Mosaicode and the visual programming of Web Application for Music and Multimedia

Flávio Luiz Schiavoni [Universidade Federal de São João del-Rei São João del-Rei, Minas Gerais. Brazil)
fls@ufsj.edu.br
Luan Luiz Gonçalves [Universidade Federal de São João del-Rei São João del-Rei, Minas Gerais. Brazil)
luanlg.cco@gmail.com
José Mauro da Silva Sandy [Universidade Federal de São João del-Rei São João del-Rei, Minas Gerais. Brazil)
jmsandy@gmail.com

Abstract: The development of audio application demands a high knowledge about this application domain, traditional programming logic and programming languages. It is possible to use a Visual Programming Language to ease the application development, including experimentations and creative exploration of the language. In this paper we present a Visual Programming Environment to create Web Audio applications, called Mosaicode. Different from other audio creation platforms that use a visual approach, our environment is a source code generator based on code snippets to create complete applications.
Keywords: Webaudio, Mosaicode, Visual Programming Language, Specific Domain (Programming) Languages, Graphical Programming Environment.

Mosaicode e a programação visual de aplicações Web para música e Multimídia

Resumo: O desenvolvimento de aplicações de áudio demanda um grande conhecimento neste domínio de aplicação, lógica de programação tradicional e linguagens de programação. Para simplificar o desenvolvimento de aplicações, permitir experimentações e a exploração criativa da programação é possível utilizar linguagens de programação visual. Neste artigo apresentamos um ambiente de programação visual para a criação de aplicações web chamado Mosaicode. Diferente de outras plataformas gráficas para criação de áudio, nosso ambiente é um gerador de código fonte baseado na reutilização de trechos de código para a criação de aplicações completas.

1. Introduction

The emergence of Web Audio API brought to the web browser the capability to create and process real time audio. These tools led several developers and artists to explore and use the possibility of creating sounds and music in a portable format that can reach different operating systems, architectures and devices, from desktops to mobiles. During the rise of the Internet, the web was used only to distribute music and the sound interaction was possible based on recorded audio in digital formats. Nowadays, it is possible to reach a new level of interaction and use the browser as a powerful tool for sound making.

Web technologies provide powerful tools to develop cross-device applications including a high level sample-accurate sound engine and a sophisticated layout system for visual feedback (ROBERTS et al., 2014; WYSE; SUBRAMANIAN, 2013). The layout system includes HTML5 new features like the canvas element and the ability to create vectorial drawing like SVG. The use of vectorial drawing allows to dispatch events on every each part of the drawing, giving a better level of interaction and a great possibility to art creation. Using handheld devices, HTML5 also enables to incorporate available sensors like accelerometers, multitouch screens, gyroscopes and others (ROBERTS; WAKEFIELD; WRIGHT, 2013; ROBERTS et al., 2015; TAYLOR; ALLISON, 2015; LETZ et al., 2015). It is also possible to incorporate legacy music interfaces with the Web MIDI API (WILSON; KALLIOKOSKI, 2015).

Digital artists often have difficulty starting their research with sound due to lack of knowledge of algorithms and traditional programming logic. To simplify the development of applications, it is possible to use Visual Programming Languages (VPLs). VPL is a class

 Recebido em: 27/11/2017 - Aprovado em: 05/03/2018

of programming languages that allow the programmer to develop software using a two-dimensional notation and interact with the code by the means of a graphical representation instead of editing an one-dimensional stream of characters, having to memorize commands and textual syntaxes (HAEBERLI, 1988). This may allow non-programmers or novice programmers to develop complete applications (GOMES; HENRIQUES; MENDES, 2008) since VPLs can bring ease to system development. In addition, code abstraction by means of a diagram can bring practicality in changing the code making them quite suitable for rapid prototyping (HILS, 1992). This is a well-known approach in the arts due to the common use of tools like Pure Data, Max/MSP or Eyesweb. On Section 2, related works will be presented and these tools will be discussed in more detail.

Another way to simplify the development of computational systems is by using domain-specific languages (DSL) (GRONBACK, 2009). DSLs have the knowledge of the domain embedded in their structure and an abstraction level higher than general-purpose programming languages. It makes the process of developing systems within your domain easier and more efficient because DSLs require more knowledge about the domain than programming skills (MERNIK; HEERING; SLOANE, 2005). For this reason, the potential advantages of DSLs include reduced maintenance costs through re-use of built-in features and increased portability, reliability, optimization and testability (DEURSEN; KLINT, 2002). Domain Specific Languages are also common in art field since this approach was used to develop languages like Csound (VERCOE, 1981), Supercollider (MCCARTNEY, 2002) or RTCMix (GARTON; TOPPER, 1997).

In this paper we present Mosaicode (SCHIAVONI; GONCALVES, 2017a), a visual programming environment that can be used to develop systems in the specific domain of digital art combining the simplicity of visual programming with code reuse of DSLs. We propose in this work, the construction of a set of Blocks for audio application based on JavaScript programming language and the Web Audio API. This work is organized as follows: Section 2 presents related works and the fundamentals to understand this work including the concepts of Visual Programming Languages, Specific Domain Languages for Music, Javascript and Web Audio and Code Generation; Section 3 presents the Mosaicode application, how it works and the basics of programming and code generation in this visual programming environment; Section 4 discusses the development of Web Audio applications with Mosaicode focusing on the features provided to programmers; Section 5 presents the initial results and some classic synthesis algorithms implemented in this environment. Finally, Section 6 presents the Conclusion and the Future Works.

2. Related Works

When focusing on the creation and code generation of web applications in music and arts, there are four categories of tools: Visual Programming Languages, Specific Domain (Programming) Languages for Music creation, Javascript and web audio frameworks and Code Generation tools focused on Art creation. These groups intersect and some tools / languages can be in more than one category.

