Introducing Makkeróni, a web-based audio operating system

Balázs Kovács

University of Pécs
kovacs.balazs@pte.hu

ABSTRACT

Makkeróni is a live coding system running in a browser, simulating a text-based linux shell, and producing sounds with and during running their commands. In this paper the author presents the idea of this web application, and summarizes the main possibilities currently implemented in it, looking forward to the uncovered topics as well. Makkeróni is available for use and experiment on the web, by opening the following URL: http://makker.hu/makkeroni/.

1. INTRODUCTION

With the development of web browser standards like HTML5 in the recent years, the gate opened before multimedia applications to be realized as a web project. Among these standards and recommendations, Webaudio API made possible to create interactive audio projects directly for the web, running in any web browser on a computer or on a mobile device. It can be regarded as a portable and platform-free solution for sound artists to create web installations or other multimedia projects. Among these realizations we can find many creative ideas for composing and/or coding: just take a look at Plink[1] as a good example for cooperative composition and audio game; at the Gibber[2] live coding system as a good example for a browser-run one; or at Earsketch for connecting an audio (net)workstation and coding. These projects helped us to understand sound on the web differently than as a way for music listening or as annoying interface sound of a website; they opened the way for utilize the network to realizing standalone, cooperative audio and interaction works.

Among the above mentioned projects we can found a special territory, where the interface is less graphical than textual, being close to the transport layer of the internet. These text-based web audio interfaces make possible to perform (live) coding process: the user writes code for an interpreter which translates the commands to the language of the browser specification. These web applications run mainly client-side, and also can share the benefits of their networked status with server-side portions, for ex.: file storage on a common server, communicate with other users and so on. Makkeróni is an attempt to connect the user interface not only with a virtual operating system, but also to be part of the real processes and file structure on the server itself running it.

2. BACKGROUND

The idea of Makkeróni came from two directions: on one side, I've been working on linux system modifications for creating the running processes audible (it's called sonification),[3] inspired by the SonicFinder project by William W. Gaver (1989). In this case my insterest contiously focused on low-level approaches for sound synthesis or processing, trying to use system-level processes instead of auditory icons. On the other side I began to create inter-website or real-world ↔ web communication projects, for example a webpage-controlled mechanical device or public light organ projects on the chimneys of the Zsolnay Cultural Quartier, Pécs. These projects opened op the way for starting cross-website multimedia interaction.

From the technical point of view, the latter projects used Pure Data's netserver object[4] on the server side, and phpOSC library by Andy Schmeder[5] on the website back-end. These solutions worked perfectly even with a public server and with clients with any type of network connections; but they lack the flexibility of network connections and the bi-directional communica-

[1] http://labs.dinahmoe.com/plink/

[2] https://gibber.mat.ucsb.edu/

[3] Proc filesystem music, http://kbalazs.periszkopradio.hu/index.php?file=works/2005-lad

[4] https://puredata.info/downloads/maxlib

[5] https://github.com/frequenc1/php-osc

tion between clients and the server, as well a p2p-like communication among clients. After introducing node.js, websocket and socket.io[6], I changed my focus from phpOSC to the node.js-based server, because of its in-built solutions for the previously mentioned problems.

The other benefit of a node.js application is portability: any device with javascript running capabitilities is ready to run it, and it's rather easy to configure the setup for performing peer-to-peer communication. Following this approach, I realized a webcard project (in 2016),[7] including both socket.io and webaudio technologies in order to create a quasi-interactive image and sound manipulation system. After that I moved to realize simple webadio projects (such as Forgattató,[8] a twirl simulator), looking for possibilities in the area of interactive/cooperative audio games and instruments.

Collecting something from all of the above projects, the idea of a web-based sonic operating system was born: a web application that simulates a linux shell - the operating system which is running the webserver itself - while the user is controlling it with commands and their arguments on the client's side. The result is Makkeróni,[9] a text-based audio "operating" system.

In the followings I'm going to present the principles and current possibilities of it.

