Nü Soundworks: Using spectators smartphones as a distributed network of speakers and sensors during live performances.

David Poirier-Quinot
UMR STMS
IRCAM-CNRS-UPMC
Sorbonne Universités
Paris, France
david.poirier-quinot@ircam.fr

Benjamin Matuszewski
CICM/Musidance EA1572,
Université Paris 8,
UMR STMS
IRCAM-CNRS-UPMC
Sorbonne Universités
Paris, France
benjamin.matuszewski@ircam.fr

Norbert Schnell
UMR STMS
IRCAM-CNRS-UPMC
Sorbonne Universités
Paris, France
norbert.schnell@ircam.fr

Olivier Warusfel
UMR STMS
IRCAM-CNRS-UPMC
Sorbonne Universités
Paris, France
olivier.warusfel@ircam.fr

ABSTRACT

This paper presents the *Nü* framework. The objective of the framework is to provide composers with a tool to control web-based audio processes on spectators smartphones during live performances. Connecting their devices to a webpage broadcasted by the performer's laptop, spectators become part of the composition: from simple sound sources to active musicians. From a Max based interface, the performer can then control the behaviours of conceptual units, referred to as Nü *modules*, designed for live composition (distributed room reverb, granular synthesis, etc.). Each module is composed of a pair of JavaScript classes – one for the client, another for the server – relying on the Web Audio API for audio processing, and OSC messages for parameters control. Nü is an open source project based on the *Soundworks* framework.

1. INTRODUCTION

The notion of web browser has vastly evolved since the "hypertext interpreters" of the 1990s. Constantly pushed forward by an ever changing Internet, web browsers have become versatile platforms for interactive multimedia applications. Amongst their many features, audio processing has come a long way since the first `<bgsound>` tag used for audio playback. In 2015, the W3C[1] released the first version of the Web Audio API (Application Programming Interface) [8]. This standardised API was designed to provide web developers with cross-platform audio processing tools (mixing, filtering, etc.) that would match modern game engines.

Much research and development have been achieved since concerning web-based audio design and applications. Close to the problem at hand, studies addressing the question of audio rendering on spectators devices for live performances have been published in [2, 10, 4]. The interfacing of web-based applications with music creation software as been studied as well, see for example the Csound based implementation detailed in [9]. The framework discussed herein pairs a Max[2] (visual programming language) interface with the Soundworks framework[3] [6], to provide composers with a versatile solution for the design of live performances relying on spectators devices.

Soundworks is a JavaScript (JS) framework designed to create collaborative and collective audiovisual web-based experiences. A typical Soundworks *experience* will lead users to actively participate in an audio performance with their smartphone, through interaction paradigms defined by the designer. Soundworks is based on web APIs, serving web-pages to clients via a NodeJS[4] server. The core of the framework implements basic data exchanges and connection routines. Features common to most experiences (load audio files, synchronise devices clocks, etc.) are packaged as Soundworks *services*. The ambition of the Soundworks framework is to allow developers and composers to focus on user experience and interaction design rather than JavaScript implementation. Experiences can be either deployed as web pages or Cordova applications[5]. The latter is required to access bluetooth communications and microphone inputs on most modern devices.

[1] W3C website: https://www.w3.org

[2] Max website: https://cycling74.com/products/max
[3] Soundworks git: github.com/collective-soundworks
[4] NodeJS website: https://nodejs.org
[5] Cordova website: https://cordova.apache.org

Web Audio Conference WAC-2017, August 21–23, 2017, London, UK.