2.1 Visual Programming Languages

From the point of view of VPLs for audio and music programming, the related works start with Pure Data, Max/MSP and Eyesweb. Pure Data is a Visual Programming Environment for Sound and Music that plays host to GEM environment (DANKS, 1997) to 3D graphic processing (PUCKETTE, 1996). Max/MSP is a music and video real time gra-

phical programming environment that predates Pure Data (PUCKETTE; ZICARELLI, 1990). Pure Data and Max/MSP share the same paradigm being both created by the same author, Miller Puckette. Eyesweb is a project focused on real time analysis of body movement and gesture (CAMURRI et al., 2000).

The idea of using a VPL to generate web based music application was already explored by EarSketch (MAHADEVAN; FREEMAN; MAGERKO, 2016) but in this case, a web based environment was used to create the applications based on the Blockly programming environment (TROWER; GRAY, 2015). Earsketch is an online platform to make music based on loops and remixing and normally the code is developed in Python or Javascript instead of using the VPL approach.

2.2 Specific Domain Languages (SDL) for Music

Computer programming languages started with the pioneering system called Music N languages (MATHEWS, 1963) using text to code music. Max (PUCKETTE; ZICARELLI, 1990) introduced the visual metaphor to code music. Several languages can be considered evolutions of the Music N language and concepts (POPE, 1993) like CMix (LANSKY, 1990), Cmusic (MOORE, 1990) and Csound (VERCOE, 1991) and, recently, Supercollider (MCCARTNEY, 2002).

CMix evolved to RTcmix adding the Real Time (RT) to the language name and the capability to be a real-time software "language" for doing digital sound synthesis and signal processing (GARTON; TOPPER, 1997). Developed in C/C ++ language, RTcmix is open source and free of charge. RTcmix is available standalone through command line use, for use compiled in other programming environments or for use in other applications such as iPhone, IPad and IPod Touch.

Csound is a sound and music computing system that has a strong tradition as a tool to compose electroacoustic pieces. It can be used by composers and musicians for any kind of music done with the help of the computer. In its conception, Csound was used in a non-interactive score driven context, but currently it is also used in real-time contexts. In addition to being able to be used on different platforms, including all major operating systems or Android and iOS systems, it can be called through other programming languages, such as C/C++, Lua, Java, Python, etc.. The main feature of Csound is the compatibility between versions, so a source file created in the first versions can still run on the most current versions.

Using the concepts of object-oriented programming, SuperCollider is a platform for audio synthesis and algorithmic composition used by musicians and artists. It is open source and free software, available on Windows, Mac and Linux operating systems. SuperCollider has three main components (MCCARTNEY, 2002):

- scsynth, a real-time audio server that is the center of the platform, with more than 400 generating units, known as "UGens", that allow various combinations of techniques for sound manipulation. In addition, users can create their own "UGens";
- sclang, is an interpreted programming language, which although focused on sound, is not limited to a single domain and supports the creation of new methods;
- scide, is an editor for sclang with an integrated help system.

2.3 Javascript and web audio

JavaScript and Web Audio have triggered the development of several libraries, like Flocking.js (CLARK; TINDALE, 2014), gibber (ROBERTS et al., 2014) and WebCsound

(LAZZARINI et al., 2014). Flocking was inspired by Supercollider and uses JSON to specify and declare synthesizers. For this reason, it is very easy to combine and create new instruments and create music. Gibber is a coding environment (IDE) that runs on the browser and comes with built-in code that users employ to create beats and melodies. Also inspired by the concept of ugens (unit generators), Gibber includes Gibberish, a JavaScript synthesis and scheduling library. WebCsound brought Csound to the browser client-side, without the need of any plugin installation.

2.4 Code Generation

Regarding code generators, Processing (REAS; FRY, 2006) is a DSL textual programming language and an IDE that generates code to for Visual Art applications. A sister project called Processing.js (RESIG; FRY; REAS, 2008) was designed to write visualizations, images, and interactive content. There is also p5.js a JavaScript library based on the core principles of Processing. Another interesting code generator is Faust (ORLAREY; FOBER; LETZ, 2009), a purely functional programming language that features libraries and APIs for audio signal processing and that is able to generate code in several target languages.

Code Generation has both advantages and limitations when compared to standard audio and musical data generation. It is an interesting form to initiate novices in computer programming with a abstraction layer between the programmed code and the generated code. Thus, it is possible to understand the computer abstraction and how data is transformed into other data or code into other code. The generated code can be improved and does not depend on the programming environment used to develop the initial code. On the other hand, the programming environment demands setting up the machine for correct execution of the code and this is not trivial.

3 About the Mosaicode

Mosaicode (SCHIAVONI; GONÇALVES, 2017a) is a Graphical programming environment to create Visual Programming Languages, normally based on a specific domain, like a DSL. Initially developed as a Computer Vision Programming Environment to generate C code to OpenCV library, Mosaicode has been expanded to other programming languages and domains such as Python and Image Processing. The visual programming environment uses the Block metaphor to create computer programs. Blocks are organized into groups in our environment GUI, presented on Figure 1. A block is the minimal source code part and brings the abstraction of a functionality of our desired domain. Blocks have static properties, used to set up their functionality, presented in Figure 2.

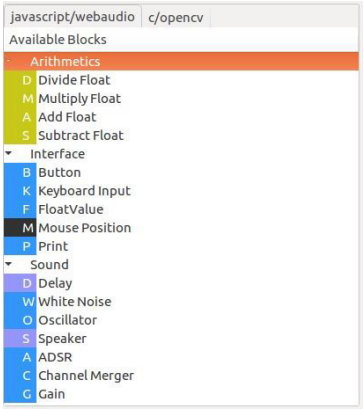

Figure 1: Part of Javascript/Web Audio Block groups.

