Prototyping web instruments with Mosaicode

André Lucas Nascimento Gomes , Frederico Ribeiro Resende ,
Luan Luiz Gonçalves , Flávio Luiz Schiavoni

[1] Arts Lab in Interfaces, Computers, and Everything Else - ALICE
Federal University of São João del-Rei – UFSJ
São João del-Rei - MG - Brazil

fredribeiro97@gmail.com,andgomes95@gmail.com

luanlg.cco@gmail.com,fls@ufsj.edu.br

Abstract. *Many HTML 5 features enable you to build audio applications for web browsers, simplifying the distribution of these applications, and turning any computer, mobile, and portable device into a digital musical instrument. Developing such applications is not an easy task for lay-programmers or non-programmers and may require some effort by musicians and artists to encode audio applications based on HTML5 technologies and APIs. In order to simplify this task, this paper presents the Mosaicode, a Visual programming environment that enables the development of Digital Musical Instruments using the visual programming paradigm. Applications can be developed in the Mosaicode from diagrams – blocks, which encapsulate basic programming functions, and connections, to exchange information among the blocks. The Mosaicode, by having the functionality of generating, compiling and executing codes, can be used to quickly prototype musical instruments, and make it easy to use for beginners looking for learn programming and expert developers who need to optimize the construction of musical applications.*

1 Introduction

Recently, with the emergence of HTML 5 and the Web Audio API, the web browser became a feasible environment to host a DMI (Digital Musical Instrument). Beyond the acclaimed Web Audio API, HTML 5 brought several others technologies useful to DMIs development like the Web MIDI API, WebRTC, Gamepad API, Websockets, SVG Canvas, WebGL, new tags and elements. Together, all these technologies can be a really powerful framework to develop new DMIs. In order to merge the technologies behind HTML 5, unleashing the development of DMIs for the web, this paper presents the Mosaicode, a Visual Programming Environment that can help novices and experts programmers in implementing and prototyping applications on the field of Digital Arts – better detailed in Section 3.

Several programming tools and languages already support the development of DMIs, including a technological apparatus to help programmers, non-programmers and lay-programmers to perform this task. A non-exhaustive list of tools and languages to develop new DMIs is presented in Section 2.

Different from other related tools, the Mosaicode is an application to generate code and complete applications using the Visual Programming paradigm to the Digital Arts domain.

This tool can be complemented by extensions, allowing one to add resources that define a Visual Programming Language (VPL) for developing applications for new domains. Among all extensions of Mosaicode, there is an extension to develop Web Art applications that supports the creation of DMIs. This extension has several blocks of code to access physical and logical inputs, audio sources and effects, MIDI devices, HTML elements and more. The development of DMIs using Mosaicode is presented in Section 4, including examples of DMIs developed in the environment.

The Mosaicode has already being used as a support tool to the course "Introduction to Computer Music" on the Computer Science Department at the Federal University of São João del-Rei – UFSJ. The students developed DMIs in an easy way, just by dragging and connecting blocks, focusing only on the concept of DMIs development. It was really interesting to use the Web Art extension to this purpose. Mosaicode also was used to create instruments to a live performance with the audience participation developed in our research group. This experiences and discussion about the use of Mosaicode is presented in Section 5. At the end, Section 6 presents some final considerations.

2 Related works

Currently there are several musical programming languages and tools, with different aspects and paradigms, to help the creation of new DMIs.

Pure Data[1] (a.k.a. Pd) is visual programming environment developed to create real time sound and music projects. This open source tool was developed by Miller Puckette in the 90's [1]. Although its main focus on audio manipulation, Pd enables to work with data from different sources that can be treated in an interconnected way. As a consequence, it facilitates the coupling of audio, video and MIDI applications, among others, that the tool supports.

Max/MSP [2] is, like Pd, a visual programming environment for image, sound and video processing [2], commonly used by artists, performers and composers to create their applications. Different from Pd, Max is a proprietary software.

[1] Available on http://puredata.info.

[2] Project Website: https://cycling74.com/products/max.

EyesWeb[3] is another visual development environment. This is specifically an open-source programming tool for image, sound and video, with an emphasis on gesture analysis and body movement [3]. A major advantage of it is the number of input devices supported, such as cameras for image capture, video game accessories like Kinect and Wii controls and multi-channel audio.

