Web Audio application development with *Mosaicode*

Flávio Luiz Schiavoni[1] **, Luan Luiz Gonçalves**[1*]**, André Lucas Nascimento Gomes**[1]

[1]Universidade Federal de São João Del Rei – UFSJ
Computer Science Department - DCOMP
São João Del Rei, MG – Brazil

fls@ufsj.edu.br, luanlg.cco@gmail.com, andgomes95@gmail.com

Abstract

The development of audio application demands a high knowledge about this application domain, traditional programming logic and programming language. 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 visual approach, our environment is a source code generator based on code snippets to create complete applications.

1. Introduction

Since the emergence of Web Audio, the web browser has been used as a platform to create and process real time audio. This API leaded 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. In the raise of the Internet, the web was used only to distribute music and the sound interaction was possible only 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 sonic interaction.

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 [1]. It also enables to incorporate accelerometers, multi touch screens, gyroscopes and other sensors available on mobile devices [2, 3, 4, 5]. It is also possible to incorporate legacy music interfaces with the Web MIDI API [6]. Nonetheless, digital artists often have difficulty starting their research and working with digital art 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 of programming language that allows 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 [7]. This may allow non-programmers or novice programmers to develop complete applications [8] 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 [9]. This is a known approach in arts due to the common use of tools like Pure Data, Max/MSP or Eyesweb.

Another way to simplify the development of computational systems is by using domain-specific languages (DSL) [10]. DSLs have the knowledge of the domain embedded in their structure and are at a higher abstraction level 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 [11]. For this reason, the potential advantages of DSLs include reduced maintenance costs through re-use of built-in features and increased portability, reliability,

*Supported by UFSJ.

optimization and testability [12]. Domain Specific Languages are also common in art field since this approach was used to develop languages like CSound, Supercollider or RTCMix.

In this paper we present *Mosaicode*, 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, Section 3 presents the *Mosaicode* application, Section 4 presents the development of a Block set to *Mosaicode* and JavaScript/Web Audio, Section 5 presents initial discussions of our research. Finally, Section 6 presents Conclusion and Future Works.

2. Related Works

From the point of view of VPLs to audio and music programming, our related works start with Pure Data, Max/MSP and Eyesweb. Pure Data[1] is a Visual Programming Environment for Sound and Music that plays host to GEM environment[13] to 3D graphic processing [14]. Max/MSP[2] is also a music and video real time graphical programming environment [15]. Pure Data and Max/MSP share the same paradigm being both created by the same author, Miller Puckete. Eyesweb[3] is a project focused on real time analysis of body movement and gesture[16].

From the point of view of JavaScript and Web Audio programming into an easier form of programming, related works are several programming libraries like Flocking.js[17], gibber[18], WebCsound[19] and others.

From the point of view of Code generators, Processing[4] is a DSL textual programming language and an IDE that generates code to Graphical Art development. Beyond Processing, a sister project called Processing.js [20] 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[21], a purely functional programming language to signal processing that is able to generate code to several target languages, libraries and APIs.

The idea of using a VPL to generate web based music application was already explored by EarSketch [22] but in this case, a web based environment was used to create the applications based on the Blockly programming environment.

3. About the *Mosaicode*

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.

This tool is a visual programming environment that 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.

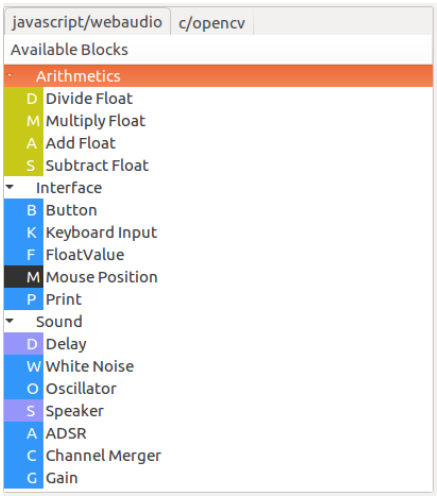

Figure 1: Part of Javascript/Web Audio Block groups

[1]Project Website: `http://puredata.info`.

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

[3]Project website: `http://www.infomus.org/eyesweb_ita.php`.

[4]Project Website: `https://processing.org/`.

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.

Figure 2: Oscillator Block static properties

The Collection of Blocks and Connections creates a **Diagram**, as presented in Figure 3.

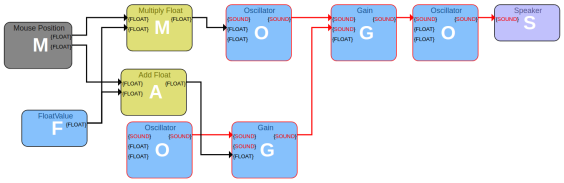

Figure 3: A Diagram with several Blocks interconnected.

Despite it can be seen as a Visual Programming Language, like Pure Data or Max/MSP, *Mosaicode* is not an interpreted environment but a code generator. Every single Block and Connection define code fragments and add code Snippets to 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 the Diagram is completed, it is possible to generate the source code and to run it. Figure 4 presents a generated source code.