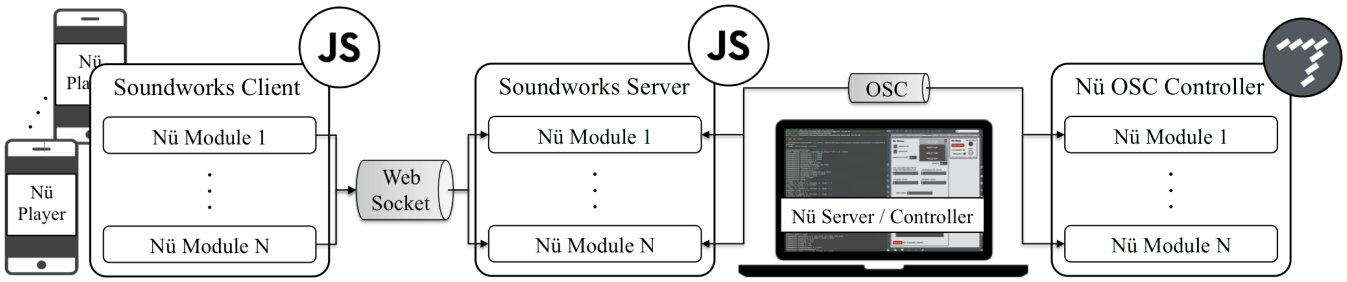


Figure 1: Simplified architecture of the Nü framework. The behaviour of Nü players is controlled from a Max graphic interface (OSC Controller), based on OSC messages forwarded by the Nü server. The whole framework relies on Soundworks: the Nü player class extends the Soundworks Client class, the Nü server is a Soundworks Server, etc. Each Nü module is self-contained and defines its own OSC API. A given module on the player's side will only exchange data with its counterpart on the server. Inter-module interactions are defined from the OSC controller.

The Nü framework is based on Soundworks. It has been designed to allow composers and performers to create experiences involving the audience smartphones. Experiences are created based on the Max programming environment rather than on Javascript routines. Nü applications range from using spectators' devices as a distributed network of speakers, to creating clusters of interactive and sound-capable devices connected to one another.

The remainder of this paper is organised as follows. The architecture of the Nü framework is presented in Section 2. The audio units, or Nü *modules*, that compose the framework are described in Section 3. Section 4 summarises the current state of the framework and outlines its future developments.

Source code, use cases and documentation are available at:
`https://github.com/ircam-cosima/soundworks-nu`

2. FRAMEWORK ARCHITECTURE

This section details the architecture of the Nü framework, illustrated in Figure 1. Nü is based on three core components: a *server*, instances of *players* (*i.e.* web clients), and a Max based *OSC controller*. The framework is organised around JavaScript classes referred to as Nü *modules*. Each module handles a unique conceptual feature, such as "visual feedback", "sensor monitoring", "distributed synthesiser", etc.

The Nü server is a NodeJS HTTP (HyperText Transfer Protocol) server extending the Soundworks Server class. It typically runs on the performer's laptop during the performance, and acts both as a central processing unit and as a bridge between the OSC controller and the players. The OSC controller is a standalone Max patch, connected to the server via OSC (Open Sound Control [11]). It relies on a set of predefined OSC messages, referred to as the Nü OSC API, to address the different Nü modules parameters. The term Nü player refers to any device (smartphone, tablet, etc.) connected to the default web-page served by the Nü server. As for the server, the Nü player is an extended Soundworks client. Each Nü module relies on a dedicated OSC API coupled with two classes, one instantiated by the server, the other by every player.

The design of a performance based on the Nü framework typically starts with a prototyping session. At the early stage of this prototyping, Nü players are instantiated in browser windows emulating spectators devices, running on the composer's laptop. Figure 2 illustrates such a session, involving 16 emulated devices for panned-based audio playback in a room (see Section 3.2). To assess the spatial rendering of the composition, the audio output of each emulated device can be virtualised on the performer's laptop through binaural rendering (see Section 3.9). Once the prototyped application validated, the experiment can be deployed as is for live performance. Browser windows are then replaced by spectator devices (see Figure 3).