ISADORA[4] is an interactive tool for media (sound and video) that is widely dedicated to artists and performers looking for the creation of pieces and performances by iterating image and sound, allowing real-time processing and customization [4]. This environment has advantages such as appearance and friendly interface, along with the practicality of visual programming; in addition, it is proprietary software.

Processing[5] is an open-source textual programming language for the Digital Art domain developed by the MIT Media Lab [5]. This language is used in a didactic way for teaching programming to facilitate, captivate and bring up new students to the area of software development.

CodeCircle[6] is a software that provides a real-time, collaborative and social coding by means a web prototype environment for musically expressive instrument development. The user can implement applications using a specific code, including HTML, CSS and JavaScript by the web interface that consists of the code editor and the resulting web. For generate sound in the browser and to implement interactive machine learning, this software uses the MaxiLib and RapidLib libraries, respectively. Expressive interactions are designed using "programming by demonstration" [6].

JythonMusic[7] supports computer-assisted composition by a free and open source environment based on Python for interactive musical experiences and application development. The use of the Jython programming language enable to work with Processing, Max/MSP, PureData and other environments/languages, and also giving access to Java API and Java based libraries. It's allowed to the user to interact with external devices such as MIDI, manipulate images and also create graphical interfaces [7].

FAUST[8] is a functional programming language for sound synthesis and audio processing. A code developed in FAUST can be translated to a wide range of non-domain specific languages such as C++, C, Java, JavaScript, LLVM bit code, and WebAssembly[8]. There is a web platform called "Faust Playground" designed to enable children to learn basic audio programming, providing graphics resources to create DMIs.

Certainly, all these related tools have some difference with the tool presented in this paper and all of them are more mature and solid tools to create audio and music applications. Some of them are also visual, some are code generators. The most important to say about all these tools is that they all inspired our team to develop our programming environment and there are much more to learn about them to evolve our tool.

[3]Project Website: `http://www.infomus.org/`.
[4]Project Website: `https://troikatronix.com/`.
[5]Project Website: `https://processing.org/`.
[6]Project Website: `https://codecircle.gold.ac.uk/`.
[7]Project Website: `http://jythonmusic.org`
[8]Available on `https://faust.grame.fr/`

3 Mosaicode

Mosaicode [9] is a visual programming environment focused on the development of applications in the specific domains of Digital Arts. This programming environment, depicted in Figure 1, is a Desktop application developed in Python that provides elements to create applications for Digital Art domain involving Computer Science topics like Artificial Intelligence, Virtual Reality, Computer Graphics, Computer Vision and Computer Music.

This environment is an open-source code generation tool that aims to unleash development of applications and fast prototyping. Thus, it can be quick and simple for those who do not have the programming skill to create an application and for expert programmer to fast prototype, to fast change code – can quickly test new code settings – and also to change/optimize the application directly by the generated code. Based on this, the Mosaicode can be used to a fast start a project generating the code of a working prototype. The simplicity of the environment is based on the Visual Programming paradigm, used by several other related tools and very common on Digital Arts domain. An application in the Mosaicode is created dragging, dropping and connecting blocks.

A **Block** is the most basic unit in the environment and it is responsible to perform minimal, atomic and specific activity within a domain. A Block can have different behaviours and it can be set up by the Block's properties. Besides, a **Property** can be set up statically or dynamically. A static property is a parameter that influences the Block's execution and that can be modified in programming time, however, it is constant during application running. Figure 1.C presents the side bar to set up a Block's static properties in the environment.

A dynamic property uses a input **Port** of the Block to change the Block's settings and it is done at run time. A Block can also have output ports, to send values to other blocks. Thus, using these typed ports and connections, is possible to exchange values between blocks. A set of these blocks connected together is called **Diagram**.

A Diagram, like presented in Figure 1.D, is used to generate an application code based on a **Code Template**, merging code snippets and creating the final application.

By default, when there is a data stream on the input ports, these values override the static properties associated with each input port. Thus, when connecting a block, which outputs information from a sensor, to the input of a block that performs an arithmetic operation, for instance, the arithmetic block will no longer use the value assigned in the static property, it will use the sensor values to perform the operation.