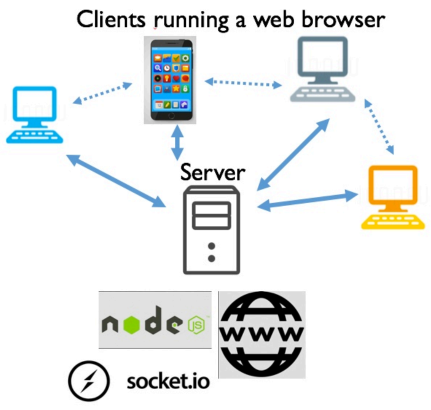


Figure 1. Typical communication in a node.js / socket.io network

3. PRINCIPLES OF MAKKERÓNI

While designing Makkeróni, the following principles were used:

1: create a simple-to-use, easy-to-run live coding system with zero or minimal external libraries; create an useful and practical interface, which makes easy to seamlessly learn the use of the Linux shell;

2: create an audio system which utilizes the efficiency of linux: not only with the command and file structure, but with the low-level approach to processes, data etc. as well. It could be a goal also to create a system, which could be a virtual terminal too for a real, locative server;

3: utilize the native possibility of the web: two-way communication between client and server or client and other clients, data sharing etc.;

I tried to achieve these principles with the following methods:

[6] https://nodejs.org/en/, http://websocket.org/, https://socket.io/

[7] http://kbalazs.periszkopradio.hu/works/webcards/

[8] http://kbalazs.periszkopradio.hu/egyeb/forgattato.html

[9] http://makker.hu/makkeroni/

- Makkerónі is much simplier to use than other live coding systems. It cames from the shell behaviour: one line + enter key = one command. Everything is self-documented in a practical way. The user can be quickly introduced into the application while simply using and interacting with it: help and other command works in order to explain the use. Changelog, reference and other documents are also in-system.

- using existing shell commands (ls, cat, ps etc,) include the operating system commands into the live coding process - which means also that they produce sounds also by running, creating sounds which are reflecting to the result of the command itself;

- using joker characters to randomize parameters;

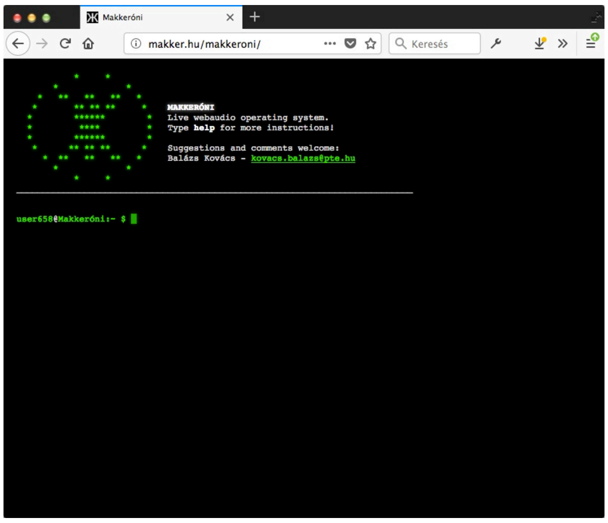
- using command history to quickly recall the previous events;

- using autocomplete for efficiently typing of the commands;

- the system should be capable to be expanded with user-added sounds and presets, available for the other users of Makkerónі;

- the users can save the running processes' list on the server, letting them possible for others to use their results.

After considering the realizations by the standard approach described by Nilson 2007, I decided to find an efficient, web-based text input method, in order to fulfill the interface needs. Therefore Makkerónі has been realized with adapting the JQuery Terminal Emulator by Jakub Jankiewicz,[10] which offers many possibilities to create a shell-like interpreter, parsing commands and arguments, provides mobile-friendly interface etc. As a starting point, I expanded this wonderful framework with extra functions that could perform the audio processing and live controlling. In the next chapters, I'll going to present how it works.



```
MAKKERÓNI
Live webaudio operating system.
Type help for more instructions!

Suggestions and comments welcome:
Balázs Kovács - kovacs.balazs@pte.hu

user658@Makkerónі:~ $
```

Figure 2. Makkerónі start-up screen. Type 'help' for getting started!

[10] https://terminal.jcubic.pl/

4. USING MAKKERÓNI

It's extra easy to use Makkeróni, specially if the user is familiar with the Linux shell (called BASH[11]). If not, it's a good starting point for learning it.

Makkeróni, like BASH, uses the keyboard for interaction. The keyboard is active only when the browser window is in focus. The place for typing commands is equal as in the Linux bash: the cursor blinks after an 'userXY@Makkeróni:~ $' tag, where XY is the system-provided user ID (a number between 0-1000). The first command recommended to try is 'help': it prints out the currently implemented commands with a short description. The command itself generates a random-modulated sinewave too. It's because I tried to unify the usage of the application and the result of the audio commands. For more detailed reference the user can use 'cat reference.txt' also, where 'cat' is the command for printing out the contents of a file: in this case the textfile called with the argument. The result is a bigger textdump of the available commands and descriptions, presenting some examples also. If one is interested what kind of files can be cat-ted, 'ls' - the file listing command - shows the soundfile and textfile names stored on the server. Ls is a bit modified, simplified version of the original linux command: currently it lists all of the subfolders in the home directory placed in the website folder, separating the files only into categories like soundfiles, textfiles, and saved presets.