Blocks also have dynamic properties whose values can be set up by other blocks. This capability to exchange information is represented by the blocks input/output ports. The information exchange by different blocks is made creating a connection between two or more ports. A block port has a defined type and a connection can be done using ports of the same type. A Collection of Blocks and Connections creates a Diagram, as presented in Figure 3.

Figure 2: Oscillator Block static properties.

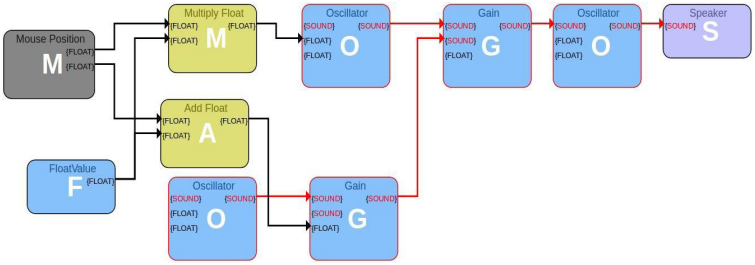

Figure 3: A Diagram with several Blocks interconnected.

In contrast with other visual programming languages, like Pure Data or Max/MSP, Mosaicode is not an interpreted environment but a code generator. Every single block and connection defines code fragments and add code Snippets of the application final code. The application code also depends on a code template that defines how the code snippets will be merged into the final application. Once a diagram is completed, it is possible to generate the application source code and run it. Figure 4 presents a generated source code.

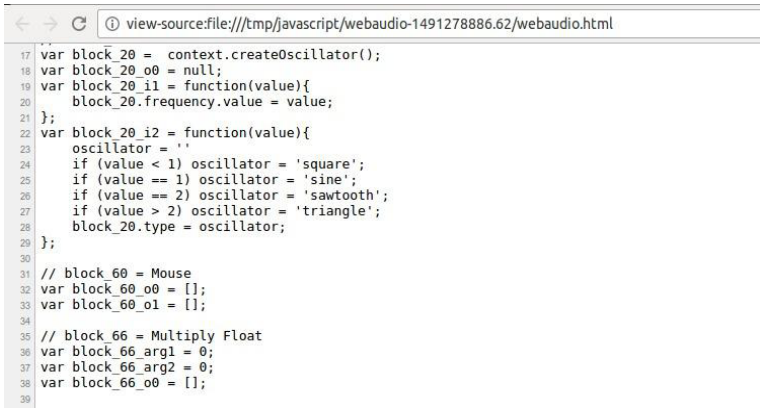

Figure 4: Example of Javascript generated Source Code.

The tool has an interface and a plugin manager that allows the creation of new components for the environment, like Blocks, Ports and Code Templates, so the tool can be extended, allowing the generation of source code to different programming languages and specific domains. The creation of new blocks for code generation consists of a Python class extending the BlockModel class. The BlockModel class offers three sets of properties that must be specialized for that the new block to behave properly. The first set, called Definition, consists of determining an identifier of the block, the language, the framework it will

compose and its source information. The second set is responsible for defining the appearance of the blocks when presented within the tool, properties such as help text, label, color, grouping and input and output ports can be specified. Finally, the third set, responsible for code block generation to access the properties and code snippets that can be customized for the new blocks.

It is worth mentioning that all generated blocks are under the control of a code template that defines the commands, code snippets, extensions and code that should be generated for each block. In the context of Mosaicode these new templates must be extended from the CodeTemplate class.

4. Developing Web applications with Mosaicode

The HTML5 emergence brought the possibility of real time audio creation on a very portable environment, the web browser. Web browsers are present in almost all contemporary operating systems and allow the integration of different media like text, image files, audio files and others. Thus, first we focused on the possibilities of HTML5 and import these possibilities into our Graphical Programming Environment.

We developed a Block for JavaScript/Web Audio code generation. We investigated which Blocks would be necessary to satisfy the Digital Arts requirements. The Blocks creation was based on other tools like Gibberish and Pure Data. Gibberish has a collection of Audio processing classes classified into the following categories: Oscillators, Effects, Filters, Synths, Maths and Misc. The Math group contains Add, Subtract, Multiply, Divide, Absolute Value, Square Root and Pow (ROBERTS; WAKEFIELD; WRIGHT, 2013). The Misc group contains Sampler objects (play/record), Envelope (ADSR, AD), Line Ramp and others. Later, authors added to this list a Drums category (ROBERTS et al., 2015) and GUI widgets like Sliders, Buttons, Sensors, knob, Piano and other GUI components.

Pure Data has an interesting object list organized into categories: General, Time, Math, MIDI and OSC, Misc, Audio Math, General Audio Manipulation, Audio Oscillators And Tables and Audio Filters. Our last investigation included an analysis of Web audio Native Nodes and the possibilities that our target language and API could offer. The Web Audio API offer audio resources implemented as interfaces (called audio nodes) and associated events, which are split up into nine categories of functionality (Table 1).

Categories	**Resources**
General audio graph definition	General containers and definitions that shape audio graphs in Web Audio API usage
Audio sources	Interfaces that define audio sources for use in the Web Audio API
Audio effects and filters	Interfaces for defining effects to audio sources
Audio destinations	Once the audio processing is done, these interfaces define where to output it;
Data analysis and visualization	Interfaces for extract time, frequency, and other data from audio
Splitting and merging audio channels	To split and merge audio channels
Audio spatialization	These interfaces allow to add audio spatialization panning effects to audio sources
Audio processing in JavaScript	Used to define custom audio nodes written in JavaScript or WebAssembly
Offline/background audio processing	It is possible to process/render an audio graph very quickly in the background, rendering it to an AudioBuffer rather than to the device's speakers

Table 1: Web Audio API resources split up into categories of functionality

Native Nodes

Our first set of blocks was done using the Web Audio Native Nodes. It included audio oscillators, audio filters, audio effects and general audio manipulation, like gain, speakers, channel merge and channel splitter. These Nodes have two configuration parameters: fixed parameters and a-rate audio parameters. Fixed parameters, like OscillatorNode. type, has fixed values like sine, square, sawtooth, triangle and custom. These parameters were directly mapped onto a block static property in Mosaicode. The other parameter types, a-rate Audio Parameter, like OscillatorNode.frequency, were mapped as input ports. This kind of parameter has also a value field (OscillatorNode.frequency.value) also mapped onto a static property. The AudioNode channel inputs were mapped to input ports and the channel outputs were mapped onto output ports. These ports can be connected, following the basic idea of Node connection from the Web Audio API (Figure 5).