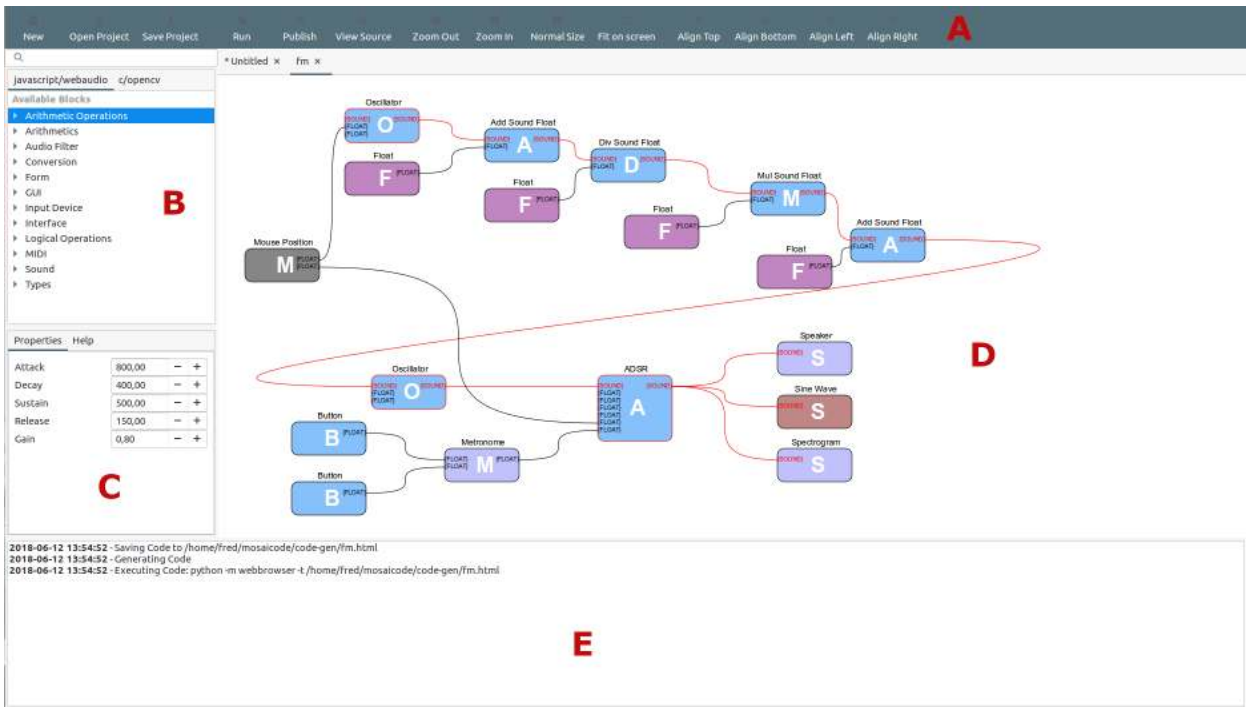

Figure 1: It shows the graphical interface of the Mosaicode.

When implementing a block diagram in Mosaicode, users do not have to worry about remind programming languages commands and syntax, they just need to be aware of the specific domain to know which blocks must be used and how to connect them to generate the desired application. Once a user acquires knowledge about the domain, it becomes easier to open the source code to study it and to learn how the application is implemented using a particular programming language.

3.1 Extending the environment

A set of blocks, ports, and code templates composes a Mosaicode **extension**. An extension can generate source code/applications for a particular programming language and a specific domain.

Currently, Mosaicode has some extensions to generate application code in C/C++ language for subjects like Sound Design, Digital Image Processing, Image Synthesis, GUI, Joystick Control and Computer Vision. Each one of these extension use an external library to support the application development, such as OpenCV for Computer Vision and Digital Image Processing [10], GTK for GUI creation, openGL to image synthesis and the PortAudio for Sound Design [11].

This environment also has an extension for Web Art development that generates HTML 5 + CSS + JavaScript application code. This extension is explained in the next section, discussing about how it can be used to create DMIs.