The tool has an interface and a plugin manager that allows the creation of new components for the environment, so the tool can be extended, allowing the generation of source code to different programming languages and specific domains.

```
view-source:file:///tmp/javascript/webaudio-1491278886.62/webaudio.html
var block_20 =  context.createOscillator();
var block_20_o0 = null;
var block_20_i1 = function(value){
    block_20.frequency.value = value;
};
var block_20_i2 = function(value){
    oscillator = ''
    if (value < 1) oscillator = 'square';
    if (value == 1) oscillator = 'sine';
    if (value == 2) oscillator = 'sawtooth';
    if (value > 2) oscillator = 'triangle';
    block_20.type = oscillator;
};

// block_60 = Mouse
var block_60_o0 = [];
var block_60_o1 = [];

// block_66 = Multiply Float
var block_66_arg1 = 0;
var block_66_arg2 = 0;
var block_66_o0 = [];
```

Figure 4: Example of generated Source Code.

4. Developing Web Audio Blocks to the *Mosaicode*

Once we started developing a Block set to JavaScript/Web Audio code generation, we investigated which Blocks could be necessary on the environment to satisfy the Digital Arts requirements. The Blocks creation was based on other tools like Gibberish and Pure Data.

We investigated Gibberish and this library 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 [2]. The Misc group contains Sampler objects (play/record), Envelope (ADSR, AD), Line Ramp and others. Later, authors added to this list a Drums category[3] and GUI widgets like Sliders, Buttons, Sensors, knob, Piano and other GUI components.

We also investigated Pure Data native objects and this tool 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.

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 kind of configuration parameters: Fixed parameters and a-rate Audio Parameter. Fixed parameters, like OscillatorNode.type, has fixed values like “sine”, “square”, “sawtooth”, “triangle” and “custom”. This parameters were directly mapped to a Block static property in *Mosaicode*. The other parameter type, a-rate Audio Parameter, like OscillatorNode.frequency, were mapped as an input port. This kind of parameter has also a value field (OscillatorNode.frequency.value) also mapped to a static property.

The AudioNode channel inputs were mapped to input ports and channel outputs were mapped to output ports. These ports are connectible and follow the basic idea of Node connection from Web Audio API, presented on Figure 5.

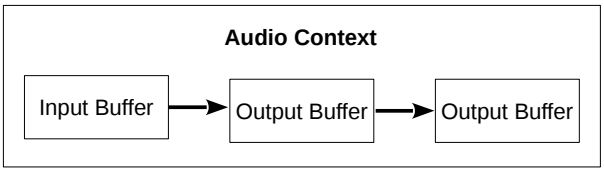

Figure 5: A typical workflow for Web Audio Nodes [23].

Script Processor Node

The high-level Nodes provided by the Web Audio API do not include all possibilities and necessities found in our investigation. To attend other sound processing features, we used a special node called ScriptProcessorNode that enables users to define complete audio systems entirely in JavaScript [1].

The Script processor Node, presented in Figure 6, allows one to develop new Sound Nodes and integrate it to the Web Audio API.

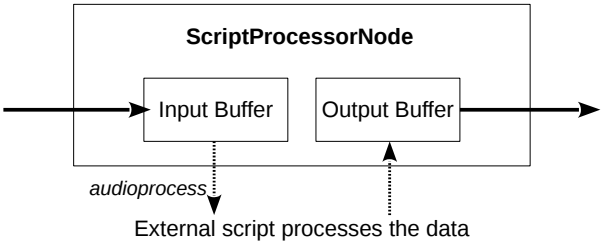

Figure 6: ScriptProcessorNode used to create Blocks like the WhiteNoise generator[24]

Some Blocks considered important to our environment and not present as Native Nodes were implemented using the Script Processor Node like ADSR envelop, AD envelop, White Noise Generator and explicit signal maths. None of these Blocks needed static parameters.

HTML 5 and GUI

Our last set of Blocks included some HTML 5 widgets that compose the application’s GUI, like an object to print values on the screen, get mouse position and keyboard input, button, sliders, change background color and some other GUI features.

We also developed some Misc objects to convert MIDI to frequency, convert 3 float values into a RGB Color and convert char to float and float to char, for example.

All these objects were implemented using basic HTML and javascript code and without using other javascript library.

Block Connections

Once we defined a set of Blocks, the next step on 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 other and create Blocks connections. Since in Javascript there are no pointers, we had to look for another approach.

The solution used was to create an array of call-back functions to be called when some event occurs. This solution granted a responsive interface. Thus, when a GUI object action is performed, it can transmit the result to a list of connected Blocks and grant the value data flow.

5. Discussion

Our first set of audio Blocks were very experimental and had several issues to be solved. The

Mosaicode had only C Blocks and we never tried to generate Javascript source code. Adapting the tool to another language meant to create Blocks, Ports, Connections and a Code Template 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 simple.

The solution created solved the problem to our small set of Blocks and ports and it is extensible to every new developed Block. Since these issues were solved, to create several others Blocks is a more easy and trivial task.

Before starting the development of other Blocks, we performed a usability test with a group of users with audio development skills. This test could prove that a VPL/DSL really ease the development of Web Audio applications[25]. Also, it allowed naive users to create interesting sounds even without specific domain knowledge.