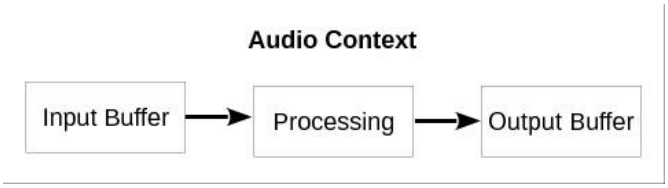

Figure 5: A typical workflow for Web Audio Nodes.

This AudioNode API supports playing audio and video files and saves data as audio or video in real time. This feature was explored to store the sound output and to play audio files.

Script Processor Node

The high-level Nodes provided by the Web Audio API do not include all sound processing features. We used a special node called Script Processor Node to enable the definition of complete audio systems entirely in JavaScript (ROBERTS et al., 2014). The Script Processor Node (Figure 6), allows for the development of new sound nodes integrated into the Web Audio API.

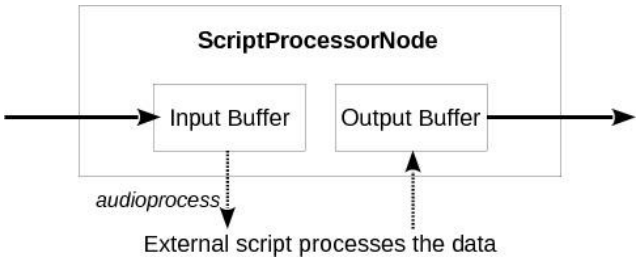

Figure 6: ScriptProcessorNode used to create Blocks like the WhiteNoise generator.

Blocks considered important to our environment and not present as Native Nodes were implemented using the Script Processor Node: ADSR envelopes, AD envelopes, White Noise Generators and signal math operations. None of these blocks needed static parameters.

GUI Input / Output

To allow interaction with sounds, we developed a set of HTML5 widgets to compose GUIs. User input includes a block to get mouse position and keyboard input, allowing

interaction with the computer default devices and with form elements from HTML, like button and sliders, and with HTML default content, like page title, page inner HTML and background color.

We also developed Misc objects to convert MIDI to frequency, 3 float values into a RGB Color, characters to floats and floats to characters. All these objects were implemented using basic HTML and javascript code without any other javascript library.

Input devices

We also explored the capability of interact with sensors and physical input devices available in mobile devices and notebooks. There are blocks that can be used to get device orientation using the internal gyroscope, triggering a signal when the orientation changes, i.e., when a device is in portrait and changes to landscape or from landscape to portrait. Another implementation passes the coordinates of the three axes, X (beta), Y (gamma) and Z (alpha) obtained from the device at each instant.

Using the Web Real-Time Communications (WebRTC) it is also possible to capture and to stream audio and/or video media using a microphone or a webcam. Managing the integration of legacy input devices, the WebMIDI API allows to use MIDI controllers in websites as input devices.

4.1 Creating port connections

Once we defined a set of Blocks, the next step on this development was to create connections between blocks. Audio input and output data could be easily connected since the Web Audio API uses this programming model to connect its Nodes. The creation of other connections between objects, like char or float, was our first challenge. Our former implementation with OpenCV and C could use C pointers to pass values from one object to another to create Blocks connections.

Since there are no pointers in Javascript, we had to look for other approaches to create block connections. The solution used was based on the Observer Design Pattern (Gamma, 1995) involving an array of callback functions to be called when an event occurs. This solution granted a responsive interface with a small code base. Thus, every connection is an observer registration to an observable event and when a GUI object action is performed, it transmits the result to a list of connected blocks that ensure the data flow.

5. Initial Results

As an initial result, we present the library mosaicode-javascript-webaudio, with resources offered by the Web Audio API and HTML 5. We also implemented resources for Graphical User Interfaces, MIDI devices on Web MIDI API and input devices. Each of these resources has been implemented as a block in Mosaicode and can be classified as input blocks, processing blocks, and output blocks (Table 2).

Input blocks	RGB, Button, Check, Color, Number, Range, Select, Text, Title, Date, Hour, Keyboard, Microphone, Audio Files, Mouse Click, Mouse Position, Metronome, Random, MIDI, Oscillator, Playback, White Noise, Float and Point.

Processing blocks	Add Float, Divide Float, Max Float, Multiply Float, Subtract Float, All-pass, Band-pass, High-pass, High-shelf, Low-pass, Low-shelf, Notch, Peaking, Char 2 Float, Increment, Midi 2 Freq, Strip MIDI, Div Sound, Add Sound, Add Sound Float, ADSR, Delay, Distortion, Div Sound, Div Sound Float, Gain, HRTF, Mul Sound, Mul Sound Float, Sub Sound, Sub Sound Float.
Output blocks	Freq Bar, Print, Sine Wave, Spectrogram, Background, Audio Files and Speaker.

Table 2: Present a schematic organization of our library.

Our Mosaicode library has the ability to create Web Audio applications for the development of computer music techniques. The following examples present the development of Web Audio applications using our library and applying standard computer music techniques such as audio synthesis and envelopes to create sounds.