4 DMI development with Mosaicode

To present the DMI development with Mosaicode, we are adopting a common vision that a DMI can be splitted in three parts: the input, that captures gestures of the musician and involves the physical and virtual interactions of the user; the output, responsible for synthesizing the sound of the instrument and give other feedback to the user like visual and haptic; and the mapping, a strategy to interconnect the input with the output. Figure 2 presents this schematic, as will used in next Sections.

4.1 User Input

User input is intended to receive data streams referring to user interactions with the application. User inputs can be performed by physical devices, like sensors, or graphical interfaces components, like buttons, input text boxes and sliders. A computer mouse and keyboard are common devices that can be used to interact with applications and that can use GUI elements to intermediate this interaction. Beyond GUI elements, the JavaScript programming language easily provides resources to capture mouse events like click and movements and keyboard events like key press and release. Other input devices connected to the computer can also be accessed using HTML 5, specially when using smartphones and tablets. Using HTML5, the GPS position can be reached using the Geolocation, Joystick and game devices can be accessed with the GamePad API and MIDI devices can be accessed by the Web MIDI API (physical or virtual). Other sensors and physical devices like touch screen, gyroscope and accelerometers can also be accessed and used as user input using the JavaScript language. Javascript also allows to access the camera and microphone using the WebRTC API in real-time [12, 13].

GUIs designed to provide communication between users and applications (user input) are composed of HTML 5 elements. These elements can be: text fields, buttons, slider controls, number field, radio, check and others.