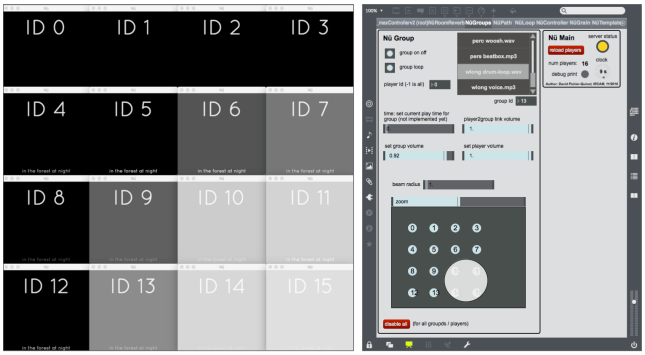
3. MODULES OVERVIEW

As mentioned above, each module relies on two classes: one on server's side, one on player's. Parameters and methods of these classes are addressed from an OSC API. To each module is associated a graphical user interface implemented in Max for simple use-case illustration.

Nü modules all extend a base class, defining 1) general OSC message routing, and 2) client-server parameters synchronisation mechanisms. On the server side, upon reception of a message issued by the OSC controller, the routing mechanism will distribute the message to the concerned Nü module, based on the content of the OSC header. Messages concerning only the Nü players are directly forwarded from server to players.

A synchronisation mechanism has been designed for seamless integration of newly connected players to an ongoing performance. Messages issued from the OSC controller concerning all players are used to update the internal parameters of a mockup player on the server. The mockup player state is broadcasted to new players upon connection. To save and restore the parameters of a given performance between sessions, a per-module preset mechanism has been added to the Max OSC controller.

The remainder of this section details the modules already implemented in the framework. The objective is not so much to list the framework features as to provide a global picture of it's overall ambition. Every module is self-contained and can be used in parallel with others at any time during a performance. Unless stated otherwise, all the sounds played back by these modules are issued from audio files downloaded from the server upon player's connection.



a b

Figure 2: Screenshot of a prototyping session on the composer's laptop. (left) Browser windows emulating Nü players. The screen hue of each player is controlled by its audio output level (see Section 3.9). (right) Nü OSC controller. Specific interface of the "Nü Group" module described Section 3.2 . The mini-room display at the bottom allows to define the sound position in the array of Nü players, along with its spread on neighbouring players (radius of the white circle).

3.1 Nü Main

The Main module handles context synchronisation between the Max OSC controller and the server. It forwards players ID and coordinates, registered in the server, to the OSC controller (see mini-map Figure 2). A mechanism similar to the mockup player mentioned above ensures that any upstream parameter modification made on the Max controller is forwarded to the server upon connection. The Nü Main module also implements a callback for clock synchronisation between both JS and Max context. Amongst other things, this shared clock allows for synchronised audio rendering over both Nü players speakers and a full scale loudspeaker setup.

3.2 Nü Groups

This module is a straightforward implementation of an audio panner. Its Max patch is illustrated Figure 2, where the composer can define the position of a sound in the network of speakers formed by the Nü players. The underlying OSC API and associated JS methods simply control the playback of audio files. Besides start and stop, each audio file track volume can be defined on a per-player basis. The playback is synchronised throughout players based on the Soundworks *sync* service [7, 3].

3.3 Nü Loop

The Loop module is a distributed sequencer. An $N \times M$ matrix in the Max controller is used to distribute N available audio samples over M time steps. The sequencer period and time step can each be modified on the fly, forwarded to the players upon modification. Each player possesses its own sequence matrix. Inter-player synchronisation is once more based on the Soundworks sync service. Internal playback and scheduling relies on the *AudioBufferSourceNode* of the Web Audio API.

3.4 Nü RoomReverb

The RoomReverb module simulates the propagation of an acoustic wave in a virtual room. Said room dimensions are defined from the interface of the OSC Controller. Its walls are "drawn" around the players, on a mini-map like the one of Figure 2. The performer will then typically "emit" a sound (*i.e.* click on the mini-map) at a given position in the room. This position, along with the room dimensions are forwarded to the server prior to any audio rendering. Based on a shoebox model [1], the server estimates the image sources associated with the propagation in the room from that position. From this list of image sources, the server computes an Impulse Response (IR) for each player according to its position in the virtual room. Each IRs is then sent to its associated player. Upon IR reception, the player convolves it with audio data loaded from an audio file, and stands ready for audio rendering. But for player's CPU, there is no limit on the number of sounds that can be simultaneously emitted in the room.