Additive Synthesis

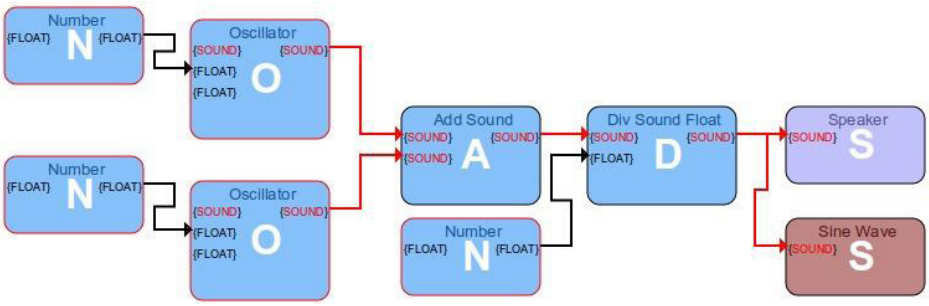

Figure 7: Example of Additive Synthesis using the mosaicode-javascript-webaudio library.

Figure 7 shows a Mosaicode diagram that implements additive synthesis. In this example we have the sum of two oscillators (Add Sound), one in the sine wave format and the other in the sawtooth waveform. Both oscillators receive the value of their frequency value through input field and the result of the sum of the oscillators was divided by 2 for avoid clipping, directing the result to the speaker and to the block that draws the waveform (Sine Wave). The execution of the additive synthesis example is shown in Figure 8, using the value 1760 Hz for the oscillator frequency in the sinusoidal waveform and the value 2200 Hz for the oscillator in the sawtooth waveform.

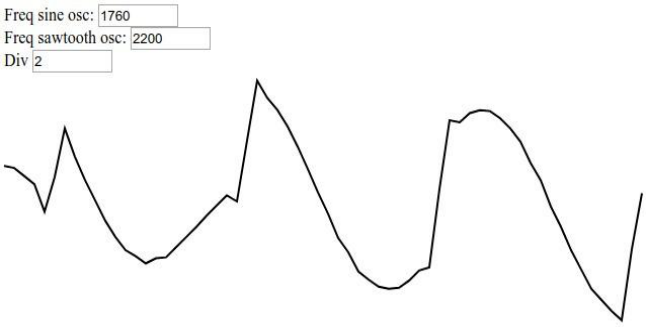

Figure 8: Visualization of a waveform generated by the additive synthesis on the example of Figure 7.

Subtractive Synthesis

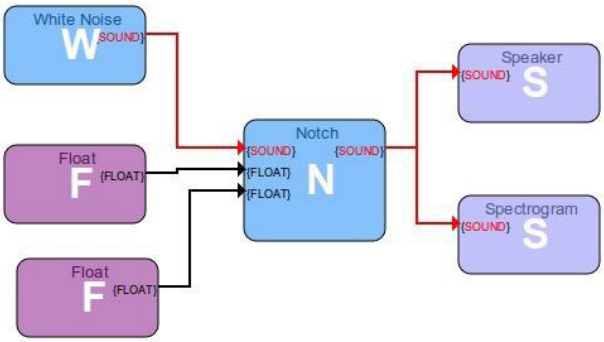

Figure 9: Example of subtractive synthesis using the mosaicode-javascript-webaudio library.

The example of Figure 9 shows subtractive synthesis implemented by applying the Notch filter on a White Noise generator. The filter cuts the frequencies according to the filter configuration - this cut can be seen in Figure 10 , represented by the dark line.

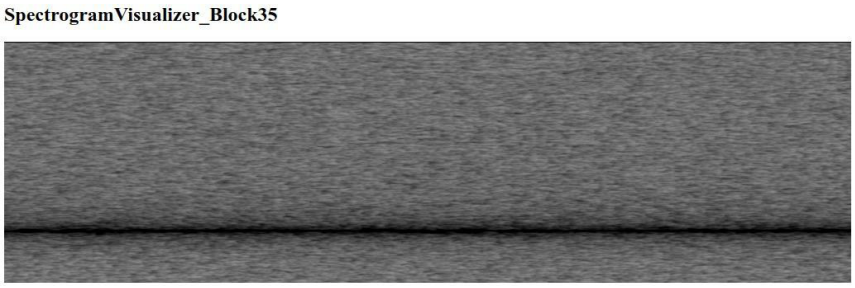

Figure 10: Visualization of waveform generated by the subtractive synthesis on the example of Figure 9.

Frequency Modulation Synthesis – FM synthesis

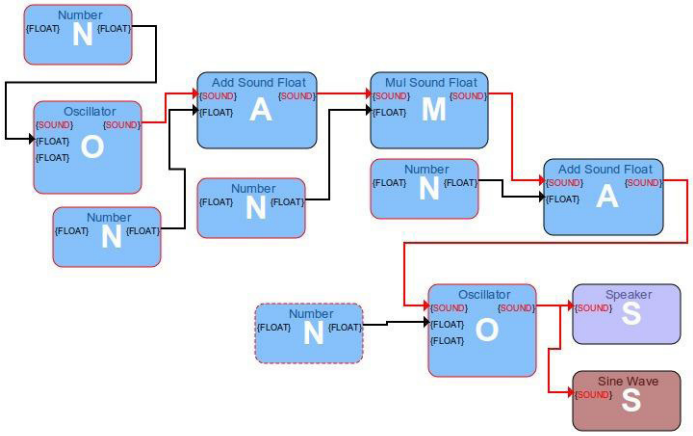

Figure 11: Example of FM synthesis using the mosaicode-javascript-webaudio library.