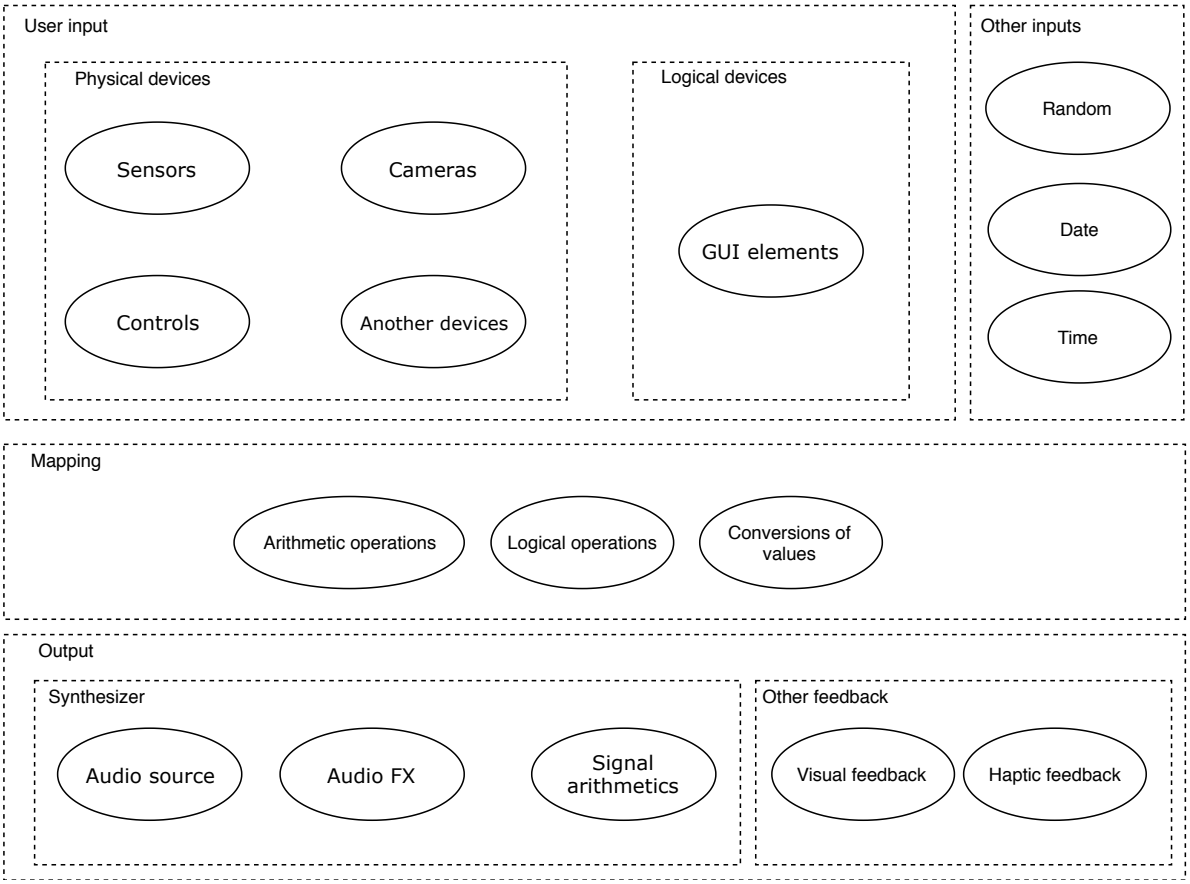

Figure 2: Basic structure of DMI.

GUIs are interesting to control DMIs for their accessibility, not requiring the purchase of other devices to play them.

Table 1: List of Mosaicode's input blocks to generate Web Audio applications in HTML 5/JavaScript language.

Categories	Blocks
Form	Button, Check, Number, Range, Select, Text.
Input Device	Date, Device Orientation, Hour, Keyboard, Microphone, Mouse Click, Mouse Position, Orientation Change, Up Keyboard.
Interface	Increment, Metronome, Random.
MIDI	MIDI in
Sound	Microphone

In our Web Art extension, we have joined the APIs resources and made them into Mosaicode blocks with the same functionality, enabling many ways for the application to receive data by input user. This extension blocks were organized by categories to make it easier to find the desired features for developing applications, as presented in Table 1

4.2 Other Inputs

There are other sensors and values, available in devices, that can be used as input in DMI development, but that are not controlled by user actions. These devices are the computer clock, providing date and time, and random values, for instance. This input can be used to add some stochastic parameters to DMIs, schedule events, delay events and create sequencers. All these functionalities are native in JavaScript programming and also became blocks to our environment.

We also implemented Numbers and other constants Blocks to use it as value input in diagrams. It is very useful to set up fixed values and normally used to set up initial values to run the application.

4.3 Synthesizer

The synthesizer is the voice of the DMI. Normally, a synthesizer can implement or be inspired by one or more classic algorithms of audio synthesis like AM, FM, PM, additive, subtractive, physical modelling, and others.

The Web Audio API provides several elements to create a synthesizer like Oscillators, Gain, Filters, Audio Spacialization, and also audio FX like delay, reverb, chorus, phaser, flanger, distortion, filters and others. Thus, to implement a synthesizer with this API is really simple and depends on computer music knowledge of how to create synths. All these elements were implemented as Blocks in the Mosaicode environment.

Some audio operations that are not available in Web Audio API but are important to create synthesizers were implemented using the Web Audio ScriptProcessorNode. Arithmetic operations to audio signals, White

Noise and ADSR envelope are examples of resources implemented to Mosaicode as Blocks using this feature of the Web Audio API. A not complete list of the Blocks to create a sound synthesizer is presented in Table 2.

Table 2: List of Mosaicode's blocks to generate web audio applications in HTML 5/JavaScript language.

Categories	Blocks
Audio Filter	Allpass, Bandpass, Highpass, Highshelf, Lowpass, Lowshelf, Notch, Peaking.
Sound	Add Sound, Add Sound Float, ADSR, Channel Merge, Delay, Distortion, Divide Sound, Gain, HRTF, Multiply Sound, Multiply Sound Float, Oscillator, Playback, Speaker, Subtract Sound, Subtract Sound Float, White Noise.

4.4 Other Feedback

A DMI can also use another outputs to help the user to understand its behavior. Our first explored output is a visual feedback, the resource to print a value on the web page; maybe it is the most used Block to debug code. To give visual feedback to user and to create a nice design it is also possible to change the background color of the page and other page elements, like the page title.

Some more interesting visual feedback were created using the HTML 5 Canvas element to draw representations of audio signals. There are Blocks to show a frequency bar chart, waveform and audio spectrum. These blocks also use the WebGL API, which supports rendering of 2D/3D graphics in real-time HTML Canvas elements, enabling analysis of audio signals at runtime [14].

