UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Luan Luiz Gonçalves

Sound Design com o *Mosaicode*

São João del-Rei

2017

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Luan Luiz Gonçalves

Sound Design com o *Mosaicode*

Monografia apresentada como requisito da disciplina de Projeto Orientado em Computação II do Curso de Bacharelado em Ciência da Computação da UFSJ.

Orientador: Flávio Luiz Schiavoni

Universidade Federal de São João del-Rei – UFSJ

Bacharelado em Ciência da Computação

São João del-Rei

2017

Luan Luiz Gonçalves

Sound Design com o *Mosaicode*

Monografia apresentada como requisito da disciplina de Projeto Orientado em Computação II do Curso de Bacharelado em Ciência da Computação da UFSJ.

Trabalho aprovado. São João del-Rei, 13 de dezembro de 2017:

Flávio Luiz Schiavoni
Orientador

Bruno Soares Santos
Convidado 1

Elder José Reioli Cirilo
Convidado 2

São João del-Rei
2017

Este trabalho é deficado aos meus pais, irmãos e a toda minha família que não mediram esforços para que eu chegasse até essa etapa da minha vida, me apoiando sempre.

Agradecimentos

A Deus, por me proteger e guiar meus passos nessa caminhada.

Aos meus pais Hélio e Rose Mary, por abraçarem mais esse desafio comigo. O apoio de vocês foi fundamental para que esse projeto se concretizasse. São vocês minha base e inspiração.

Aos meus irmãos Ênio e Acríssio pelo incentivo e apoio constantes, me dando forças nessa caminhada.

A todos os meus familiares que torceram por mim.

Ao meu orientador Flávio Luiz Schiavoni, pela dedicação, paciência e compreensão. Sou grato por todos os ensinamentos.

A todos os professores pelos os ensinamentos nesse processo de formação.

A todos os amigos e colegas, pelo companheirismo e incentivo.

À UFSJ pelas oportunidades e pelo auxílio financeiro.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Resumo

A música tem sido influenciada pela tecnologia digital ao longo das últimas décadas, proporcionando a criação de novos instrumentos eletrônicos e de novas maneiras de fazer música. Com o computador, a composição musical conseguiu ir além das limitações humanas e dos instrumentos analógicos, exigindo de músicos e compositores conhecimentos de programação de computadores para o desenvolvimento de aplicações musicais. Com o intuito de simplificar a criação de aplicações musicais, este trabalho apresenta a construção de um conjunto de blocos para aplicações de áudio, no ambiente de programação *Mosaicode*, simplificando o Sound Design e tornando a síntese e manipulação de áudio mais acessível aos artistas digitais. Essa construção resultou em uma Linguagem de Programação Visual (*VPL – Visual Programming Language*) e na biblioteca *libmosaic-sound*, ambos para o domínio específico de Computação Musical.

Palavras-chaves: Mosaicode. libmosaic-sound. Biblioteca. VPL. DSL. Computação Musical. Arte Digital.

Abstract

The music has been influenced by digital technology over the last few decades, providing the creation of new electronic instruments and new ways of music making. With the computer, the musical composition could tresspas the human limitations and analogical instruments, requiring musicians and composers computer programming knowledge for the development of musical applications. In order to simplify the creation of musical applications, this work presents the construction of a set of blocks for audio applications in the *Mosaicode* programming environment, simplifying Sound Design and making the synthesis and manipulation of audio more accessible to digital artists. This construction resulted in a Visual Programming Language (VPL) and *libmosaic-sound* library, both for the specific domain of Music Computing.

Key-words: Mosaicode. libmosaic-sound. Library. VPL. DSL. Computer Music. Digital Art.

Lista de figuras

Lista de tabelas

Lista de códigos-fonte

Lista de abreviaturas e siglas

ALSA	Advanced Linux Sound Architecture
API	Application Programming Interface
DSL	Domain-Specific (Programming) Language
MIDI	Musical Instrument Digital Interface
OSC	Open Sound Control
OSS	Open Sound System
TAD	Tipo Abstrato de Dados
VPL	Visual Programming Language

Sumário

1 Introdução

A música tem sido influenciada pela tecnologia ao longo de décadas, principalmente após avanços tecnológicos que foram fundamentais para essa aproximação entre a ideia de música e tecnologia, proporcionando instrumentos eletrônicos e maneiras de fazer música (MILETTO et al., 2004; ZUBEN, 2004). Com o computador, a composição musical vai além das limitações do corpo do artista e de seus instrumentos, mas exige conhecimentos de programação de computadores para o desenvolvimento de aplicações de áudio, uma vez que compor um música e desenvolver um software são duas tarefas muito diferentes (IAZZETTA, 1997). Artistas digitais muitas vezes possuem dificuldade de iniciar sua pesquisa e trabalho com arte digital devido ao não conhecimento programação de computadores.

Para simplificar o desenvolvimento de aplicações, há a opção da utilização de Linguagens de Programação Visual (VPL – *Visual Programming Language*), que permitem ao usuário programar utilizando uma notação bidimensional e interagir com o código através de uma representação gráfica em vez de editar um fluxo unidimensional de caracteres, tendo que decorar comandos e sintaxes textuais (HAEBERLI, 1988). A programação por meio de diagramas pode trazer facilidade para o desenvolvimento de sistemas. Isto pode permitir que não programadores ou programadores iniciantes desenvolvam aplicações (GOMES; HENRIQUES; MENDES, 2008). Além disso, a abstração de código por meio de diagrama pode trazer praticidade na alteração do código fazendo com que as mesmas sejam bastante adequadas para rápida prototipação (HILS, 1992).

Outra forma de simplificar ainda mais o desenvolvimento de sistemas computacionais é utilizando linguagens de domínio específico (DSL – *Domain-Specific (Programming) Language*) (GRONBACK, 2009). Por terem o conhecimento do domínio embutido em sua estrutura, as DSL's estão em um nível de abstração maior do que a linguagens de programação de propósito geral. Isso faz com que o processo de desenvolvimento de sistemas dentro do seu domínio seja facilitado e torne-se mais eficiente, pois as DSLs exigem mais um conhecimento sobre o domínio do que um conhecimento sobre programação (MERNIK; HEERING; SLOANE, 2005). Por essa mesma razão, as vantagens potenciais das DSL's incluem custos de manutenção reduzidos por meio de reutilização de funcionalidades já criadas e maior portabilidade, confiabilidade, otimização e testabilidade (DEURSEN; KLINT, 2002).

Com a proposta de simplificação do desenvolvimento de aplicações de áudio, temos a ferramenta *Mosaicode*, um ambiente de programação visual que tem como foco permitir o desenvolvimento sistemas para os domínios específicos de Arte Digital. O desenvolvimento

dos sistemas é realizado através da construção de diagramas, compostos por blocos e conexões entre eles, e através do diagrama é gerado um código-fonte em uma linguagem de programação específica. A ferramenta oferece recursos para criar e editar componentes (blocos, portas e padrão de código) para o ambiente. Com isto, a ferramenta pode ser estendida, permitindo a geração de código-fonte para qualquer linguagem de programação e domínio específico – construção de uma VLP para um DSL.

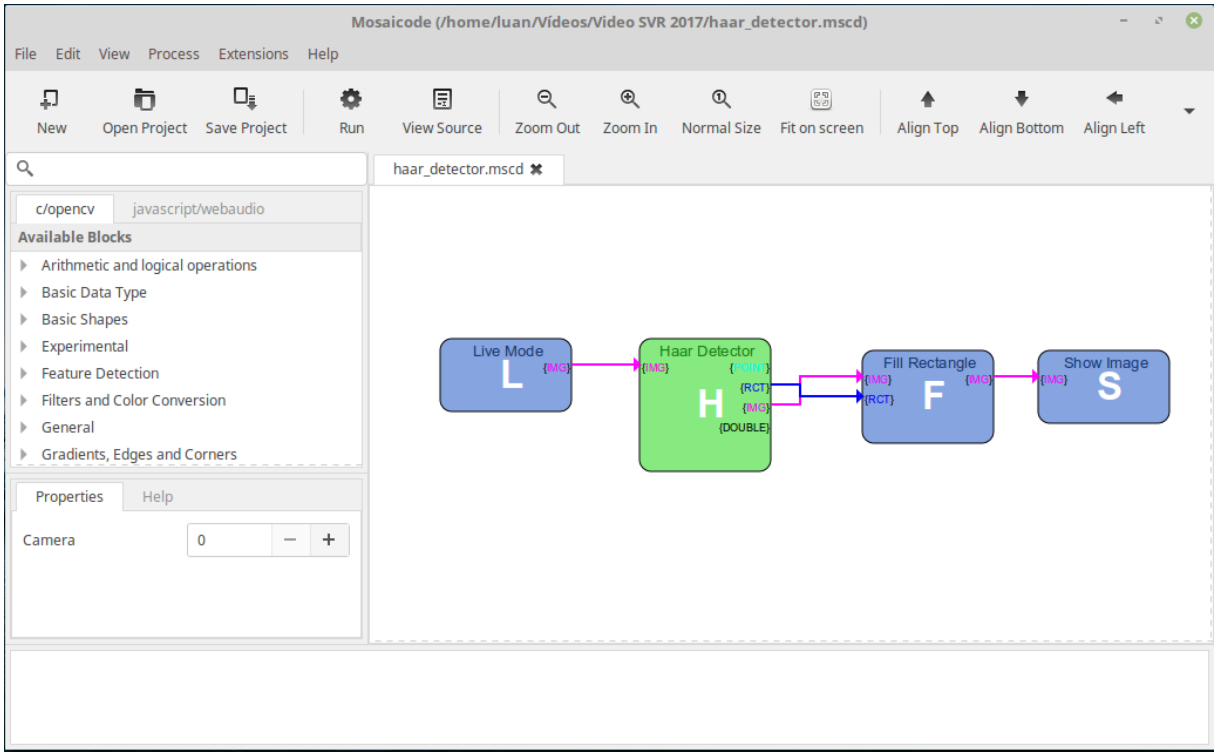

Figura 1: Ambiente de programação Visual da ferramenta Mosaicode

Inicialmente o *Mosaicode* foi desenvolvido para gerar aplicações para o domínio de Visão Computacional, mas aos poucos está sendo remodelado para outras áreas. O objetivo é unir as áreas necessárias para suprir as demandas da Arte Digital, permitindo trabalhar com áudio, imagens, sensores, controladores e redes de computadores. Anteriormente a este projeto, foi desenvolvido um protótipo funcional para geração de aplicações de áudio em Javascript, baseado na API Web Audio, acrescentando ao *Mosaicode* o primeiro conjunto de blocos para aplicação de áudio com o foco para aplicações Web.