The example of Figure 11 presents an FM synthesis example that makes a periodic change in the frequency of an oscillator (carrier oscillator) using another oscillator (modulator oscillator). For this example, the carrier oscillator sounds in the sine wave format with initial input value the frequency 440 Hz and the modulator oscillator sounds in the square wave format with input value the frequency 4000 Hz. The oscillators generate values in the range of -1 to 1, but we want this value to be in the range of 220 to 2220. We add the value 1 to the output of the oscillator making it vary in the interval from 0 to 2, then multiply by value 1000 making it vary from 0 to 2000 and finally we add the value 220 making it vary 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 frequency of the carrier oscillator over that time. The output of the carrier oscillator is directed to the Sine Wave block to be drawn in the waveform as shown on Figure 12 and is also directed to the Speaker block.

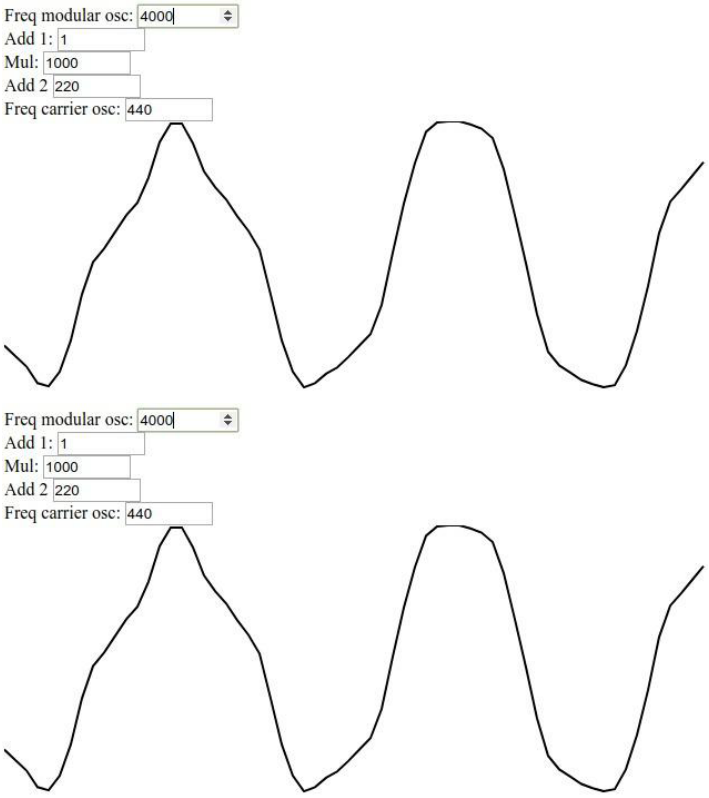

Figure 12: Visualization of waveform generated by the FM Synthesis on the example of Figure 11.

Amplitude Modulation Synthesis – AM Synthesis

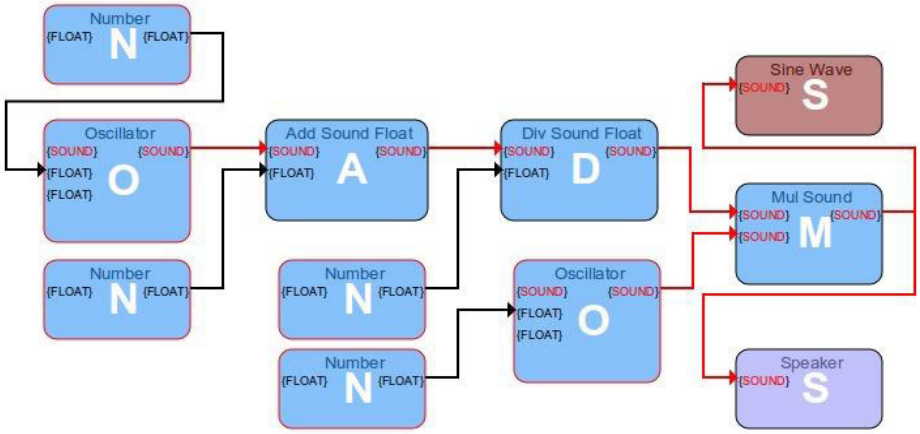

Figure 13: Example of AM synthesis using the mosaicode-javascript-webaudio library.

While FM Synthesis changes the frequency periodically, AM Synthesis changes the gain of a signal by using another signal. In the example of Figure 13, both signals are oscillators in the sine wave format, whose gain is being modulated by the "carrier", and the oscillator responsible for the "modulation" is called the modulator. The modulator oscillator signal sounds at frequency 440 Hz and is added by the value 1 and divided by the value 0.5, making a variation in the range of 0 to 0.4, which is multiplied by the carrier oscillator making it vary the gain 440 times per second in the gain interval defined by the modulator (Figure 14).

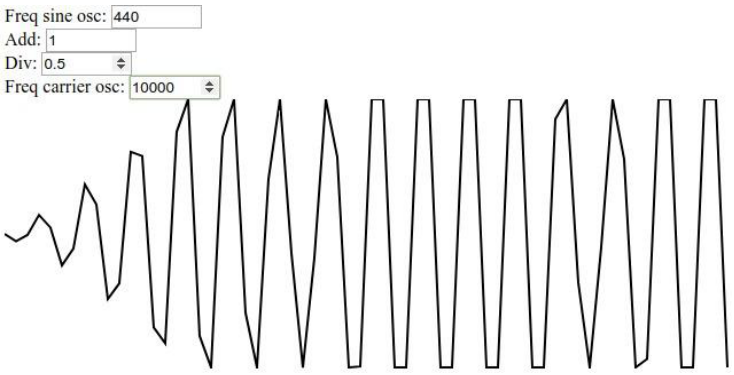

Figure 14: Visualization of waveform generated by the AM Synthesis on the example of Figure 13.