Another possible output is to create and MIDI virtual device to output values from the web application to a local synthesizer, logical or physical, using the WebMIDI API.

We can use another two interesting resources to create feedback: the webcam flash light, accessible by WebRTC and the vibracall, using the Vibration API. Using the flash provides visual feedback such as graphics, background color and page title, and the vibrating alert provides tactile feedback.

All these possibilities are implemented in our extension and available in Blocks, like present in Table 3.

Table 3: List of Mosaicode's blocks to give feedback to user.

Categories	Blocks
HTML	Title, background color, print
Canvas	Frequency Bar, Print, Sine Wave, Spectogram
MIDI	MIDI Output
Mobile Output	Flash, Vibracall

4.5 Mapping

Although mapping does not uses to be a resource that can became a block , during the implementation of syntheses and audio effects, sometimes it became necessary to map values by adjusting them to vary in a certain range. For example, the position of a mouse on the screen, ranging from 0 to 1024, can be mapped to a gain, ranging from 0 to 1. Thus, it became necessary to have some blocks of mathematical and other operations to be used to convert values from the input to the output.

We created a set of Blocks to perform operations between float numbers, logical operations and to decrement and increment values. There are also a set of conversion blocks provided in Mosaicode to strip MIDI values and to convert MIDI notes to frequency, to convert from float to RGB, RGB to Float, Float to Boolean and Float to Char.

These elements are delegated by the mapping unit as parameters of DMI synthesis. In this way, joystick buttons, hand mapping with cameras and accelerometers and cell phone gyroscopes can be used as inputs to the synthesizing, controlling elements such as note duration and frequency, filter frequency, general gain and noise gain. Table 4 presents some Blocks that can be used to mapping values.

Table 4: List of Mosaicode's mapping blocks to generate web audio applications in HTML 5/JavaScript language.

Categories	Blocks
Logic	Equals To, Greater Than, Greater Than Or Equals, Less Than, Less Than Or Equals, Not Equals To.
Arithmetics	Add Float, Divide Float, Max Float, Modulos, Multiply Float, Subtract Float.
Conversion	Bool To Float, Char To Float, RGB.
MIDI	MIDI To Frequency, Strip MIDI.

4.6 Examples

An example of a FM synthesizer developed in Mosaicode is presented in Figure 3. In this example, the carrier oscillator (sine wave format) has as initial frequency value 440 Hz and the frequeency of the modulator oscillator in 4000 Hz.

The values generated by the oscillators range from -1 to 1. To change this value to a range of 220 to 2220, we added the value 1 to the oscillator output, making it vary in the interval from 0 to 2. Then we multiplied it by 1000 doing it vary from 0 to 2000 and finally we added the value 220 to vary the value from 220 to 2220. Thus, we have a periodic frequency change in the range of 220 Hz to 2220 Hz occurring 4000 times per second, which determines the carrier oscillator frequency over that time. The carrier oscillator output was directed to the Sine Wave block, to be drawn in the waveform as shown on Figure 4, and also directed to the Speaker block.

Another example is presented in Figure 5a. This DMI uses a ADSR envelope receiving a white noise as input and being triggered by a button. Finally, the envelope output is connected to a Speaker and a frequency bar.

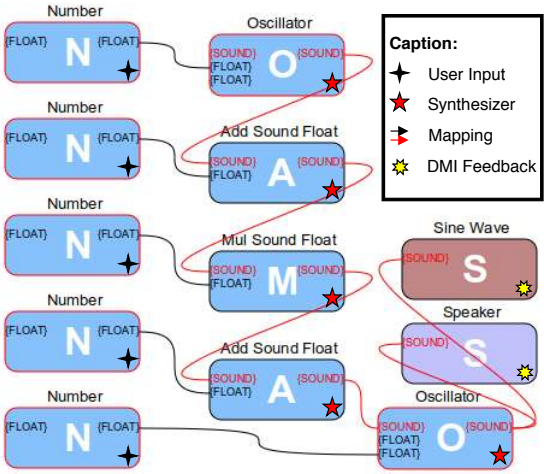

Figure 3: FM synthesis using the Mosaicode-javascript-webaudio extension.