Este trabalho apresenta a construção de um conjunto de blocos para aplicação de áudio, dentro do ambiente da ferramenta *Mosaicode* e na linguagem de programação C. A construção desse conjunto no *Mosaicode* tem como intuito simplificar o Sound Design (manipulação e criação de sons), tornando a síntese e manipulação de áudio mais acessível aos artistas digitais. Para a linguagem do código gerado foi escolhido a linguagem de programação C, utilizado a biblioteca *libmosaic-sound*. Essa biblioteca também é resultado deste projeto, desenvolvida para auxiliar na construção do conjunto de blo-

cos, tendo o intuito facilitar a construção dos blocos reduzindo o esforço necessário para implementá-los.

A biblioteca *libmosaic-sound* também foi construída com o intuito oferecer recursos que a API PortAudio não traz implementado e reduzir o esforço de programação por meio do uso da biblioteca na linguagem de programação C. Assim o usuário consegue sem muito esforço gerar aplicações para o domínio de Computação Musical. A estrutura criada para biblioteca oferece essa facilidade de uso e tornou mais fácil implementação dos blocos no *Mosaicode*, resultado na VPL para o domínio de Computação Musical.

Já existem VPL's para o domínio de Computação Musical, como por exemplo o *Pure Data* e o *EyesWeb*, apresentadas na Subseção 1.5 deste documento. No entanto, como será apresentado, nenhuma dessas VPL's geram códigos. Construindo a VLP proposta no ambiente do *Mosaicode*, teremos como vantagens a geração de um código-fonte, que poderá ser estudado, adaptado, executado e usado da forma que o usuário desejar.

1.1 Cenário

O Mosaicode é um ambimente de programação visual baseado em componentes chamados de Blocos. O bloco é uma peça mínima de código no fluxo de programação da ferramenta, representando uma abstração de código de uma funcionalidade.

Um Bloco pode ter propriedades estáticas ou dinâmicas. Propriedades estáticas são configuradas por uma janela do ambiente e representam características do processamento do Bloco como tamanho, largura, cor, ângulo, entre outras. Propriedades estáticas são valores de configuração do código do Bloco sendo que estes valores são fixados no momento da geração do código. A Figura 2 apresenta as propriedades estáticas de um oscilador de áudio.

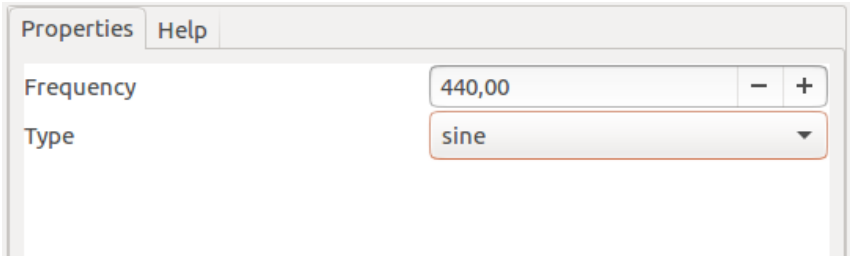

Figura 2: Propriedades estáticas de um Bloco Oscilador.

As propriedades de um Bloco também podem ser dinâmicas de maneira que o mesmo possa ter parâmetros configurados por outros blocos. As propriedades dinâmicas de um Bloco são representadas por uma **Porta** de entrada e a saída do processamento do bloco é representada por uma porta de saída. Entradas e saídas definem o fluxo de dados da programação e podem ser conectadas por meio de **Conexões**.

As Conexões definem a comunicação entre as portas de Blocos distintos. Uma Conexão possui um tipo específico de dados e irá representar como o valor de saída de um bloco de código pode servir como parâmetro de entrada para outro bloco de código no momento da geração do código. Além de ser tipadas, Conexões também são definidas para linguagens de programação específicas pois a geração do código irá depender da compatibilidade do código gerado entre os blocos e as conexões.

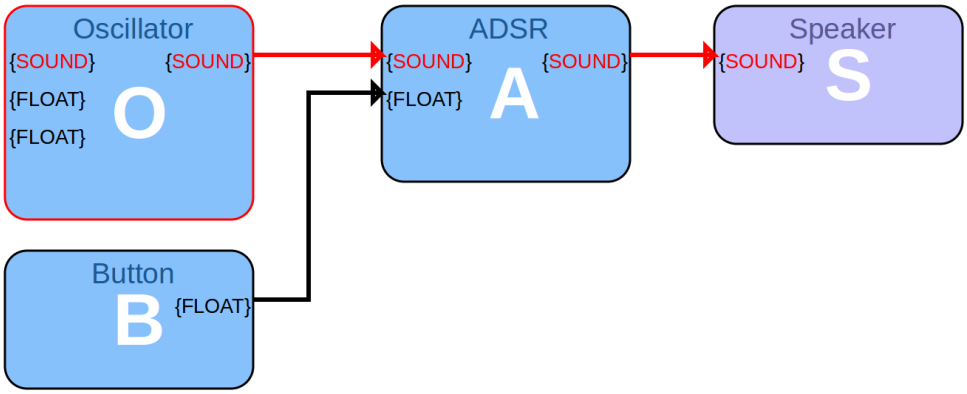

Figura 3: Exemplo de um diagrama no *Mosaicode*.

A coleção de Blocos e Conexões definem um **Diagrama**, como apresentado na Figura 3. Cada Diagrama pode gerar um código fonte individual e pode ser salvo em um formato de arquivo próprio da ferramenta. A geração do código do Diagrama depende da escolha de um **Padrão de Código** que também é associado a uma linguagem de programação.

1.1.1 Arquitetura do Mosaicode

A arquitetura atual da ferramenta baseia-se no modelo MVC, conforme ilustrado pela Figura 4.

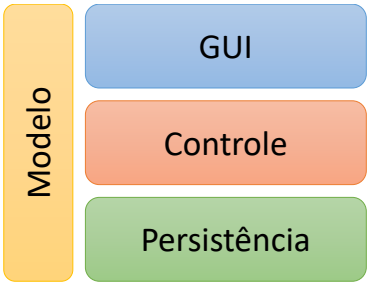

Figura 4: Arquitetura da ferramenta.

A **Camada de Persistência** é responsável por salvar e carregar os Diagramas feitos na ferramenta e também por carregar e persistir as extensões da ferramenta. Esta camada está implementada utilizando a biblioteca BeautifulSoup (RICHARDSON, 2015).

Com isto, toda a persistência XML da ferramenta está isolada nesta camada, o que permite alterar facilmente esta dependência futuramente. A camada de persistência utiliza a camada de Modelo e seus métodos são chamados exclusivamente pela camada de Controle.

A **Camada de Controle** é responsável pelas ações do ambiente. É nesta camada que ocorrem a ligação entre as demais classes do ambiente, garantindo um baixo acoplamento do código. Todas as ações do ambiente estão definidas na camada de controle que toma decisões sobre quais classes devem tomar parte de quais ações. Uma classe especial do controle é o Gerador de Código (CodeGenerator) que possui a missão de validar diagramas e gerar código.

A **Camada de GUI** define a interface de usuário e é baseada no Gimp Tool Kit (GTK[1]) versão 3. Todos os componentes gráficos do ambiente extendem uma classe Gtk e especializa esta classe para as necessidades da ferramenta. Assim, temos classes Menu, Janela Principal, Barra de Ferramentas e assim por diante, conforme ilustrado pela Figura 5. Os componentes gráficos como Blocos, Conexões e Diagramas baseiam-se na biblioteca Goocanvas[2].

A **Camada de Modelo** representam os componentes do sistema como Porta, Conexões, Blocos e Padrão de Código, conforme ilustrado pela Figura 6. Todo Padrão de código (CodeTemplate) possui um Diagrama, sendo que um Diagrama é uma coleção de Blocos e Conexões. Um Bloco pode possuir portas de Entrada e de Saída e uma Conexão será obrigatoriamente uma relação entre um Bloco origem (source) e um Bloco destino (Sink) e suas respectivas portas.