In Makkeróni, like in BASH, the commands can be autocompleted (by pressing the tab key), repeated (browsing in the history with the up/down arrow and hitting the enter key), furthermore, I implemented auto-arguments (not needed to add all of the options, the system tries to find a solution) and joker characters (* is used for getting random values, while for ex. 3-9 is used for selecting regions of numbers) as well.

4.1. Playing a soundfile

While 'cat' is perfect for dump out the contents of a textfile, 'play', 'loopplay' and 'fadeplay' are designed to play back soundfiles (currently there's no possibility to play a textfile or cat a soundfile). 'Play' simply play it - for ex. 'play bang1.wav' plays back that soundfile with random rate, velocity and panning values. If You want to set them manually, simply attach their values to the command, like 'play bang1.wav 0.7 1.0 0.0'. You can use asterisks to randomize some arguments also: 'play bang1.wav * 1.0 0.' - plays the soundfile with random rate but constant volume and panning. 'Loopplay' and 'fadeplay' are similar to it: they play the soundfile loop'd until a specified time (fadeplay) or until the user stops them manually with the 'stop X' command, where X is a thread number or thread number region. It' also possible to stop a thread by clicking on the asterisk representing the process on a random position of the screen: this is one the few point, where the user can do something with the mouse. It's rather easy to generate a sound cloud by starting many loopplays with typing one time and recalling the history with the up arrow!

The other example of these few points where mouse is required is upload: the user has possibility to place soundfiles on the server: just type 'upload' and click first on the browse then on the upload buttons. The uploaded files are public and ready to use for anybody.

4.2. Synthesizing a sound

Because of its special capabilities, Webaudio API is deeply familiar with sound syntesis techniques also, therefore I needed only feq minutes of work to implement some of the generic nodes. While 'freq' plays a simple sinewave, 'fmfreq' is a complete fm synthesizer: 'fmfreq 440 4 1.0 3 60' plays a 440 Hz sinewave with 1.0 velocity for 4 sec, with a 3Hz sinewave modulator with a depth of 60. There's no panning to set here manually, because it's choosen randomly

[11] https://en.wikipedia.org/wiki/Bash_(Unix_shell)

each time. You can simply use asterisk signs to randomize other parameters too, however there are no advanced modulation lines (like pipe of several synthesis methods) implemented yet. Freq and fmfreq are designed to easily create textures of sounds by simply re-running the command in the save way as described above: just type 'freq' + enter, then ↑ arrow + enter key, and so on. The user can automatize the starting process of these commands as well, as presented below.

A more advanced synth can be called with the 'makkeróni' command. It's a variable waveform fm oscillator with an amplitude-modulated phasor pair, summed with audio rate mathematical operators like '*' (multiple), '/' (divide), '%" (modulo), and their inverted ones. The command itself is not the easiest one to use with its 15 parameters; but it's glitchy, extra-loud, browser-dependent digital sound worth.

4.3. Processes

In Linux, the user can start processes which run on a programmed time basis with 'crontab'; and also there's a way with the 'watch' command to start a process instantly, and running repeatedly. At this time in Makkeróni the 'watch' command is implemented. For example, when 'watch -n 0.9 freq' is typed, an automatic frequency generator process starts, playing a new tone in every 0.9 sec. After starting it, a new process ID is created, letting the user possible to sleep (with the 'sleep' command), resume (with the 'resume' command), restart (with the 're-makker' command), kill (with the more-humanistic 'stop' command) it, while also it's possible to exchange it to a new one (with the 'replace' command). Most of these commands work with such aguments like 'all' or an user-set ID number-regions. For example, 'sleep 3-9' sleeps processes with the IDs #3-#9, while 'resume all' awakes all of them.

Watch can accept the -e parameter as well, which sets the ratio for eroding the value of the repeat time. Value 1.0 keeps the process on the same repeat time, but lower values raise, higher values slower the speed gradually.