The audio rendering is triggered by a "rendez-vous" message, broadcasted by the server when it receives the emission position from the OSC controller. This message basically defines a future point in time where every player should start rendering its internal convolved audio buffer. Along with room dimensions, room materials and scattering coefficients can be defined, to modify the acoustics of the simulated room. The propagation speed ($m.s^{-1}$) and attenuation ($dB.m^{-1}$) can be defined as well to produce either realistic or artificial (*e.g.* with negative propagation speed) reverberation patterns.

A *time bound* parameter allows to define whether the propagation time is to be considered when tapping into the input audio buffer. If set to zero, each IR tap will generate a full copy of this audio buffer, with a time offset corresponding to the tap delay, in the player's output buffer. If set to one, the propagation time of each specific tap (*i.e.* image source) will define an initial reading offset for that tap in the input audio buffer. This mechanism is depicted Figure 4. An associated "segment percentage" parameter allows to define the duration of the audio buffer copied for each tap relative to the duration of the input audio buffer.

3.5 Nü Path

The Path module has been designed to draw audio trajectories through the network formed by the Nü players. The trajectory is drawn on the same mini-map as for the previous modules. Drawing a trajectory on the OSC controller defines a set of emission points in the room. This set is sent to the server upon trajectory completion. Based on the same image source logic as the RoomReverb module of Section 3.4, the server will define a per-player IR based on these emission points and the player position. On the player side, the IR is convolved with an input audio buffer to generate the local pattern for the current path.

The Path module relies on abstract emission points rather than on players ID to define a trajectory. This allows to define trajectories independent from players position and eventual movements during the performance. As for the RoomReverb module, propagation speed, gain, time bound, and segment percentage can be modified to alter the rendered audio trajectory. Several path can be predefined, stored as Max preset, and launched simultaneously during the performance.

3.6 Nü Synth

The Synth module relies on the *OscillatorNode* of the Web Audio API to implement a distributed synthesiser. Its OSC API allows to attach each note of a midi keyboard in the OSC controller to one or more players. Note volume, periodic waveform (square, sawtooth, harmonic coefs, etc.), attack time, and release time can be defined on a per-player basis.

3.7 Nü Grain

This module relies on a granular synthesiser, distributed on synchronised Nü players. On the players side, the module implements a basic granular synthesiser. At startup, the module splits an input audio file into segment buffers. The duration of these segments depends on a "relative duration" parameter defined from the OSC controller. Once started, the synthesiser plays one segment per "period", the duration of which is defined from the OSC controller as well. The played segment is selected based on an "energy" value, defined as the summed output of the device acceleration sensor on all three axis. The greater the energy, the louder the selected audio segment.

Players synthesisers are synchronised in time. Their energy vector can be partially or fully overwritten by the server, based on a 0 to 1 value issued from the OSC controller. This mechanism allows to use the Nü Grain module either as a unique granular synthesiser which output is rendered over the players speakers, or as an ensemble of granular synthesisers each playing its own patterns based on a common pulsation.

3.8 Nü Stream

The Stream module is used for live synchronised streaming of audio data from the Max OSC controller patch to the Nü players. This module requires the Max patch and the Nü server to run on the same machine.

Nü Stream takes any audio stream generated by the Max patch and writes it as audio file chunks to the disk. The server reads these chunks and sends the associated raw audio data to the players. A timestamp is prepended to each audio data packet for synchronised playback across players. A "rendez-vous" time offset, defined from the OSC controller, is used to anticipate data transmission delay. A player receiving a chunk of audio data after its due playback time will simply discard it.

Designed for dynamic sound design and DAW-based streaming (Digital Audio Workstation) during Nü sessions, this module heavily depends on the bandwidth of the network at hand. Regardless of the network capacity, a minimum delay is to be expected between audio stream emission and playback time, due to the rendez-vous mechanism.

3.9 Nü Output

The Nü Ouptut module acts as a global mixing unit downstream of all Nü modules. The audio output of each module is routed to the input (simple *GainNode*) of this module rather than to the default *audioContext.destination* of the Web Audio API.