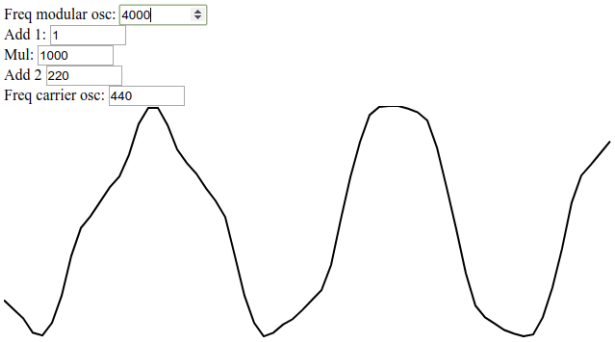

Figure 4: Visualization of waveform generated by the FM Synthesis on the example of Figure 3.

So, when running the application, we have a button on the web interface (5b), the frequency bar chart, which varies when playing the instrument, and we can hear the instrument sound.

These examples use numeric fields to control synthesizer parameters and a button to dispatch the ADSR envelope. It is possible to replace this fields with any other user input block just changing it in the diagram or dispatch the white noise's envelope with a metronome, for instance.

5 Discussion

We used Mosaicode as a support tool to implement and prototype DMIs in a Computer Science course called "Introduction to Computer Music" [15] that had the main audience undergrad students on Computer Science field. In these classes we noticed how important is to have a development environment that enables rapid prototyping and creation, also to initiate instrument designs. Often, a synthesizer is born from experimentation with signal arithmetic, testing and experimenting with settings and parameters. The same can occur with the choice of interfaces or mappings.

At least during prototyping or in the classroom, constructing a DMI resulted in ephemeral codes, an experimentation that can be reused, but that also can be simply discarded due to this ephemerality. Throwing work away may not be a problem when we know that doing it again can be simple and even fun.

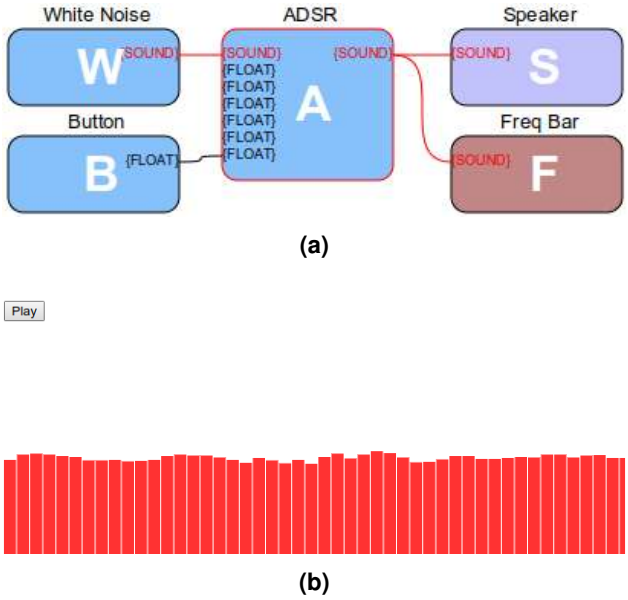

Figure 5: Diagram with White Noise controlled by an ADSR envelope, with outputs on a Speaker and a Freq Bar, shown in (b).

We also used it in the context of DMI development, with audience participation in a multimedia performance called Chaos das 5. In this performance, the audience could access web instruments and take part of the soundscape of the performance. The development of these instruments by our research group was made improvising and playing sounds, sometimes totally free, based on experimentation and trials. We used pair programming including students with different levels of knowledge in Computer Music and synthesis algorithms. However, after a few meetings, all students could already create sounds and develop DMIs, even without a formal course in Music Computing.

6 Conclusion

This paper presented the Mosaicode, a visual programming environment for the domain of Digital Arts that here was explored to create DMIs using a Web Art extension. This extension, based on several APIs from the HTML5, can be used to create really interesting and multi-platform DMIs that would be hard to code using JavaScript directly. The development of this extension on Mosaicode can offer the power of HTML 5 by the means of a visual programming environment.