1.1.2 Extendendo a arquitetura por plugins e bibliotecas de blocos

Para permitir uma maior flexibilidade e adequação do código gerado, o ambiente Mosaicode utiliza extensões. A extensão do ambiente se dá pela criação de novos Blocos, Portas e Padrões de Código, que compõem um biblioteca de blocos para o *Mosaicode*. Essa funcionalidade de criar novos componentes também é uma extensão , disponibilizada pelo plugin *Library Manager*, que ao ser instalado, cria um novo item de menu na barra de menu do sistema.

Os componentes, sejam eles Blocos, Portas ou Padrões de Código, pode ser feitos tanto como uma classe Python quanto como um arquivo XML. Arquivos XML podem ainda estar em espaço de usuário ou instalada com o sistema. O carregamento da ferramenta se dá com o carregamento dos plugins na seguinte ordem: Classes python instaladas com o sistema, Arquivos XML instalados com o sistema e Arquivos XML no espaço de usuário. Assim, se o usuário quiser alterar e personalizar Blocos em sua instância da fer-

[1] Disponível em `https://www.gtk.org/`.
[2] Disponível em <https://wiki.gnome.org/Projects/GooCanvas>.

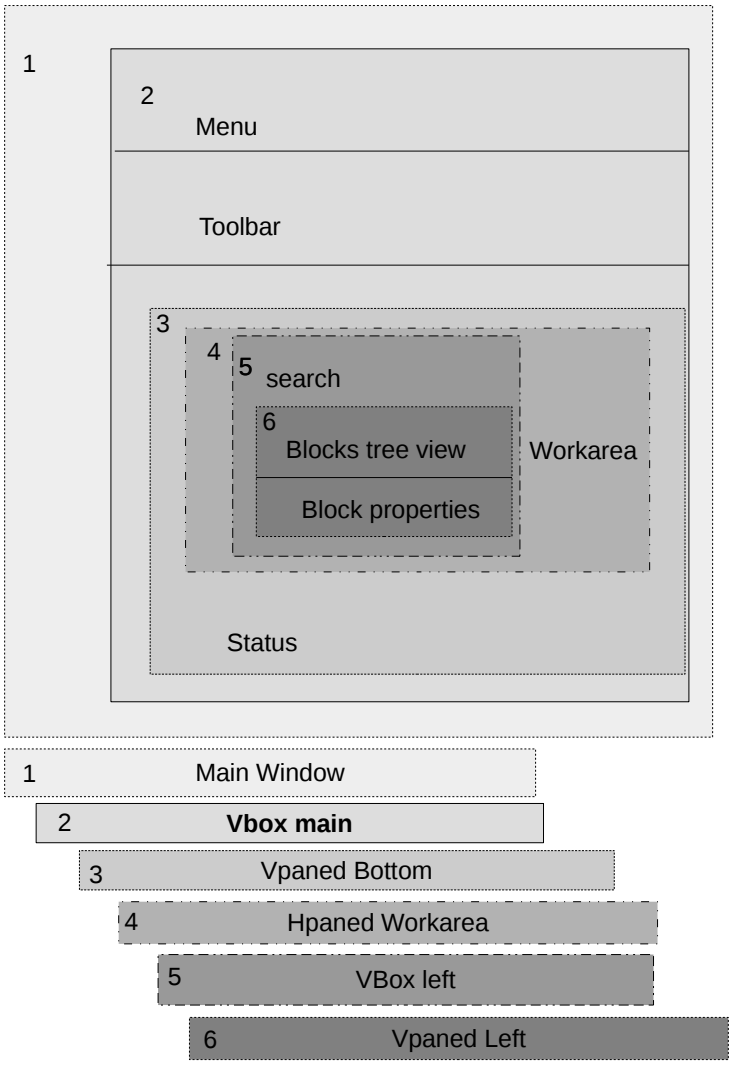

Figura 5: Estrutura dos componentes de GUI.

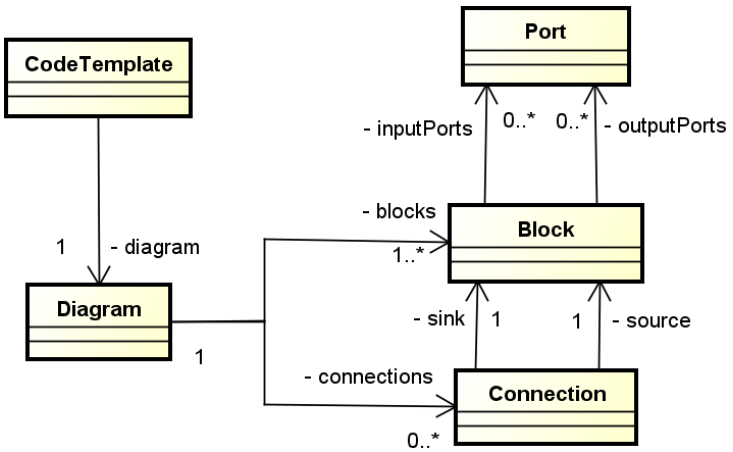

Figura 6: Classes da camada de modelo.

ramenta, o mesmo consegue fazer de maneira a alterar a instalação da ferramenta apenas para seu usuário e sem depender de senha de acesso especial para isto.

A ferramenta conta atualmente com um conjunto de Blocos para a geração de có-

digo na linguagem C baseadas na biblioteca openCV (PULLI et al., 2012) e um conjunto de Blocos para a geração de código na linguagem Javascript baseados na *API Webaudio* (ROBERTS; WAKEFIELD; WRIGHT, 2013).

1.2 Objetivos Gerais

Este trabalho tem como objetivo implementar um conjunto de blocos para aplicação de áudio no *Mosaicode*na linguagem C. Este conjunto definirá uma linguagem de programação visual, com o intuito de simplificar o desenvolvimento de aplicações para o domínio de Computação Musical, tornando o Sound Design mais acessível aos artistas digitais.

1.3 Objetivos Específicos

Além do objetivo geral, busca-se alcançar os seguintes objetivos específicos:

- Definir os recursos a serem implementados, para que artistas digitais possam trabalhar com Sound Design;
- Os recursos definidos devem abranger fontes sonoras em forma de ondas simples e capacitar a construção de sínteses de áudio, efeitos sonoros e envelopes, para a geração de sons complexos.
- Implementação dos recursos na linguagem de programação C/PortAudio, construindo uma biblioteca, intitulada *libmosaic-sound*, para trabalhar com Sound Design;
- Implementação dos recursos no ambiente *Mosaicode*:
  - Os recursos serão implementados em forma de blocos, construindo uma VPL para trabalhar com Sound Design;
  - A geração de código-fonte será na linguagem de programação C, baseado na biblioteca *libmosaic-sound*.

1.4 Metodologia

O desenvolvimento desde projeto foi dividido em 3 etapas. Estas etapas são apresentadas na Figura 7 e detalhadas neste capítulo.

O projeto começou com a escolha da linguagem de programação para implementação da biblioteca e do código gerado e a escolha de uma API de áudio para auxiliar

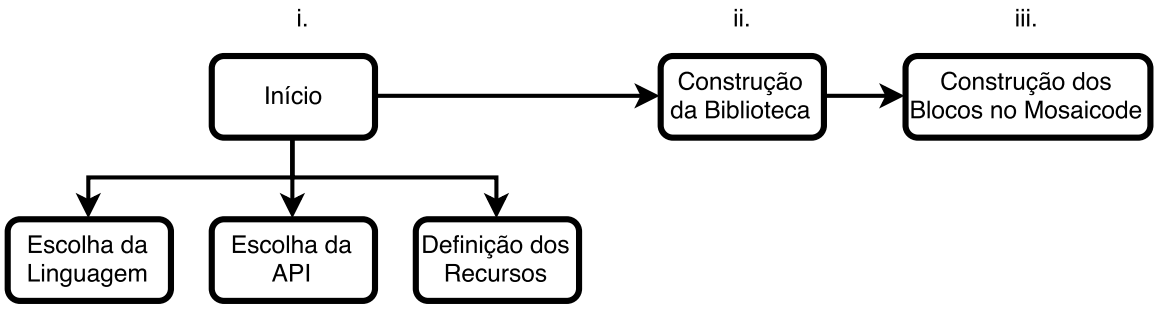

Figura 7: Fluxograma da metodologia do POC II

no desenvolvimento. O processo de escolha da linguagem e API consistiu na leitura de artigos, buscando encontra opiniões sobre linguagens e API's eficientes que dispõem dos recursos para desenvolvimento de aplicações de áudio.

Houve a preocupação de escolher uma linguagem adequada para o projeto proposto assim como uma API que possam simplificar o desenvolvimento, trazendo funcionalidades já implementadas e oferendo boa portabilidade, licença de software livre e possibilitando a adição de recursos MIDI futuramente (SCHIAVONI; GOULART; QUEIROZ, 2012). A escolha da API pode influenciar na escolha da linguagem de programação pois a compatibilidade entre ambas é fundamental para simplificar o desenvolvimento de sistemas. Outra preocupação para implementação de aplicações de áudio é a eficiência da linguagem de programação. Aplicações de áudio tem uma taxa de amostragem alta, geralmente em 44100 Hz, isso implica no processamento de 44100 amostras por segundo, de um sinal digital. A linguagem escolhida deve suportar esse processamento, caso contrário, o resultado da aplicação não será o esperado (JIANG; ALLRED; HOCHSCHILD, 2002).