The Output module allows to define players master output volume. It implements callbacks on both player and server sides for player-wise recording of a Nü audio session, triggered from the OSC controller. During a recording session, each player stores the audio data streamed through its Output module into an internal audio buffer, while still rendering said output on its speakers. At session's end, each player uploads its recorded buffer to the server that adds them up and writes the resulting audio buffer to the disk.

Figure 3: Live performance reproducing the scenario prototyped in Figure 2.

During virtual prototyping sessions (*i.e.* device-less, see the introduction of Section 3), the *spatialisation* mode of the Output module allows to spatialise the audio output of each emulated player. This binaural spatialisation, based on the JSAmbisonic library [5], relies on the player hypothetical position in the room to spatialise its audio stream. This mode is activated from the OSC controller, from which can also be defined the position of a virtual listener relative to the emulated players. The spatialisation, coupled with room IRs of the JSAmbisonic library, can be used to assess the spatial audio distribution of a given Nü performance over headphones.

3.10 Nü Probe

The Probe module has been designed to access player's sensor data from the OSC controller. Devices *acceleration*, *orientation*, and *touch* data can be uploaded to the server for direct forwarding to the OSC controller on a per-player basis. There is no limitation to the number of simultaneously "probed" players but for the available network bandwidth. A throttle mechanism is implemented to limit the amount of data streamed from the activated players, based on threshold parameters for all three acceleration, orientation and touch values.

Streamed to the OSC controller, sensor data can be used to control the framework modules as would any composer's input. The acceleration data of a subset of players can for example be used to control the Nü Grain (see Section 3.7) energy parameters of there neighbours.

3.11 Nü Controller experience

The Controller experience resembles the Probe module in that it also grants access to a device's sensor data in the Max OSC Controller. The Controller is however an independent Soundworks experience, on equal footing with the Nü player experience, published by the server on a different port from the default client (*i.e.* player). The Controller experience is dedicated to sensors streaming, instantiating none of the other Nü modules. Unlike the Probe module, the Controller constantly streams device sensors data, and cannot be deactivated from the OSC controller. It has been designed to be used on performer controlled devices rather than on the

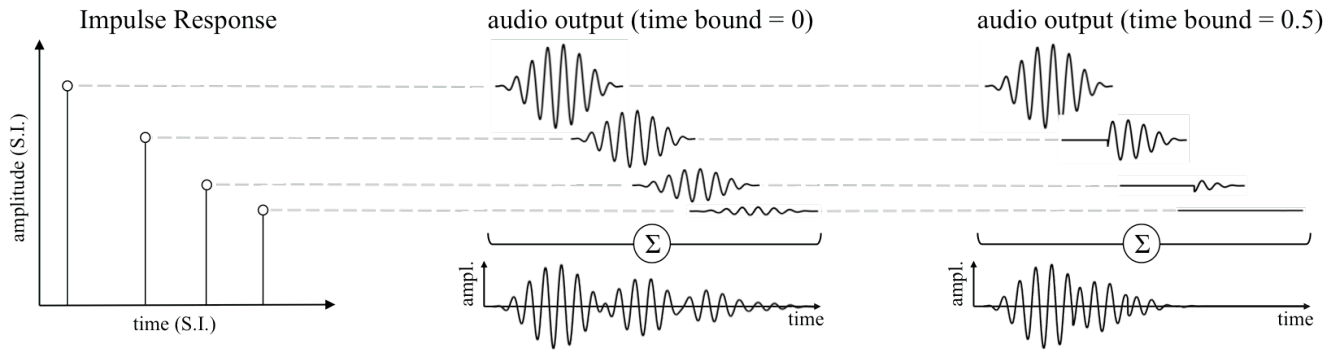


Figure 4: Illustration of the time bound mechanism of the RoomReverb module. (left) Estimated room IR at a given player position. (center) The full input audio buffer is added to the audio output for each IR tap. (right) A read offset is defined based on tap delays and time bound value. The time bound value can be continuously defined in [0, 1] to create different artificial reverberation effects.

spectator's. As for the Probe module, the Controller experience can be used to address the parameters of other Nü modules or to control Max audio processing routines.