Instrument

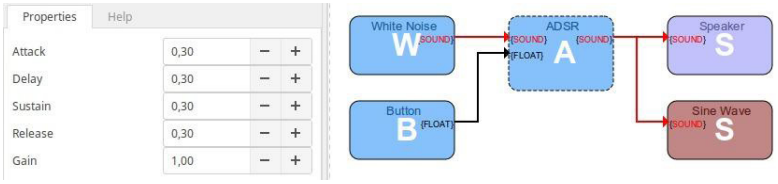

Figure 15: Example of creating a instrument using the mosaicode-javascript-webaudio library.

We built an instrument applying an ADSR Envelope over a white noise, triggered by pressing a button (Figure 15). The attack, decay, sustain, release and gain of the envelope is defined in the properties of the ADSR block, thus defining the instrument. On Figure 16, the sound was triggered, being modified by the envelope, having an attack, a decay, a sustain and a release.

Figure 16: Visualization of waveform generated by the instrument on the example of Figure 15.

Sequencer

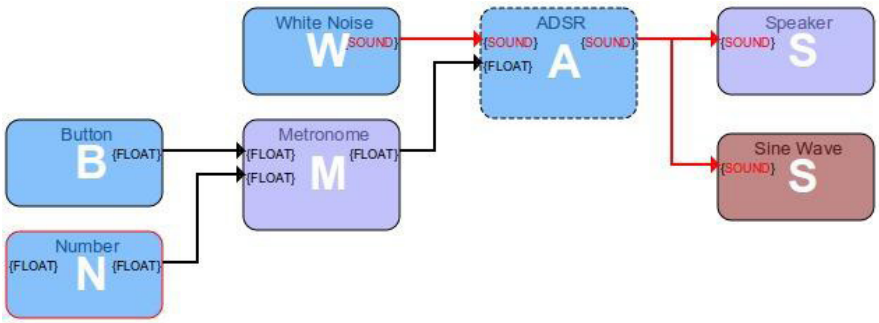

Figure 17: Example of creating a sequencer using the mosaicode-javascript-webaudio library.

This example 17 uses the previously built instrument, but instead of using a button to trigger the instrument, it uses the Metronome block to trigger an event. The button connected to the metronome triggers the first event. The others are triggered automatically as defined by the metronome, thus creating a sequence of events.

Discussion of Results

Our first set of audio blocks were very experimental and had several issues to be solved. The Mosaicode had only C blocks and did not generate Javascript source code. Adapting the tool for another language meant to create Blocks, Ports, Connections and a Code Templates to generate HTML + CSS + Javascript code. As explained before, to connect audio ports was simple but to create a interactive code with GUI elements was not. We solved the problem using a small set of blocks and ports and it is extensible to every new developed block.

Before starting the development of other blocks, we performed a usability test with a group of users with audio development skills. This test investigated whether a VPL/DSL approach would ease the development of Web Audio applications (SCHIAVONI; GONCALVES, 2017b). Also, it allowed naive users to create interesting sounds without domain-specific knowledge.

6. Conclusion

We presented a proposal to simplify the development of applications for the Digital Arts domain by means of the development of a set of Mosaicode, a visual programming environment. A set of blocks was implemented using the Javascript programming language and the Web Audio API, allowing to develop cross-device applications. Through VPLs with DSL code reuse, the Mosaicode blocks generate source code of complete applications. This is the main difference between Mosaicode and other VPLs for the Digital Arts domain like Pure Data or Max/MSP.

Based on others tools like Gibberish and Pure Data, we investigated which Blocks would be necessary to satisfy basic Digital Arts requirements. We started by implementing Web Audio Natives Nodes. Then we implemented blocks not supported by Web Audio Native Nodes using a Script Processor Node. We also included HTML 5 widgets to compose the application's GUIs and created Misc objects to convert values (for example, MIDI to frequency).

We studied block connections, furnishing a way to connect and expand the environment through new components. Several classic examples of computer music algorithms and techniques were presented as initial results, illustrating how the tool can help users to create audio code. Our future works include the expansion of the Mosaicode, using APIs such as Web MIDI, WebGL, Canvas and SVG. We also intend to generate audio code for other programming languages.

6.1 Acknowledgment

We would like to thank the Interdisciplinary Postgraduate Program in Arts, Urbanites and Sustainability – PIPAUS and the Computer Science Postgraduate Program – PPGCC from the Federal University of São João del-Rei.

References

CAMURRI, A.; HASHIMOTO, S.; RICCHETTI, M.; RICCI, A.; SUZUKI, K.; TROCCA, R.; VOLPE, G. Eyesweb: Toward gesture and affect recognition in interactive dance and music systems. *Computer Music Journal*, 24(1), 57-69, 2000.

CLARK, C. B.; TINDALE, A. Flocking: a framework for declarative music-making on the Web. *In SMC Conference and Summer School,* 1550-1557, 2014.

DANKS, *M. Real-time Image and Video Processing in GEM. In International Computer Music Conference,* Thessaloniki, 1997.

GAMMA, E. *Design patterns: elements of reusable object-oriented software.* Pearson Education India, 1995.

GARTON, B.; TOPPER, D. RTcmix-Using CMIX in Real Time. *In International Computer Music Conference,* Thessaloniki. 1997.

GOMES, A.; HENRIQUES, J.; MENDES, A. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. *Educação, Formação & Tecnologias*, 1(1), 93-103, 2008. (ISSN 1646-933X)

GRONBACK, R. C. *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009.

HAEBERLI, P. E. ConMan: a visual programming language for interactive graphics. *In ACM SigGraph Computer Graphics*, 22 (4), 103-111, 1988.

HILS, D. D. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, 3(1), 69-101, 1992.

LANSKY, P. *Cmix release notes and manuals*. Department of Music, Princeton University. Princeton, New Jersey: Princeton University, 1990.

LAZZARINI, V.; COSTELLO, E.; YI, S. Csound on the Web. *Csound conference*. University of Bath, 2014.