Após definir a linguagem de programação e a API de áudio, foi realizado um levantamento de recursos necessário para uma VPL/DSL que possibilite artistas digitais desenvolverem aplicações para o domínio de Computação Música e trabalharem com Sound Design. Os recursos foram escolhidos com base na ferramenta Pure Data (PUCKETTE et al., 1997) e na biblioteca Gibberish Ugens (ROBERTS; WAKEFIELD; WRIGHT, 2013), por meio da investigação dos recursos das mesmas.

Com a linguagem de programação, API e recursos definidos (etapa *i*), a próxima etapa consistiu na implementação destes recursos, construindo uma biblioteca para trabalhar com Sound Design (etapa *ii*). Essa biblioteca, intitulada *libmosaic-sound*, traz recursos que não são implementados pela API PortAudio (definida na etapa *i*), buscando uma forma mais fácil de implementar e exigindo um menor esforço de programação. O livro *DAFX - Digital Audio Effects* (ZOLZER, 2011) foi usado como base na implementação dos recursos.

A última etapa *iii* consistiu na implementação dos recursos em forma de blocos no *Mosaicode*, reaproveitando o código da biblioteca implementada. Para implementação

dos blocos e geração do código-fonte foi necessário definir um padrão de código, que define como implementar os blocos e informa ao gerador de código do *Mosaicode* como gerar código usando o conjunto de blocos criados posteriormente. Sendo assim, o primeiro passo desta etapa *iii* foi observar, na biblioteca criada na etapa *ii*, as partes do código que sempre serão comuns no código gerado e as partes que devem ser geradas com base nos diagramas do *Mosaicode*, para definir o padrão de código. Definindo o padrão de código, o gerador do *Mosaicode* consegue interpretar o diagrama para o código-fonte.

O último passo da etapa *iii* consistiu na implementação dos blocos no *Mosaicode*. Cada bloco contém a abstração de código de cada recurso definido na etapa *iii*, fazendo um reuso de código por meio da utilização da biblioteca criada na etapa *ii*.

1.5 Ferramentas Relacionadas

Por serem amplamente utilizadas por artistas digitais, as feramentas a seguir são consideradas relacionadas com esta pesquisa.

O **Processing**[3] é uma linguagem de programação e um ambiente de programação (IDE) desenvolvido no Media Lab do MIT(REAS; FRY, 2007). A linguagem de programação contém abstrações para diversas operações com imagens e desenhos e permite a prototipação rápida de animações em pouquíssimas linhas de código. A ferramenta tem como propósito ser usada para ensino de programação e para desenvolvimento de arte gráfica. A partir de programas feitos em Processing, chamados de sketchs ou esboços, a IDE gera código Java e executa o mesmo.

O **Pure Data**[4] ou simplesmente PD é um ambiente de programação gráfica em tempo real para áudio e vídeo (PUCKETTE et al., 1997). Um programa em PD é chamado de *patch* e é feito, segundo o próprio autor, por meio de "caixinhas" conectadas por "cordinhas". Este ambiente é extensível por meio de plugins, chamados de *externals*, e possui diversas bibliotecas que permitem a integração do PD com sensores, Arduinos, wiimote, mensagens OSC, Joysticks e outros. O PD é um projeto open source e é vastamente utilizado por artistas digitais. O *Engine* do ambiente foi ainda empacotado como uma biblioteca, chamada libpd(BRINKMANN et al., 2011), que permite utilizar o PD como *engine* de som em outros sistemas.

O **Max/MSP**[5] é também um ambiente de programação gráfica em tempo real para áudio e vídeo (WRIGHT et al., 1999). Desenvolvido pelo criador do Pure Data, Miller Puckete, o Max é atualmente mantido e comercializado pela empresa Ciclying 74.

[3] Disponível em <https://processing.org/>.
[4] Disponível em <http://www.puredata.info>
[5] Disponível em <https://cycling74.com/products/max>.

Característica	Processing	Pure Data	Max/MSP	Eyesweb	Mosaicode
Licença	GPL e LGPL	BSD	Proprietário	Proprietário	GPL V3
Funcionamento	Gera Código	Interpreta	Interpreta	Interpreta	Gera Código
Programação	Textual	Visual	Visual	Visual	Visual
Aceita plugins?	Sim	Sim	Sim	Sim	Sim

Tabela 1: Comparação entre ferramentas relacionadas

O **EyesWeb**[6] é um ambiente de programação visual focado em processamento e análise de movimentos corporais em tempo real (CAMURRI et al., 2000). De acordo com os autores, estas informações podem ser utilizadas para criar e controlar sons, músicas, mídias visuais, efeitos e até mesmo atuadores externos. O Eyesweb é gratuito, de código aberto mas proprietário e possui uma licença própria para sua distribuição.

As ferramentas relacionadas acima foram comparadas com o Mosaicode. Esta comparação pode ser observada na Tabela 1. Comparando essa tabela, apenas o *Mosaicode* e o *Processing* geram código. Ambas são ferramentas FLOSS, mas se diferem na programação, sendo que o *Mosaicode* é a única ferramenta que a programação é visual e ao menos tempo gera código, possibilitando o estudo do código-fonte gerado, a modificação e uso da maneira que desejar. Todas as ferramentas aceitam extensões por meio de plugins e apenas o *Max/MSP* e o *EyesWeb* são ferramentas proprietárias.

Todas as ferramentas relacionadas são ambientes de programação, sendo que o Processing é o único ambiente de programação textual. Os ambientes de programação visuais listados funcionam com a interpretação dos diagramas e por isto o produto desenvolvido nestes ambientes são atrelados ao próprio ambiente. A Ferramenta Mosaicode é, nesta comparação, a única de programação visual com geração de código, o que permite que o produto desenvolvido seja utilizado sem o ambiente de programação.

1.6 Organização deste trabalho

Este trabalho está organizados da seguinte forma: No Capítulo 2, é apresentado uma análise das referências bibliográficas. No Capítulo 3 é apresentado o desenvolvimento de cada etapa deste projeto. No Capítulo 4 é apresentado como resultado a construção da biblioteca, o conjunto de blocos propostos e outro resultados como artigos publicados. Por fim, no Capítulo 5, é apresentado a conclusão e trabalhos futuros.

[6] Disponível em <http://www.infomus.org/>

2 Revisão da Literatura

Este capítulo aborda conceitos e trabalhos relacionados. É discutido sobre VPL e DSL, que deixam mais claro a importância desse trabalho que pretende facilitar o desenvolvimento de aplicações de áudio. Como esse trabalho agrega mais funcionalidades ao *Mosaicode* para a geração de aplicações para as Artes Digitais, também é discutido neste capítulo sobre Arte Digital e Realidade Virtual, áreas foco do *Mosaicode*.

2.1 Linguagem de Programação Visual

Segundo (GUDWIN, 1997), desenvolver um software usando recursos gráficos tende a ser mais fácil do que escrever um código de programação, levando em consideração que é mais fácil entender gráficos do que textos. O ser humano consegue processar informações de maneira otimizada, utilizando melhor o poder do cérebro, quando se trata de dados multidimensionais, como diagramas. Estas representações bidimensionais tendem a ser uma descrição de alto nível de programação, aproximando aos usuários e aos objetos manipulados no mundo real (MYERS, 1990).

Essa facilidade de entender e processar elementos gráficos permite um desenvolvimento de forma rápida e fácil, fazendo com que usuários com pouca habilidade de programação consigam gerar softwares usando uma VPL, dentro de um escopo bem definido, o que restringem a aplicações bem específicas, diferente de linguagens de programação para propósitos gerais (GUDWIN, 1997).

2.2 Linguagem de Domínio Específico

As Linguagens de Domínio Específicos são definidas para um determinado domínio, oferecendo uma notação adequada ao mesmo. Essas linguagem tendem ser mais expressivas e fáceis de uso em seu domínio, comparando com linguagens de programação de propósito geral. Por ter uma notação adaptada ao seu domínio, as DSLs também podem oferecer um desenvolvimento mais produtivo, tendo como desvantagem o custo do seu desenvolvimento (MERNIK; HEERING; SLOANE, 2005).

Os autores (DEURSEN; KLINT; VISSER, 2000) definem DSL como uma linguagem de programação ou linguagem de especificação executável que oferece, através de notações e abstrações apropriadas, poder expressivo focado e geralmente restrito a um domínio de problema específico[1].

[1] Essa definição foi traduzida do artigo *Domain-Specific Languages: An Annotated Bibliography* (DEURSEN; KLINT; VISSER, 2000)

2.3 Arte Digital e Realidade Virtual

A convergência cultural da arte, da ciência e da tecnologia proporcionou aos artistas um novo desafio para criar arte em um universo digital (GRAU, 2003). Este desafio impactou e transformou radicalmente as atividades tradicionais de arte como música, pintura, dança e escultura. Além disso, surgiram novas formas de arte inteiramente novas e são agora reconhecidas como práticas artísticas, como arte de rede, arte de mídia, instalação digital e Realidade virtual (WANDS, 2007).

A interseção entre Artes Digitais e Realidade Virtual permanece no domínio sensível onde os artistas pretendem criar sensações para o público. Essas sensações nas artes são comumente criadas pelo uso de imagens e som, ingredientes básicos da Realidade Virtual, que agora podem ser aliados à possibilidade de imersão. Um próximo passo na arte é a interatividade e a interação pública, também possível e tangível pela Realidade Virtual. A essência do que é VR são as três ideias juntas: imersão, interatividade e envolvimento (MORIE, 1994), conceitos totalmente explorados nas Artes Digitais.