It can bring several advantages, such as rapid prototyping, as well as practicality, easy experimentation, trials and combination of blocks, generating completely different applications. It is also possible to use it as a support tool to teach lay programmers and artists to develop their applications without requiring learning a textual programming language. For this, Mosaicode offers to the user a

wide range of possibilities and different combinations of Blocks.

For future work, we intend to maintain the developed extension and to review the set of blocks, adding new features whenever possible to make the extension more complete. In addition, an extension for MIDI controls is being developed, with various input types available to different media, using the C language. Initially, the MIDI category of audio synthesis extensions is scarce and limited, we have the intention of merging the extensions of the same programming language in order to rapidly expand the possibilities of development, allowing the user to integrate different domains in a simple way.

When creating DMIs it was important to think about communication interfaces that offer a good musical expression. For this, we implemented blocks that allow the use of external devices to control the synthesizers with a certain degree of complexity, trying to reach a more expressive DMI [16].

7 Acknowledgments

Authors would like to thanks to all Arts Lab in Interfaces, Computers, and Everything Else (ALICE) members that made this research and development possible. The authors would like also to thank the support of the funding agencies CNPq, FAPEMIG, and also the PROAE/PROPE/UFSJ for the institutional support.

References

[1] Miller S Puckette et al. Pure data. In *ICMC*, 1997.

[2] Matthew Wright, Richard Dudas, Sami Khoury, Raymond Wang, and David Zicarelli. Supporting the sound description interchange format in the max/msp environment. In *ICMC*, 1999.

[3] Antonio Camurri, Shuji Hashimoto, Matteo Ricchetti, Andrea Ricci, Kenji Suzuki, Riccardo Trocca, and Gualtiero Volpe. Eyesweb: Toward gesture and affect recognition in interactive dance and music systems. *Computer Music Journal*, 24(1):57–69, 2000.

[4] TroikaTronix. Isadora. `https://troikatronix.com/`. Acessado em: 2018-07-29.

[5] Casey Reas and Ben Fry. *Processing: a programming handbook for visual designers and artists*. Mit Press, 2007.

[6] Michael Zbyszynski, Mick Grierson, Matthew Yee-King, et al. Rapid prototyping of new instruments with codecircle. In *Proceedings of the international conference on new interfaces for musical expression*. NIME, 2017.

[7] Bill Manaris, Blake Stevens, and Andrew R Brown. Jythonmusic: An environment for teaching algorithmic music composition, dynamic coding and musical performativity. *Journal of Music, Technology & Education*, 9(1):33–56, 2016.

[8] Yann Orlarey, Dominique Fober, and Stéphane Letz. Faust: an efficient functional approach to dsp programming. *New Computational Paradigms for Computer Music*, 290:14, 2009.

[9] Flávio Luiz Schiavoni, Luan Luiz Gonçalves, and José Mauro da Silva Sandy. Mosaicode and the visual programming of web application for music and multimedia. *Revista Música Hodie*, 18(1):132–146.

[10] Gary Bradski and Adrian Kaehler. Opencv. *Dr. Dobb's journal of software tools*, 3, 2000.

[11] Ross Bencina and Phil Burk. Portaudio-an open source cross platform audio api. In *ICMC*, 2001.

[12] Hongchan Choi and Jonathan Berger. Waax: Web audio api extension. In *NIME*, pages 499–502, 2013.

[13] David B Ramsay and Joseph A Paradiso. Grouploop: a collaborative, network-enabled audio feedback instrument. In *NIME*, pages 251–254, 2015.

[14] John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar Ruiz. Interactive visualization of volumetric data with webgl in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology*, pages 137–146. ACM, 2011.

[15] Flávio Luiz Schiavoni Schiavoni, Thiago Thadeu Souto Cardoso, André Lucas Nascimento Gomes, Frederico Ribeiro Resende, and José Mauro da Silva Sandy. Utilização do ambiente mosaicode como ferramenta de apoio para o ensino de computação musical. In *Proceedings of 8th workshop on ubiquitous music (UbiMus)*, 2018.

[16] Christopher Dobrian and Daniel Koppelman. The'e'in nime: musical expression with new computer interfaces. In *Proceedings of the 2006 conference on New interfaces for musical expression*, pages 277–282. IRCAM—Centre Pompidou, 2006.