3.12 Nü Display

The Display module allows to control simple visual feedback routines on players screens. Besides messages and instructions display, the Display can be used to define, on a per-player basis, *active* and *idle* background colours from the OSC controller. When this visualisation mode is enabled, the hue of a player screen will oscillate between its active and idle colours, based on its audio output level. This hue value, defined on a 0 (idle) to 1 (active) scale, is defined from the summed *ByteFrequencyData* of an *AnalyserNode* of the Web Audio API, attached to the Nü Output module (see Section 3.9). The Analyser min/max frequencies, its dB range, and output multiplier can be defined from the OSC controller. All these parameters can be defined on a per-player basis to create different ambiance or instruction set through the players network.

4. CONCLUSION

This paper presented the Nü framework. Based on the Soundworks framework, Nü has been designed for composers to control web-based audiovisual processes on spectators smartphones during live performances.

Nü relies on the Web Audio API for audio processing on client's end while its server runs in a NodeJS context. The Nü server, typically running on the composer's laptop, publishes a web-page to which spectators can connect their device. Once connected, devices become part of the composition as Nü players: from simple sound sources to interactive musicians. The composer can control the behaviour of basic conceptual audio units, or Nü modules, to produce audio and visual effects through the players network from a Max based OSC controller. A set of control OSC messages has been designed for each module, allowing for a code-free composition environment from the Max interface. The framework has been designed so that Nü modules can easily be modified and new ones created to further support composers needs.

Future work mainly involves the use of the framework in live performances. Extensive tests are planned to assess each module requirements in terms of network bandwidth for real-time interaction and rendering on a given population of Nü players. Out of the scope of the current project, the integration of an accurate indoor location technique to the framework would benefit some of the implemented modules.

5. ACKNOWLEDGMENTS

Soundworks Nü has been developed as a joint effort between both ISMM and EAC teams at IRCAM - Centre Pompidou, in the scope of the CoSiMa research project[6] supported by the French National Research Agency (ANR-13-CORD-0010).

6. REFERENCES

[1] J. B. Allen and D. A. Berkley. Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America*, 65(4):943–950, 1979.

[2] A. Bundin. Concert for smartphones. In *WAC Web Audio Conference*. Georgia Institute of Technology, 2016.

[3] J.-P. Lambert, S. Robaszkiewicz, and N. Schnell. Synchronisation for distributed audio rendering over heterogeneous devices, in html5. In *WAC Web Audio Conference*. Georgia Institute of Technology, 2016.

[4] J. Madhavan, Nihar Snyder. Constellation: A musical exploration of phone-based audience interaction roles. In *WAC Web Audio Conference*. Georgia Institute of Technology, 2016.

[5] A. Politis and D. Poirier-Quinot. Jsambisonics: A Web Audio library for interactive spatial sound processing on the web. In *Interactive Audio Systems Symposium, York, UK*, pages 1–8, 2016.

[6] S. Robaszkiewicz and N. Schnell. Soundworks–a playground for artists and developers to create collaborative mobile web performances. In *WAC Web Audio Conference*, 2015.

[7] N. Schnell, V. Saiz, K. Barkati, and S. Goldszmidt. Of time engines and masters an API for scheduling and synchronizing the generation and playback of event sequences and media streams for the Web Audio API. In *WAC*, 2015.

[6] CoSiMa website: http://cosima.ircam.fr

[8] B. Smus. *Web Audio API: Advanced Sound for Games and Interactive Apps.* " O'Reilly Media, Inc.", 2013.

[9] R. Vinay, Ashvala Boulanger. Building interactive systems with websockets and csound. In *WAC Web Audio Conference*. Georgia Institute of Technology, 2016.

[10] B. Walker, William Belet. Cross-town traffic 2.0. In *WAC Web Audio Conference*. Georgia Institute of Technology, 2016.

[11] M. Wright, A. Freed, et al. Open soundcontrol: A new protocol for communicating with sound synthesizers. In *ICMC*, pages 1–4, 1997.