A imersão de arte é diferente dos jogos e semelhante à imersão de um leitor lendo um livro (RYAN, 2001). De acordo com a mesma ideia, a interatividade artística tem seus precursores e ecos nas tradições literárias e artísticas pré-eletrônicas.

Esta associação de Realidade Virtual e Artes Digitais é compartilhada por outros autores. Christiane Paul apresenta a Realidade Virtual como "uma realidade que imergiu completamente seus usuários em um mundo tridimensional gerado por um computador e lhes permitiu uma interação com os objetos virtuais que compõem o mundo" (PAUL; WERNER, 2003, p. 125).

A convergência dos campos Arte Digital e Realidade Virtual pode ser perceptível no mundo acadêmico. Os tópicos da conferência de interesse em ambos os campos têm várias interseções, como é possível ver comparando os Tópicos de interesse de conferências como Artech[2] e SVR[3]. Ambas as Conferências estão interessadas em tópicos como Realidade Virtual e Aumentada; Dispositivos e interfaces de entrada; Áudio e música eletrônica; Ambiente imersivo; Percepção e Cognição e outros.

Nas Artes Digitais, várias aplicações foram desenvolvidas para atender desejos artísticos, a maioria dessas aplicações desenvolvidas para uma finalidade artística específica. Por outro lado, várias APIs e frameworks foram desenvolvidos para atender aos requisitos de programação e recursos da Realidade Virtual, a maioria das APIs desenvolvidas para um contexto específico, mas em uma linguagem de Propósito Geral.

[2] Artech é a Conferência Internacional sobre Artes Digitais. Os Tópicos de Interesse da Conferência estão disponíveis em http://2017.artech-international.org/call-for-papers/.

[3] SVR é o Simpósio sobre Realidade Virtual e Aumentada (SVR), o Primeira conferência sobre Realidade Virtual e Aumentada no Brasil. Os tópicos de interesse do simpósio estão disponíveis em http://usuarios.upf.br/~rieder/svr2017/submissions.html.

O ambiente Mosaicode pretende oferecer aos artistas uma Linguagem de Programação Visual (VPL) para o domínio de aplicação específico da Realidade Virtual, tornando-se um Idioma de Domínio (Programação) específico (DSL) neste domínio.

2.4 Computação Musical

A Computação Musical é a área da Ciência da Computação que utiliza técnicas, métodos e algoritmos computacionais para processar e gerar sons. Tem relações interdisciplinar com a música e visa solucionar os problemas musicais com auxílio computacional. Para começar estudar os recursos computacionais para processar e gerar sons, é importante aprender alguns conceitos básicos sobre som. Segundo (MILETTO et al., 2004):

> "*o som é a vibração do ar, ou seja, variações que percebemos com nossos ouvidos. Se essa pressão varia de acordo com um padrão repetitivo dizemos que o som tem uma forma de onda periódica. Se não, há um padrão perceptível no som este é chamado de ruído*" (MILETTO et al., 2004).

2.4.1 Representações gráficas do som

O som pode ser representado de forma gráfica, como "formas de ondas", que mostra as mudanças na pressão do ar ao longo do tempo – domínio temporal. A leitura dessas representação é feita da esquerda para a direta, tendo a pressão do ar mais baixa na parte inferior do gráfico (valor igual -1) e a pressão do ar mais alta no topo do gráfico (valor igual a 1) – amplitude da onda.

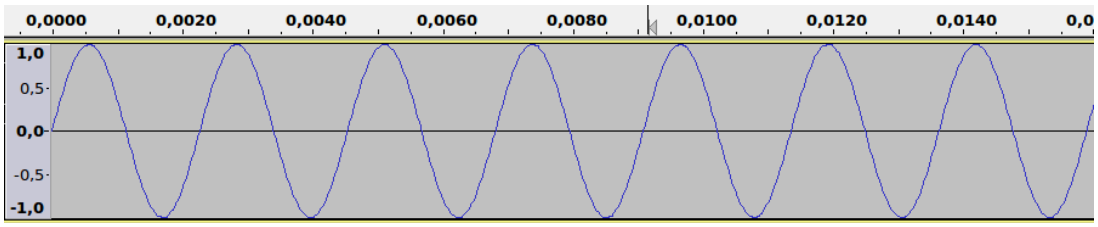

Figura 8: Forma de onda senoidal representada graficamente, no domínio temporal (Amplitude X Tempo).

O som também pode ser representado no domínio da frequência, como o espectro de frequência mostrado na Figura 9, relacionando a frequência e a amplitude do som.

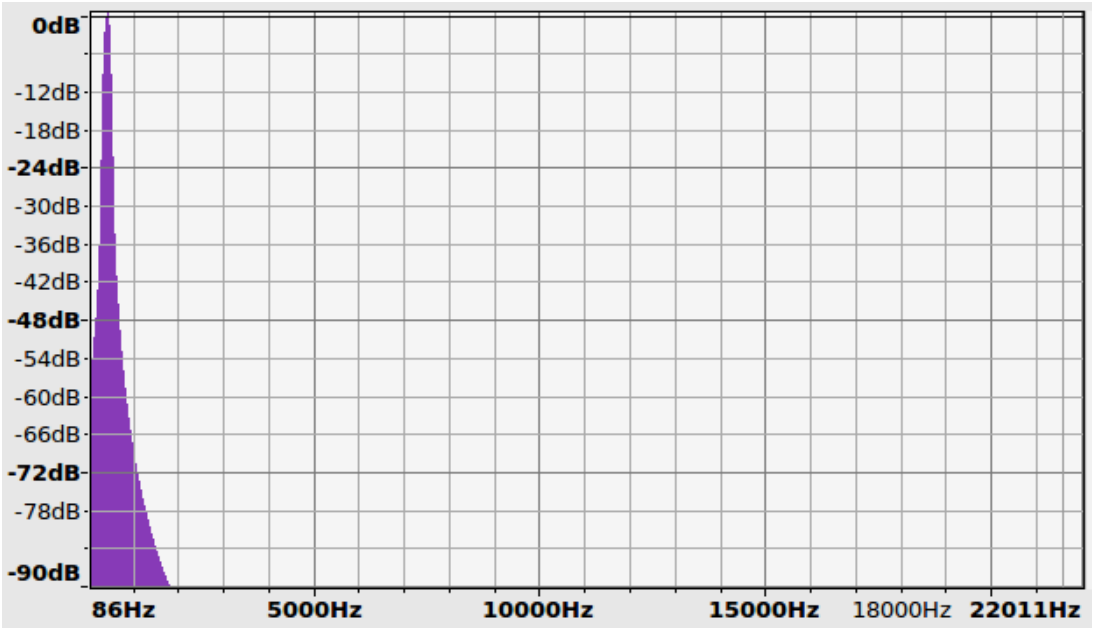

Figura 9: Forma de onda senoidal representada graficamente, no domínio da frequência – espectro de frequência (Decibéis X Frequência).

2.4.2 Elementos básicos do som

Outros conceitos básicos importantes são os três elementos básicos do som, sendo eles:

- **Altura tonal**: difere os sons agudos (maior frequência) e graves (menor frequência);
- **Volume**: determina a intensidade do som, variando a sua amplitude;
- **Timbre**: é o fator que diferencia dois instrumentos distintos. Mesmo soando com mesma altura tonal e volume, é possível diferenciá-los através do timbre.

2.4.3 Categorias sonoras

Os sons podem ser classificados pela sua forma de onda. Classificamos um som como onda simples quando quando essa onda pode ser formada por um movimento harmônico simples (Figura 10 – onda senoidal), já o som complexo (Figura 11) é resultante da combinação de ondas simples.

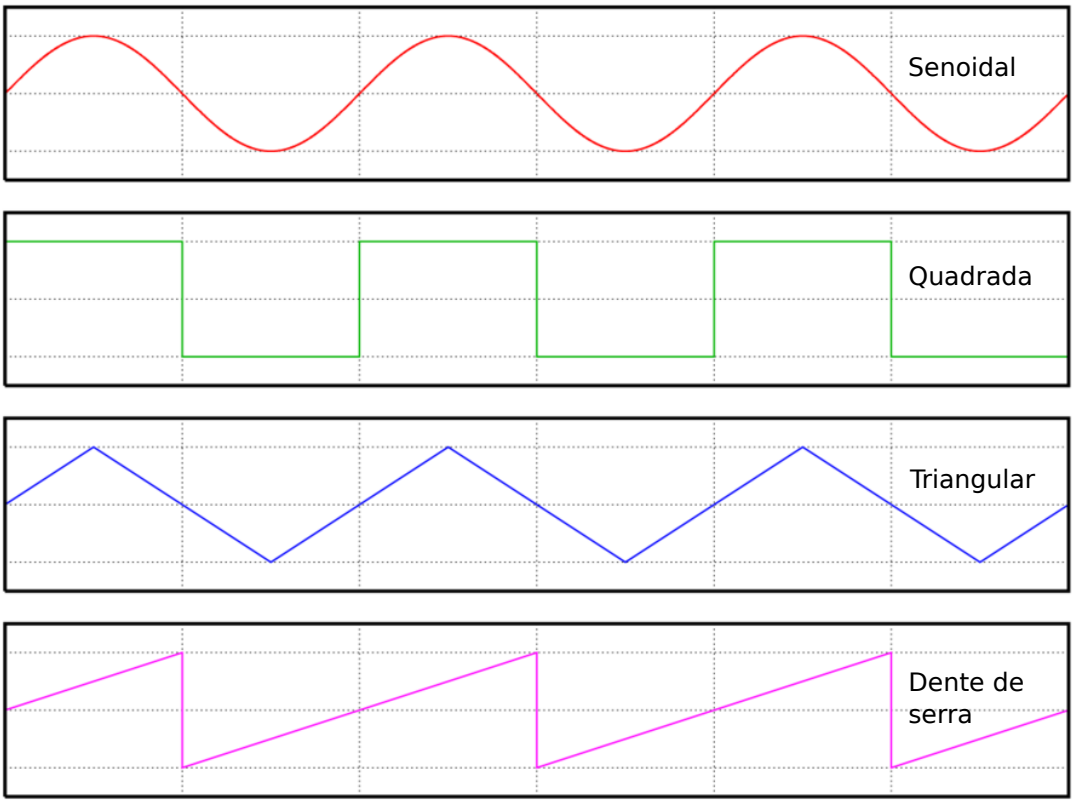

Figura 10: Formas de ondas simples[4].