The running processes are represented with special signs on the screen. If their activate in some form (playing a new tone or sample), their print a line on the top of the terminal, including all necessary informations. Also, their represent themselves as a green asterisk, floating and blinking in a random position of the screen. It's also possible to query all of the running and sleeping processes with the generic Linux command 'ps', which prints a list of all the processes' IDs, user, time since start and the commands with all of their arguments.

4.4. Starting multiple commands at a same time with batch

During live performance it could be useful to start multiple sounds at the same time. Here's also possibility for starting a command in multiple instances with the 'batch' command. Batch is not the same as the script language of the BASH: it can start only one command at a time in many instances. The user can set the number of instances with the -n argument, and can build complex recoursive command structures, and start them simply with the enter key. For instance 'batch -n 8 freq' generates 8 tones with randomly selected values, while 'batch -n * batch -n 3 freq' starts a random set of triads.

There's a plan for implementing the '&&' command line add-on as well, for starting different commands in the same time; please see changelog.txt if it's realized since the publication of this paper.

4.5. Randomizing parameters

Using * sign of any relevant parameters results a random value, including filenames (in the case of the play command) or any other parameters of a freq, fmfreq and so on. There's also possibility for limiting the random values with numbers behind the * sign, and it's also possible to shift the area of limited random numbers with the + sign. For instance, *777+300 covers a set of random numbers between 1077-1376.

Randomized parameters are used also in the case when commands without arguments have been started. All of the commands in Makkeróni are designed for auto-starting without some or all of the parameters.

4.6. Saving / recalling

Currently there are basic but relevant possibilities for saving a process in Makkeróni. The output of the ps can be piped into a textfile also, in order to save all the processes for further recalling them. Just simply use 'ps > something.txt' to save the data into something.txt. The files are stored on the server, so they're ready to use for different users or computers instantly. It's possible by typing 'cat something.txt > ps', which loads the contents of the previously saved process list textfile, and immadietly starts the commands stored inside the file. The stored files can be listed with the 'ls' command.

4.7. Low-level audio processing

Makkeróni isn't aimed to apply popular audio effects to the audio nodes like delay, filter, reverb. I tried to pursue the low-level DiY tradition of Linux, therefore the audio processes are also ready to hack the audio processes in sample-level. These strongly experimental, extremely sounding 'effects' utilize the scriptprocessor node of the Webaudio API . Scriptprocessors, a currently marked-for-depreciation part of the recommendation can process audio in sample-level very easily. It splits the audio stream into buffers (the buffer size, power of 2, can be a parameter for these audio processors), and lets make possible to apply functions to it. It's specifically easy to produce bitshift-like results! NB: please be aware while starting them - don't use headphones and use very low sound output levels while testing these effects. The author tries to feel himself not responsible for any damages in loudspeakers or ears derived from this fact.

One of these effects is 'blackdeath'. Using it, we can listen to the roar of a reverse-played window divided by a different sample. The result is extremely loud. The user has to set the window size (power or 2 from 256), and a value, and this command is not autocompleted, because of security reasons.

Other similar commands are 'bitshift' and 'degrade'. All of them are extremely loud related to the audio generators introduced above.

Currently there is only one effect to use in the same time. They can be removed from the chain with the 'deffect' command - it's the only autocompleted one among these commands.

4.8. Communicate with other users through the network

Beside file uploading and preset sharing, 'wall' command makes possible for users to send messages to other users present on the website. During loading, Makkeróni transparently connects to a node.js & socket.io server script. The script[12] works as an anonymous chatroom, where all of the accepted messages are broadcasted to the connected users. In the case of 'wall message' the message argument will appear on the title bar of the other browsers' window, and (of course) there's a sound also reflecting to the characters and text length. In the future the author plans to implement remote sound commands also.

5. COMMANDS SUMMARY

Summarizing the above, currently the following commands are present in the application:

- freq: play a sinewave tone. Arguments: frequency (int) length (int) velocity (float) attack time (secundum, float)
- fmfreq: play an fm-modulated sinewave tone. Arguments: carrier frequency (int) length (int) velocity (float) modulation frequency (int) modulation depth (int)