6. Conclusion

We present in this article a proposal to simplify the development of applications for the Digital Arts domain by means of the development of a set of Blocks to *Mosaicode*, a visual programming environment that can be used to develop systems in specific domains. This set of Blocks were implemented using Javascript programming language and Web Audio API, allowing to develop cross-device application and enjoy powerful tools provided by web technologies.

Beyond the simplicity of the VPLs with DSL code reuse, developing the set of Blocks in the *Mosaicode* enable the possibility of generating the source code of applications from VPL Diagram. This is the main difference between Mosaicode and other VPLs for the Digital Arts domain like Pure Data or Max/MSP.

To define the set of Blocks, we investigated which Blocks could be necessary to the environment to satisfy the Digital Arts requirements, based on others tools like Gibberish and Pure Data. We started our Block development implementing the Web Audio Natives Nodes, then we implemented some Blocks considered important to our environment that are not present in Web Audio Native Nodes using the the Script Processor Node. To complete the initial set of Blocks we included some HTML 5 widgets that compose the application's GUI and some Misc objects to convert values like MIDI to frequency.

This work also discusses the Block connection, presenting a way to connect them and expand the environment creating new components. We also cite an usability test performed before starting the development of other Blocks, that could confirm that a VPL/DSL really ease the development of Web Audio applications.

Combining web technologies with *Mosaicode*, this work results in a set of Blocks that provides a quick, simple and practical way to develop cross-device applications to Digital Art domain.

Our Future works include expansion of the environment developing others APIs to the *Mosaicode* like Web MIDI, WebGL, Canvas and SVG. We also intend to generate audio code for other programming languages.

References

[1] Charlie Roberts, Matthew Wright, JoAnn Kuchera-Morin, and Tobias Höllerer. Rapid creation and publication of digital musical instruments. In *NIME*, pages 239–242, 2014.

[2] Charles Roberts, Graham Wakefield, and Matthew Wright. The web browser as synthesizer and interface. In *NIME*, pages 313–318. Citeseer, 2013.

[3] Charles Roberts, Graham Wakefield, Matthew Wright, and JoAnn Kuchera-Morin. Designing musical instruments for the browser. *Computer Music Journal*, 39(1):27–40, 2015.

[4] Ben Taylor and Jesse Allison. Braid: A web audio instrument builder with embedded code blocks. In *Proceedings of the 1st international Web Audio Conference*, 2015.

[5] Stephane Letz, Sarah Denoux, Yann Orlarey, and Dominique Fober. Faust audio dsp language in the web. In *Proceedings of the Linux Audio Conference (LAC-15), Mainz, Germany*, 2015.

[6] Chris Wilson and Jussi Kalliokoski. Web midi api. https://www.w3.org/TR/webmidi/, 2015. Accessed: 201-09-30.

[7] Paul E. Haeberli. Conman: A visual programming language for interactive graphics. *SIGGRAPH Comput. Graph.*, 22(4):103–111, June 1988.

[8] Anabela Gomes, Joana Henriques, and António Mendes. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. *Educação, Formação & Tecnologias-ISSN 1646-933X*, 1(1):93–103, 2008.

[9] Daniel D Hils. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, 3(1):69–101, 1992.

[10] R. C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley, New York, 2009.

[11] Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[12] Arie Van Deursen and Paul Klint. Domain-specific language design requires feature descriptions. *CIT. Journal of computing and information technology*, 10(1):1–17, 2002.

[13] Mark Danks. Real-time image and video processing in gem. In *ICMC*, 1997.

[14] Miller Puckette et al. Pure data: another integrated computer music environment. *Proceedings of the second intercollege computer music concerts*, pages 37–41, 1996.

[15] 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.

[16] 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.

[17] Colin BD Clark and Adam Tindale. Flocking: a framework for declarative music-making on the web. In *SMC Conference and Summer School*, pages 1550–1557, 2014.

[18] Charles Roberts, Matthew Wright, JoAnn Kuchera-Morin, and Tobias Höllerer. Gibber: Abstractions for creative multimedia programming. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 67–76. ACM, 2014.

[19] Victor Lazzarini, Edward Costello, Steven Yi, et al. Csound on the web. 2014.

[20] John Resig, Ben Fry, and Casey Reas. Processing. js, 2008.

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

[22] Anand Mahadevan, Jason Freeman, and Brian Magerko. An interactive, graphical coding environment for earsketch online using blockly and web audio api. 2016.

[23] Mozilla Developer Network. Web audio api. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API, 2017. Accessed: 2017-06-30.

[24] Mozilla Developer Network. Scriptprocessornode. https://developer.mozilla.org/pt-BR/docs/Web/API/ScriptProcessorNode, 2017. Accessed: 201-09-30.

[25] Teste de usabilidade do sistema mosaicode. In *In: Simpósio Brasileiro de Sistemas de Informação, 2017, Lavras. Anais [do] IV Workshop de Iniciação Científica em Sistemas de Informação (WICSI)*, pages 5–8, Lavras - MG: Universidade Federal de Lavras - UFLA, 2017.