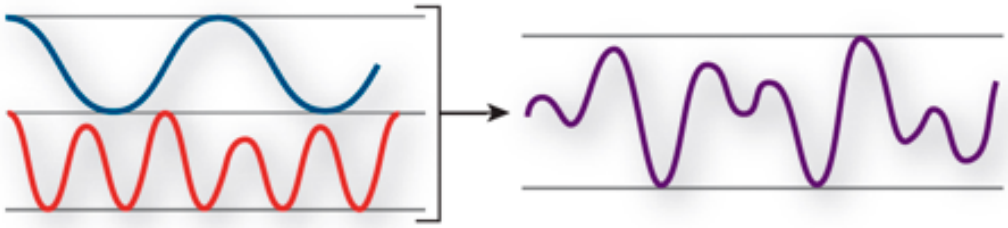

Figura 11: Duas ondas simples se combinam para criar uma onda complexa.[5].

2.4.4 Desenvolvimento de aplicações musicais

O desenvolvimento de aplicações para computadores é feito usando linguagens de programação. Com a linguagem é implementado o algoritmo que define um sequência de instruções para processar os valores de entrada (input) gerando uma saída (output), como ilustrado na Figura 12. Para a aplicação de musicais, o input e o output do algoritmo geralmente são valores que representam um sinal de áudio digital (DODGE; JERSE, 1997).

[4] Fonte: https://pt.wikipedia.org/wiki/Forma_de_onda
[5] Fonte: https://helpx.adobe.com/pt/audition/using/sound.html

Figura 12: Ilustração do conceito de algoritmo.

A Computação Musical estuda várias técnicas de processamento e síntese de áudio, que são implementadas com operações matemáticas sobre ondas, no domínio de tempo ou frequência. Por exemplo, para mudar do domínio de tempo para o domínio de frequência é utilizado a transformada de Fourier. Para implementação de filtros, efeitos, sínteses e outras técnicas manipulação de áudio também é preciso lidar com operações matemáticas.

2.5 API PortAudio

O PortAudio é uma API livre, de código aberto e multiplataforma. Essa API permite manipular interfaces de entrada e saída de áudio para desenvolvimento de aplicações de áudio. Por meio desta API é possível escolher com quais interfaces de áudio trabalhar, ler dados das interfaces de entrada e escrever nas interfaces de saída.

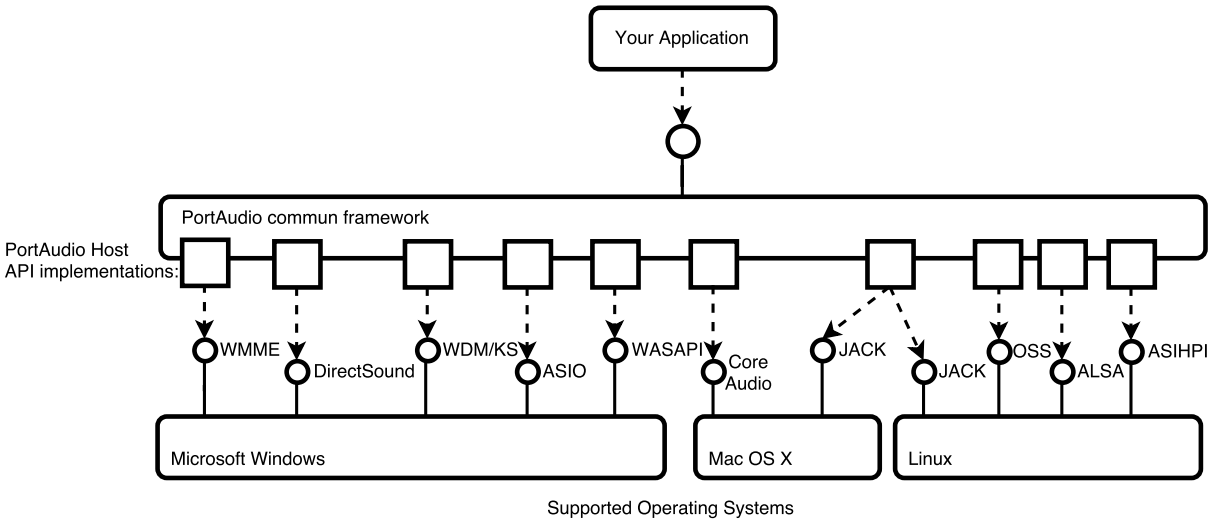

Figura 13: Visão top-down de toda a API de PortAudio (PORTAUDIO, 2017).

Como apresentado na Figura 13, o PortAudio suporta diversas API's de áudio nativas de diferentes sistemas operacionais, oferecendo uma interface comum de desenvolvimento para estas API's. Essa interface exclui a necessidade de implementação de uma aplicação para cada API nativa de cada sistema operacional, e permite ao desenvolvedor focar apenas nas chamadas do PortAudio, que são convertidas para API's nativas.

O PortAudio é uma API muito usada para desenvolvimento de aplicações de áu-

dio e está presente em algumas aplicações como: Audacity[6], CSound[7], MATLAB Signal Processing Blockset[8], Pure Data[9], Wireshark[10], VLC media player[11], VOV OGG Vorbis Decoder[12] e Wired[13].

[6] Disponível em <http://www.audacityteam.org/>
[7] Disponível em <http://csound.github.io/>
[8] Disponível em <https://www.mathworks.com>
[9] Disponível em <http://www.puredata.info>
[10] Disponível em <https://www.wireshark.org/>
[11] Disponível em <http://www.videolan.org/vlc/>
[12] Disponível em <http://www.vinjey.com/audiocodecs.html>
[13] Disponível em <http://wired.sourceforge.net/>

3 Desenvolvimento

Este Capítulo apresenta o desenvolvimento de cada etapa desde projeto, como descrito na metodologia no Capítulo 1.

3.1 Primeira etapa: Início do projeto

A primeira etapa deste trabalho consistiu em três partes: *1)* escolher a linguagem de programação para o código gerado; *2)* escolher a API de áudio para auxiliar no desenvolvimento; e *3)* definir os recursos necessário para uma VPL/DSL que possibilite artistas digitais desenvolverem aplicações de áudio para o domínio de Computação Musical e trabalharem com Sound Design. Está etapa foi desenvolvida avaliando opiniões de outros autores por meio de leitura de artigos.

Escolha da Linguagem

A maioria das API's citadas por (SCHIAVONI; GOULART; QUEIROZ, 2012) utiliza a linguagem C. Além disto, C é uma linguagem poderosa, flexível e eficiente que dispõem de recursos necessários para o desenvolvimento de aplicações de áudio (ANDERSEN, 1994), portanto, escolhemos essa linguagem de programação para o código gerado pelo *Mosaicode*.

Escolha da API de áudio

A API PortAudio foi escolhida para simplificar o desenvolvimento dos blocos no contexto musical. Por ser uma API multiplataforma, o PortAudio permite a implementação sobre APIs de áudio do sistema operacional, o que possibilita a escrita de programas para Windows, Linux (OSS/ALSA) e Mac OS X – maior portabilidade. O PortAudio utiliza a licensa MIT e oferece ainda uma API chamada PortMidi, uma biblioteca para trabalhar com o padrão MIDI (SCHIAVONI; GOULART; QUEIROZ, 2012).

Definição dos recursos

A definição dos recursos consistiu em investigar os recursos necessários para o desenvolvimento de aplicações de áudio e para trabalhar com Sound Design, satisfazendo os requisitos dos artistas digitais. Essa definição baseou-se em outras ferramentas, como Gibberish e Pure Data.

O Gibberish possui uma coleção de classes de processamento de áudio classificadas nas seguintes categorias: Osciladores, Efeitos, Filtros, Síntese de Áudio, Matemática e

Diversos (ROBERTS; WAKEFIELD; WRIGHT, 2013). Também foi investigamos os objetos nativos do Pure Data e esta ferramenta possui uma lista de objetos organizada nas seguintes categorias: Geral, Tempo, Matemática, MIDI e OSC, Diversos, Matemática de Áudio, Manipulação Geral de Áudio, Osciladores de Áudio e Tabelas e Filtros de Áudio.

Mescando as categorias levantadas em ambas ferramentas, foi definido os recursos para serem implementados no *Mosaicode* em forma de blocos. Devido a limitação de tempo para implementação dos blocos, não há como implementar todos os recursos das ferramentas Pure Data e Gibberish. Sendo assim, foram selecionados alguns dos recursos para ser implementados, desconsiderando recursos que podem ser implementados pela combinação de outros, como sínteses de áudio e envelopes. A tabela a seguir apresenta os recursos que foram implementados no *Mosaicode* em forma de blocos.