LETZ, S.; DENOUX, S.; ORLAREY, Y.; FOBER, D. Faust audio DSP language in the Web. *In Proceedings of the Linux Audio Conference* (LAC-15), Mainz, Germany, 2015.

MAHADEVAN, A.; FREEMAN, J.; MAGERKO, B. *An interactive, graphical coding environment for EarSketch online using Blockly and Web Audio API.* Web Audio Conference WAC-2016, April 4–6. Atlanta, GA: Georgia Institute of Technology, 2016.

MATHEWS, M. V. The digital computer as a musical instrument. *Science*, 142(3592), 553-557, 1963.

MCCARTNEY, J. Rethinking the computer music language: SuperCollider. *Computer Music Journal*, 26(4), 61-68, 2002.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM computing surveys* (CSUR), 37(4), 316-344, 2005.

MOORE, F. R. *Elements of computer music*. New Jersey: Prentice-Hall, 1990.

ORLAREY, Y.; FOBER, D.; LETZ, S. FAUST: an efficient functional approach to DSP programming. *New Computational Paradigms for Computer Music*, 1-33, 2009.

POPE, S. T. Machine tongues XV: Three packages for software sound synthesis. *Computer Music Journal*, 17(2), 23-54, 1993.

PUCKETTE, M. Pure Data: another integrated computer music environment. *Proceedings of the second intercollege computer music concerts*, 37-41, 1996.

PUCKETTE, M.; ZICARELLI, D. MAX-An interactive graphic programming environment. *Opcode Systems*, Menlo Park, CA. 1990.

REAS, C.; FRY, B. Processing: programming for the media arts. *AI & SOCIETY*, 20(4), 526-538, 2006.

RESIG, J.; FRY, B., REAS, C. Processing.js [Visual programming language], 2008. http://processingjs.org/

ROBERTS, C.; WAKEFIELD, G.; WRIGHT, M. The Web Browser As Synthesizer And Interface. *In International Conference on New Interfaces for Musical Expression,* 313-318, 2013.

ROBERTS, C.; WAKEFIELD, G.; WRIGHT, M.; KUCHERA-MORIN, J. Designing musical instruments for the browser. *Computer Music Journal*, 39(1), 27-40, 2015.

ROBERTS, C.; WRIGHT, M.; KUCHERA-MORIN, J.; HÖLLERER, T. Gibber: Abstractions for creative multimedia programming. I*n Proceedings of the 22nd ACM international conference on Multimedia* (pp. 67-76). New York: ACM, 2014.

SCHIAVONI, F. L.; GONCALVES, L. L. From Virtual Reality to Digital Arts with Mosaicode. In *Proceedings of the 19th Symposium on Virtual and Augmented Reality (SVR 2017)* (pp. 200-206), 2017.

SCHIAVONI, F. L.; GONCALVES, L. L. Teste de usabilidade do sistema Mosaicode. In: *Anais do IV Workshop de Iniciação Científica em Sistemas de Informação (WICSI)*. Lavras, MG: Universidade Federal de Lavras, p. 5–8, 2017.

TAYLOR, B.; ALLISON, J. BRAID: A web audio instrument builder with embedded code blocks. *In Proceedings of the 1st international Web Audio Conference,* 2015.

TROWER, J.; GRAY, J. Blockly language creation and applications: Visual programming for media computation and bluetooth robotics control. *In Proceedings of the 46th ACM Technical Symposium on Computer Science Education,* 5-5, New York: ACM, 2015.

VERCOE, B. *CSound manual*. Cambridge, MA: Media Lab, 1986.

WILSON, C., KALLIOKOSKI, J. *Web MIDI API*. on W3C Working Draft 17 March 2015, Available on: https://www.w3.org/TR/webmidi/. Accessed: 2017-09-30.

WYSE, L.; SUBRAMANIAN, S. The viability of the web browser as a computer music platform. *Computer Music Journal*, 37(4), 10-23, 2013.

Flávio Luiz Schiavoni: possui graduação em Ciência da Computação pela Universidade Estadual de Maringá(1999), especialização em Desenvolvimento para WEB pela Universidade Estadual de Maringá(2004), mestrado em Ciência da Computação pela Universidade Estadual de Maringá(2007) e doutorado em Ciências da Computação pela Universidade de São Paulo(2013). Atualmente é Professor Adjunto da Universidade Federal de São João Del-Rei. Tem experiência na área de Ciência da Computação, com ênfase em Metodologia e Técnicas da Computação. Atuando principalmente nos seguintes temas:Música em rede, áudio, MIDI, Ambiente Musical em rede, Performance musical em rede e Comunicação musical síncrona.

Luan Luiz Gonçalves: Graduando em Ciência da Computação pela Universidade Federal de São João Del Rei - UFSJ. Atualmente trabalha no Projeto de Pesquisa P&D em Redes Definidas por Software sobre redes WAN para Data Intensive Science, com o foco no desenvolvimento do back-end do Kytos (https://kytos.io/), colaborando com o front-end e na documentação do projeto. Colaborando também no desenvolvimento da ferramenta Mosaicode (https://mosaicode.github.io/), atuando principalmente nos seguintes temas: Sound Design, Geração de Código, Software Livre, VPL, DSL e Educação Musical auxiliada por computador.

José Mauro da Silva Sandy: possui graduação em Ciência da Computação pela Universidade Presidente Antônio Carlos (2010), com especialização em Arquitetura de Sistemas Distribuídos e Gerenciamento de Projetos. Atualmente é programador, analista de sistemas - Rerum Engenharia de Sistemas LTDA. Tem experiência na área de Ciência da Computação, com ênfase em Arquitetura de Sistemas de Computação, atuando principalmente nos seguintes temas: tecnologia e desenvolvimento. Possui certificações Microsoft e Oracle.