSP* patch and a *soundworks* client, any modification being propagated to all attached clients which can therefore trigger updates based on the new state of the distributed application.

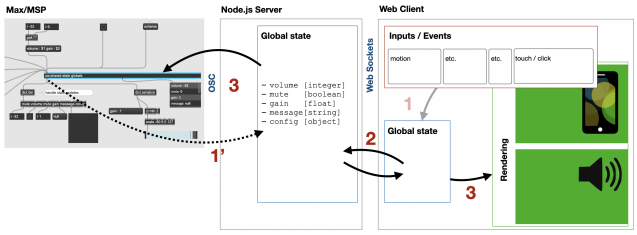

Figure 12: Data flow of state synchronization between *Max/MSP* and *soundworks* using the `soundwork.shared-state` object.

An important attention has been given in providing an API on both sides that 1. is similar enough to be intuitive for users and 2. that take into account the idiosyncrasies of the *Max/MSP* platform and of the *JavaScript* language. Also, some ad-hoc bindings have been implemented to blur the discrepancies between the primitives exposed by the two languages. For example, the component seamlessly takes care of the conversion between the *JavaScript* `boolean` or `null` values, which have no direct correspondence in *Max/MSP*, to more idiomatic types (respectively `integer` and `null` symbol in the *Max/MSP* side).

In its current implementation the component suffers from two main limitations. First, it only allows the subscription to unique and global states created by the *soundworks* server, meaning for example that there is currently no way to subscribe to dynamic states created and deleted by clients. This limitation, more than a technical issue, takes its root in the very different paradigms between *Max/MSP* where everything is statically defined and the more dynamic *JavaScript* paradigm. We are currently investing to mitigate this limitation. Second, the OSC protocol generally implemented on top of the UDP protocol does introduce de facto the limitations of UDP: unreliable message delivery and limited size of the message. These two limitations indeed impede some use-cases such as sharing audio analysis or recordings of multimodal data. We plan to provide an alternative underlying protocol, for example using Web Socket or TCP as the transport channel, to support such use-cases.

6.2 Example Use

Beyond *Koryphaîos*, the `soundworks.shared-state` object has already been used in several demonstrators and applications. For example, we successfully used it in *Max for Live* devices within the *Playground*[12] application designed with the composer Garth Paine, allowing the composer to create, record and replay Live clips controlling some distributed parameters of the application.

Once the question of the UDP limitations discussed above is handled, we also plan to use the component in conjunction with the *MuBu* [17] package for *Max/MSP*. Such link could open interesting rapid prototyping perspectives in several areas such as *concatenative synthesis* [18] or *gesture design* [7], for which all the building blocks and algorithms are not yet available in the *JavaScript* ecosystem.

7. CONCLUSION AND FUTURE WORKS

In this paper, we introduced *Koryphaîos*, an application for networked music composition. *Koryphaîos* is built on top of and aim at creating a bridge between two existing libraries: the *Bach* packages for *Max/MSP* that provide tools for computer assisted composition and the *soundworks* framework which is dedicated to the development of distributed multimedia applications. In this work we followed a user-centered and meta-design approach, emphasizing and promoting the heterogeneous nature of our team composed of researchers, developers, computer-music designers and composers.

Koryphaîos is designed around multiple connected parts. The first one is a composition interface in a *Max/MSP* patch, which at its core is a `bach.roll` object that sends out note data that is encoded and sent to the rest of the application via OSC communication. The second one is a *soundworks* application that, upon reception of the note data, dispatches them to the connected devices for Web Audio rendering. The application also provides functionalities for monitoring and control in a concert context available both in a *Max/MSP* patch and in a dedicated page in the browser. *Koryphaîos* was designed with appropriation and customization by its users in mind. It thus provides options for the development of user-made components, potentially opening for their sharing between users. We then presented different examples using a variety of compositional techniques (distributed additive synthesis, generative music, spectral analysis and resynthesis) that shows the flexibility of *Koryphaîos* for composing in a distributed context. Finally we introduced a novel low-level component, on top of which *Koryphaîos* is built, created to simplify communications and maintain coherent states between *Max/MSP* and *soundworks*.

The current version of *Koryphaîos* leaves room for improvement

[10] Available in https://github.com/collective-soundworks/soundworks-max

[11] http://collective-soundworks.github.io/soundworks/common.SharedState.html

[12] https://github.com/ircam-ismm/playground

and evolution. Future works include a better integration of user-defined components in the *Max/MSP* components. For example, due to current limitations of the `soundworks.shared-state` object, audio buses dedicated to user-defined synths cannot be controlled from the *Max/MSP* patch and the user can only rely on the browser interface in such cases. Also, the current implementation of `soundworks.shared-state` suffers from limitations concerning the size of the messages sent to soundworks. We plan to investigate solutions by relying on a different communication protocol. We would also like to expand the options for user customization. We intend to provide more options for scripting by allowing customization of technical aspects such as the visual rendering during performance and the sound to be played during the testing phase of the performance.

Despite these current limitations, we think *Koryphaîos* has the potential to provide an interesting tool to explore novel compositional techniques within distributed music systems, leveraging on existing knowledge and skills of contemporary composers. As such, we hope feedback and idiosyncratic appropriations of the application by new users will help us strengthen and further develop the application.