Tabela 2: Blocos implementados no *Mosaicode* para geração aplicações de áudio.

Categorias	**Blocos**
Audio Filter	Biquad filter (All-pass, Bandpass, High-pass, Low-pass), High Shelving, Low Shelving e Parametric Equalizer.
Audio Math	Addition, Subtraction, Division e Multiplication.
General	Audio Devices.
Output	Record e Speaker.
Sound Sources	Oscillators, White Noise, Microphone e Playback.

3.2 Segunda etapa: Construção da biblioteca *libmosaic-sound*

Neste trabalho usamos a linguagem de programação C e a API PortAudio, que oferecem recursos necessário para manipular interfaces de entrada e saída de áudio. Esta API não oferece implementações dos recursos que definimos na etapa anterior. Foi então criada uma biblioteca para disponibilizar esses recursos, pensando também em uma estrutura que não exija muito esforço de programação e seja adequada para usar na construção do conjunto de blocos, tornando mais fácil o desenvolvimento do mesmo por meio de reuso de código. A API também foi pensada de forma que a maioria dos blocos não dependessem da API PortAudio, sendo que apenas o bloco para listar as interfaces de entrada e saída e os blocos para manipular essas interfaces que dependem dessa API. Também foi utilizada a API *libsoundfile* para implementar acesso aos arquivos de mídia, em diversos formatos. O bloco *Playback* utiliza essa biblioteca para ler uma arquivo de mídia e o bloco *Record* utiliza para gravar.

Para cada recurso foi implementado um Tipo Abstrato de Dados (TAD) seguindo o mesmo padrão, como apresentado a seguir:

- **input**: Dados de entradas a serem processados, podendo ter mais de um campo *input*;

- **output**: dados processados, podendo ter mais de um campo *output*;
- **framesPerBuffer**: tamanho do buffer a ser processado em cada interação;
- **process**: função que processa os dados de entradas (input) e guarda na saída (output), caso o TAD tenha saída;
- **create**: função de criação/inicialização da variável do tipo TAD.
- **outros**: cada recursos tem as suas propriedades e valores a serem guardados para utilizar no processamento, então são criados campos para guardar esses valores.

Outro detalhe de implementação é a definição de um *namespace* para a biblioteca, usando o prefixo *mosaicsound_*, adicionado nas funções, tipos e definições da biblioteca, para garantir que não haja conflito com palavras reservadas de outras bibliotecas.

A biblioteca pode ser usada de forma estática, utilizando o arquivo *libmosaic-sound.a* gerado na compilação, ou de forma dinâmica, incluindo a biblioteca instalada no sistema. Os detalhes de como compilar, instalar e executar o código estão descritos no arquivo *README.md* no repositório da biblioteca no *GitHub*[1].

O Código-fonte 1 mostra o TAD que abstrai a implementação da captura de dados de um microfone (dispositivo de entrada):

```
1  #ifndef MOSAICSOUND_MIC_H
2  #define MOSAICSOUND_MIC_H
3     typedef struct{
4         float *output;
5         int framesPerBuffer;
6         void (*process)(void *self, float *);
7     }mosaicsound_mic_t;
8
9     mosaicsound_mic_t* mosaicsound_create_mic(int framesPerBuffer);
10    void mosaicsound_mic_process();
11 #endif /* mic.h */
```

Código-fonte 1: Definição do TAD *mosaicsound_mic_t* que abstrai a implementação de uma microfone.

Também há funções que devem ser definidas e funções que deve ser chamadas no uso da biblioteca *libmosaic-sound*. Estas funções estão descritas a seguir:

- **mosaicsound_callback**: função de callback que é chamada a cada interação para processar os valores de entradas. O usuário deve definir essa função;

[1] Disponível em <https://github.com/Mosaicode/libmosaic-sound/blob/master/README.md>

- **mosaicode_finished**: o usuário também deve definir essa função, que será chamada pela biblioteca quando a mesma terminar a execução;
- **mosaicode_initialize**: essa é função que o usuário deve chamar para inicializar a biblioteca;
- **mosaicode_terminate**: essa é a função que usuário deve chamar para finalizar a biblioteca.

O Código-fonte 2 mostrar um exemplo de uso da biblioteca *libmosaic-sound*:

```
#include <mosaic-sound.h>
#include <portaudio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NUM_SECONDS 12
#define SAMPLE_RATE 44100
#define FRAMES_PER_BUFFER 256

/* CODE PART: Declaration */
mosaicsound_mic_t *mic;
mosaicsound_record_t *rec;
mosaicsound_speaker_t *speaker;
/* ------------------------------------------------ */

/* Callback function */
static int mosaicsound_callback(const void *inputBuffer,
            void *outputBuffer,
            unsigned long framesPerBuffer,
            const PaStreamCallbackTimeInfo *timeInfo,
            PaStreamCallbackFlags statusFlags,
            void *userData) {
  float *in = (float *)inputBuffer;
  float *out = (float *)outputBuffer;

  (void)timeInfo; /* Prevent unused variable warnings. */
  (void)statusFlags;
  (void)userData;

  /* CODE PART: Execution */
  mic->process(mic, in);
  rec->process(rec);
  speaker->process(speaker, out);
  /* ------------------------------------------------ */
  return paContinue;
```

```c
}

/* This routine is called by mosaic-sound when
   mosaicsound_callback is done. */
static void mosaicsound_finished(void *data) {
  printf("Stream Completed!\n");
}

int main(int argc, char *argv[]) {
  /* CODE PART: Setup */
  mic = mosaicsound_create_mic(FRAMES_PER_BUFFER);
  rec = mosaicsound_create_record("examples/record_mic.wav",
        FRAMES_PER_BUFFER, 10, 44100);
  speaker = create_speaker(FRAMES_PER_BUFFER);
  /* ------------------------------------------------ */

  /* CODE PART: Connections */
  rec->input = mic->output;
  speaker->input = mic->output;
  /* ------------------------------------------------ */

  /* Library startup function */
  void *stream = mosaicsound_initialize(SAMPLE_RATE,
        FRAMES_PER_BUFFER);

  printf("Recording until the Enter key is pressed.\n");
  getchar();

  /* Function to finish the library */
  mosaicsound_terminate(stream);

  return 0;
}
```

Código-fonte 2: Exemplo de uso da biblioteca *libmosaic-sound*.

O Código-fonte 2 é um exemplo de uso da biblioteca *libmosaic-sound*, que implementa a gravação (record) do áudio capturado por um microfone (mic) e também direciona esse áudio capturado para o alto-falante (speaker). O fluxo de execução do exemplo é ilustrado a seguir na Figura 14.

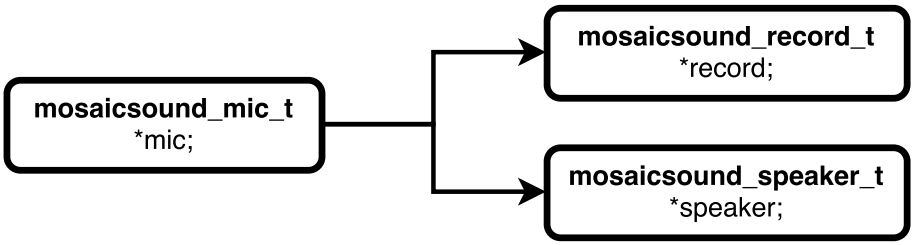

Figura 14: Ilustra o fluxo de execução do exemplo.

No Código-fonte 2 foram destacados, com cometários, partes de código chamadas de *declaration*, *execution*, *setup* e *connections*. Essas partes de código são os lugarem onde os usuários definirão a implementação da aplicação de áudio, sendo que:

- **declaration**: é a parte de código onde deve-se declarar as várias a serem usadas. Essas variáveis tem como tipo os Tipos Abstratos de Dados implementados pela biblioteca *libmosaic-sound*;
- **execution**: é a parte de código que define a ordem de chamada da função *process* de cada variável declarada na parte de código *declaration*. Essa parte está incluída dentro da função de callback do *Mosaicode*;
- **setup**: é a parte de código para inicializar as variáveis usando suas respectivas funções *create* e definir seus valores de seus campos, caso tenham;
- **connections**: é a parte de código para definir as conexões entre as variáveis. Essa conexão é feita atribuindo ao campo *input* de uma variável o valor do campo *output* de outra variável.

Para a identificação das partes de código, foram criados uma série de exemplos, usando todos os Tipos Abstratos de Dados definidos. Observando os exemplos foi possível identificar as características de cada parte do código para então definir as partes de código listadas anteriormente. O código com a implementação da biblioteca *libmosaic-sound* e dos exemplos estão disponíveis no GitHub[2].

3.3 Terceira etapa: Construção dos Blocos no *Mosaicode*

Está etapa consistiu na implementação dos blocos no *Mosaicode*, abstraindo os recursos que foram definidos na primeira etapa deste projeto. Como esses recursos foram implementado pela biblioteca *libmosaic-sound*, foi feito um reuso de código utilizando a mesma.

[2] Disponível em <https://github.com/Mosaicode/libmosaic-sound>

O primeiro passo foi definir o padrão de código (Code Template) para o *Mosaicode*. Para isso, foi preciso observar as partes do código que sempre são iguais, independente da implementação, e as partes incomuns, geradas com base nos diagramas do *Mosaicode*. As partes de que são geradas são as partes de código citado na construção da *libmosaic-sound* – *declaration*, *execution*, *setup* e *connections*. O restante do código sempre será igual em todas as implementações, então, é colocado no padrão de código.