[12] http://mumia.art.pte.hu/webaudio/nodejs/bin/socketio-6096.js

- makkeróni: fm-modulated synth with audio rate mathematical pair. Arguments: carrier frequency (float) waveform (string) velocity (float) lfo frequency (float) lfo depth (float) lfo waveform (string) add-on frequency (float), add-on waveform (string), a
- dd-on velocity (float) modulating frequency (float) modulation depth (float) modulation waveform (string) length (float) operator (*,/,!/,%,!%) buffersize (int, power of 2)
- ls: list contents of the home, soundfiles & saved presets folders
- play: play a soundfile. Arguments: soundfile.wav (string) playback rate (float, 1=normal) velocity (float) pan position (-1.0 = left, 0: center, 1.0 / right)
- loopplay: play a soundfile looped. Arguments: soundfile.wav (string) playback rate (float, 1=normal) velocity (float) pan timeout (seconds, float)
- fadeplay: play a soundfile looped, with linear fade-out
- watch -n sec: repeatedly start play, freq, fmfreq or any other commands. Arguments: -n (float) - repeat time; -e (float) erode ratio
- batch -n instances + command to start: start a command in multiple instances. Arguments: -n (int) - number of instances
- stop: stop one or more loop-play thread or watch process. Argument: thread id (int)
- sleep + thread id (int, region or all): sleep (mute) one, more or all processes
- resume + thread id (int, region or all): resume (unmute) a process
- remakker + thread id (int): restart a loop-play or watch thread. Provides new process id.
- replace + thread id (int) + command: replace a thread with a new command. If there's no thread id provided, it replaces the most recently started process.
- ps: list of running loopplay and watch processes, returns thread ID, user ID, time since start and the (main part of the) command
- degrade, bitshift: sample-level audio manipulation
- upload: upload a sample (wav,mp3 or ogg) into the soundfile folder
- help: short description of commands
- fontsize + int: set fontsize (default: 12)
- statuslength + int: set number of lines in status bar (default: 7)
- clear: clear window
- cat: print the contents of a textfile. Argument: textfile name (string)
- wall: send a message to the other users - they appear on the title and in the javascript console! Argument: message (string)
- degrade, bitshift, blackdeath: digital audio effects
- deffect: disable the previously activated digital effect

General syntax to be applied for most of the commands:

- *: random number (randomize parameters on play, loopplay, fadeplay, freq, and fmfreq)
- ↑ and ↓ arrows: browse command history
- something and TAB key: autocomplete command (for. ex 'lo' + TAB gives 'loopplay' back)

Makkeróni is constantly growing: You can get a current and detailed reference by the 'cat reference.txt' command, while the output of the 'cat changelog.txt' command tells the history as a development diary.

6. CONCLUSIONS

Webaudio is a widely used feature of web browsers, therefore it's on the way of being so transparent as an app developed for a specific operating system. Because of its native network ability, it can be used for 'cloud composing' - or in this case, we could honetly call it thunderstorm composing - opening up aesthetic questions about multi-user artistic communities. The author finds that Makkeróni is a symbol of this process: the simulated operating system runs 'physically' on a similar OS while performing two-way communication among them.

7. FUTURE PLANS

Makkeróni lacks many things: if one feature is implemented, more ideas offer themselves to implement. It's an endless process, and user feedback has an important role. Therefore the author asks live coders to send remarks, ideas, and opinions.

During this, more refined arguments, timing, pipes between commands, raw file playback, server-side features, and basic soundfile manipulating are planned to implement, following the same fashion like the system itself: using Linux syntax and approach. Currently a bunch of basic sequencer commands are under testing, please check Makkeróni refence about their possibilities!

Acknowledgments

The ground idea of the linux-like audio system was born as a Santa Claus' gift made by me in 2016 for Péter Mozsdényi, called MozsiOS - I'm thankful to him for inspiring this type of gift. I'm also grateful to the creator of JQuery Terminal: it would be much harder to achieve thesee results without his work; and to the peer reviewers who helped me to develop and correct this paper. Makkeróni's name is coming from the Makker makerspace in Pécs, organized by our NGO Moiré Association for Culture - I'm happy to develop this webapp for it.

REFERENCES

Gaver, William W. "The SonicFinder, a prototype interface that uses auditory icons." Human Computer Interaction 4, pp 67-94.

Nilson, Click. "Live Coding Practice." Proceedings of the NIME 2007

Roberts, Charles and J. Kuchera-Morin. "Gibber: Live coding audio in the browser." Proceedings of the International Computer Music Conference. 2011.

Roberts, Charles, Wakefield, Graham and Wright, Matthey. "The Web Browser As Synthesizer And Interface." NIME 2013.

Web Audio API, W3C Candidate Recommendation, 18 September 2018. https://www.w3.org/TR/webaudio/