8. ACKNOWLEDGMENTS

This work has been conducted in the framework of the *SO(a)P* Innovation Project Unit funded by Ircam. The *soundworks* framework has been initiated in the CoSiMa research project funded by the French National Research Agency (ANR, ANR-13-CORD-0010) and further developed in the framework of the Rapid-Mix Project from the European Union's Horizon 2020 research and innovation program (H2020-ICT-2014-1, Project ID 644862). We would like to thank our colleagues at IRCAM for their precious contributions to the project.

9. REFERENCES

[1] C. Agon, G. Assayag, J. Bresson, and M. Puckette, editors. *The OM Composer's Book. Volume1*. Collection Musique/Sciences. Ircam - Centre Pompidou, Éditions Delatour France, Paris, 2006.

[2] A. Agostini, É. Daubresse, and D. Ghisi. Cage: A High-Level Library For Real-Time Computer-Aided Composition. In *Proceedings of the International Computer Music Conference (ICMC)*, Athens, Greece, 2014.

[3] A. Agostini and D. Ghisi. A Max Library for Musical Notation and Computer-Aided Composition. *Computer Music Journal*, 39(2):11–27, June 2015.

[4] J. Allison, Y. Oh, and B. Taylor. Nexus: Collaborative Performance For The Masses, Handling Instrument Interface Distribution Through The Web. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Daejeon, Korea, 2013.

[5] A. Allombert, M. Desainte-Catherine, and G. Assayag. Iscore: A system for writing interaction. In *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts - DIMEA '08*, page 360, Athens, Greece, 2008. ACM Press.

[6] F. Bevilacqua, B. Matuszewski, G. Paine, and N. Schnell. On Designing, Composing and Performing Networked Collective Interactions. *Organised Sound*, 26(3):333–339, Dec. 2021.

[7] F. Bevilacqua, N. Schnell, N. Rasamimanana, B. Zamborlin, and F. Guédy. Online Gesture Analysis and Control of Audio Processing. In B. Siciliano, O. Khatib, F. Groen, J. Solis, and K. Ng, editors, *Musical Robots and Interactive Multimodal Systems*, volume 74, pages 127–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[8] J. Bresson, C. Agon, and G. Assayag. OpenMusic: Visual programming environment for music composition, analysis and research. In *Proceedings of the 19th ACM International Conference on Multimedia - MM '11*, page 743, Scottsdale, Arizona, USA, 2011. ACM Press.

[9] J.-M. Celerier. *Authoring Interactive Media : A Logical & Temporal Approach*. PhD thesis, Université de Bordeaux, 2018.

[10] G. Fischer, D. Fogli, and A. Piccinno. Revisiting and Broadening the Meta-Design Framework for End-User Development. In F. Paternò and V. Wulf, editors, *New Perspectives in End-User Development*, pages 61–97. Springer International Publishing, Cham, 2017.

[11] G. Fischer and E. Giaccardi. Meta-design: A Framework for the Future of End-User Development. In H. Lieberman, F. Paternò, and V. Wulf, editors, *End User Development*, volume 9, pages 427–457. Springer Netherlands, Dordrecht, 2006.

[12] G. Hajdu and N. Didkovsky. Maxscore: Recent Developments. In *Proceedings of the International Conference on Technologies for Music Notation and Representation*, Montréal, Canada, 2018. Zenodo.

[13] J.-P. Lambert, S. Robaszkiewicz, and N. Schnell. Synchronisation for Distributed Audio Rendering over Heterogeneous Devices, in HTML5. In *Proceedings of the 2nd Web Audio Conference (WAC-2016)*, Atlanta, USA, 2016.

[14] T. Magnusson. Designing Constraints: Composing and Performing with Digital Musical Systems. *Computer Music Journal*, 34(4):62–73, Dec. 2010.

[15] B. Matuszewski. A Web-Based Framework for Distributed Music System Research and Creation. *Journal of the Audio Engineering Society*, 68(10):717–726, Dec. 2020.

[16] N. Schnell and S. Robaszkiewicz. Soundworks – A playground for artists and developers to create collaborative mobile web performances. In *Roceedings of the Web Audio Conference (WAC'15)*, Paris, France, 2015.

[17] N. Schnell, A. Röbel, D. Schwarz, G. Peeters, and R. Borghesi. MuBu & Friends - Assembling Tools for Content Based Real-Time Interactive Audio Processing in Max/MSP. In *International Computer Music Conference (ICMC)*, Montréal, Canada, 2009.

[18] D. Schwarz. Corpus-Based Concatenative Synthesis. *IEEE Signal Processing Magazine*, 24(2):92–104, Mar. 2007. Conference Name: IEEE Signal Processing Magazine.

[19] J.-É. Sotty and F. Vicens. L'accordéon microtonal XAMP : Gestation, fabrication et évolution d'un nouvel instrument. *La revue du Conservatoire*, 5, 2017.

[20] K. Tahıroğlu, T. Magnusson, A. Parkinson, I. Garrelfs, and A. Tanaka. Digital Musical Instruments as Probes: How computation changes the mode-of-being of musical instruments. *Organised Sound*, 25(1):64–74, Apr. 2020.

[21] B. Taylor. A History of the Audience as a Speaker Array. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Copenhagen, Denmark, 2017.