O segundo passo foi criar os tipos de portas de entradas/saídas para os blocos, que neste caso foi apenas uma, a porta do tipo *sound* (som). Com o padrão de código e a porta criada, o último passo foi a implementação dos blocos no *Mosaicode*.

Padrão de código

A Figura 15 mostra o Editor de Padrão de Código do *Mosaicode*, que permite criar e editar Padrões de Códigos.

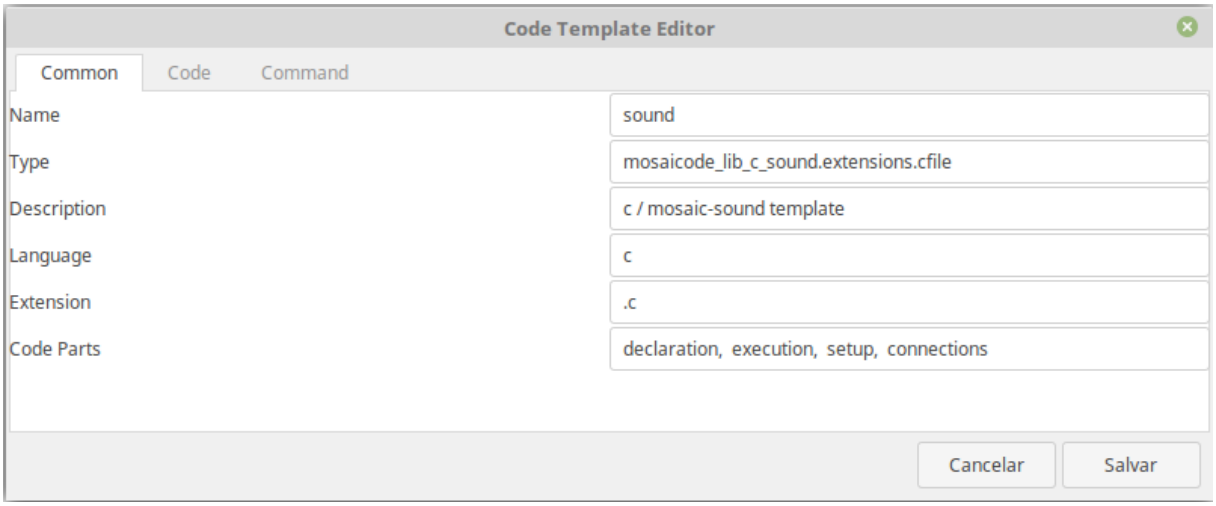

Figura 15: Editor de Padrão de Código

Para a criação do Padrão foi definido as suas propriedades como *nome*, *tipo* e partes de código. Também foi definido o Código do Padrão de Código (Código-fonte 3) e o comando de compilação (Código-fonte 4).

```
1 #include <mosaic-sound.h>
2 #include <portaudio.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6
7 #define NUM_SECONDS 12
8 #define SAMPLE_RATE 44100
9 #define FRAMES_PER_BUFFER 256
```

```
10
11 $code[declaration]$
12
13 static int mosaicsound_callback(const void *inputBuffer, void *
       outputBuffer,
14                                  unsigned long framesPerBuffer,
15                                  const PaStreamCallbackTimeInfo *
       timeInfo,
16                                  PaStreamCallbackFlags statusFlags,
17                                  void *userData) {
18   float *in = (float *)inputBuffer;
19   float *out = (float *)outputBuffer;
20
21   (void)timeInfo; /* Prevent unused variable warnings. */
22   (void)statusFlags;
23   (void)userData;
24   (void)in;
25   (void)out;
26
27   $code[execution]$
28
29   return paContinue;
30 }
31
32 static void mosaicsound_finished(void *data) { printf("Stream
       Completed!"); }
33
34 int main(int argc, char *argv[]) {
35   $code[setup]$
36   $code[connections]$
37
38   void *stream = mosaicsound_initialize(SAMPLE_RATE,
       FRAMES_PER_BUFFER);
39
40   printf("Playing until the Enter key is pressed.");
41   getchar();
42
43   mosaicsound_terminate(stream);
44
45   return 0;
46 }
```

Código-fonte 3: Padrão de código implementado para a biblioteca de blocos construída neste projeto.

Observe que nas partes de código foram colocados *wildcards* (como *$code[declaration]$*) indicando ao gerador o local de cada parte, para o gerador substituir os *wildcards* por código.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lib/;
export PKG_CONFIG_PATH=/lib/pkgconfig/;
gcc $filename$$extension$  -I/usr/include/mosaic/mosaic-sound -
   lmosaic-sound -o $filename$ -g -Wall -lportaudio -lm `pkg-
   config --libs sndfile`
LD_LIBRARY_PATH=/lib/ $dir_name$./$filename$
```

Código-fonte 4: Comando de compilação do padrão de código

Porta

A Figura 16 mostra o Editor de Portas do *Mosaicode*, que permite criar e editar Portas. Para criar a porta foi definido as suas propriedades como *tipo*, *linguagem*, *rótulo* e *cor*.

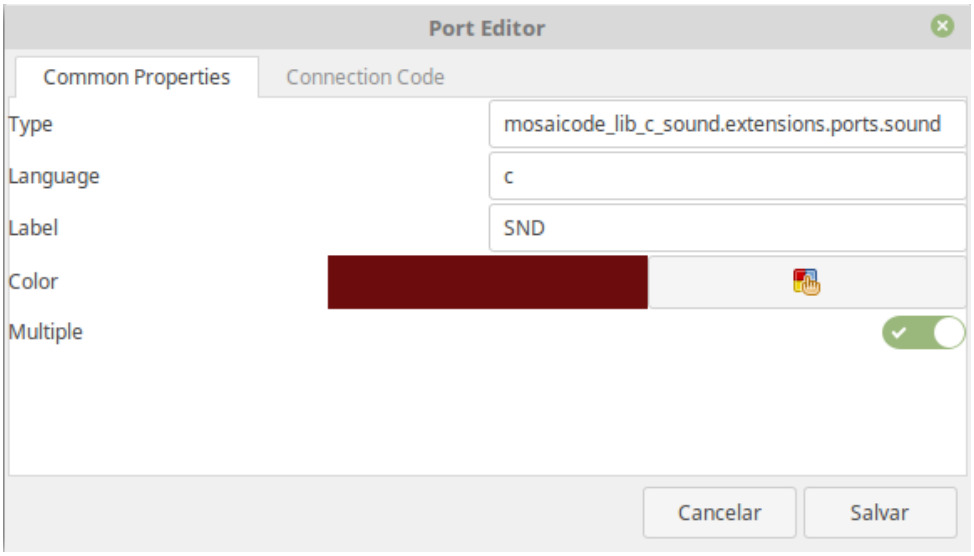

Figura 16: Editor de Portas

Também foi definido o código da conexão (Código-fonte 5), que simplesmente atribui uma porta de saída (output) a uma porta de entrada (input). O *Mosaicode* gera essas conexões automaticamente interpretando o diagrama de blocos.

```
$input$ = $output$;
```

Código-fonte 5: Código que define a conexão entre Portas

Blocos

O *Mosaicode* também tem o Editor de Blocos, que permite criar e editar Blocos. Para criar um Bloco foi definido suas propriedades comuns, como apresentado na Figura 17, definindo o *rótulo*, a *linguagem*, o *grupo* e outras propriedades.

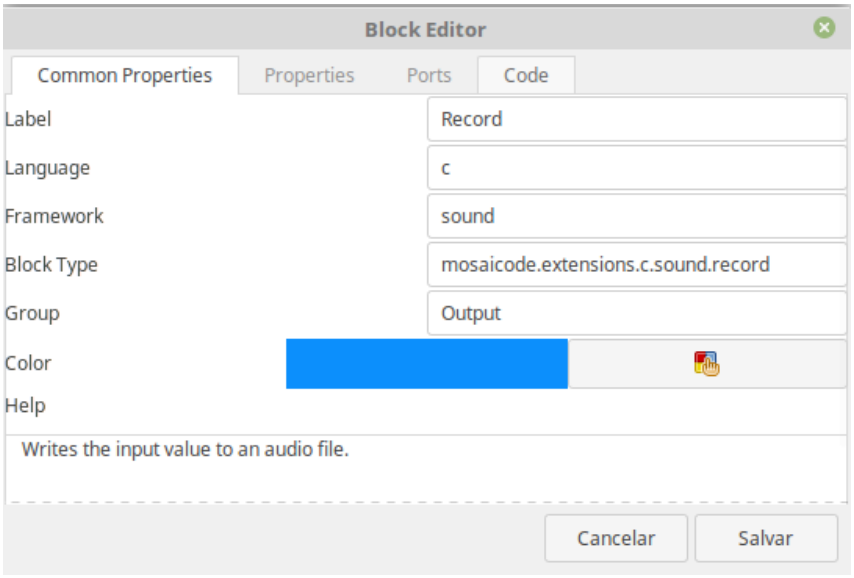

Figura 17: Editor de Blocos: propriedades comuns

Também foi definido as propriedades estáticas, que pode ser configuradas antes da execução do código. Na Figura 18, mostra a propriedades do Bloco *Record*, que permite ao usuário escolher o nome do arquivo de saída.

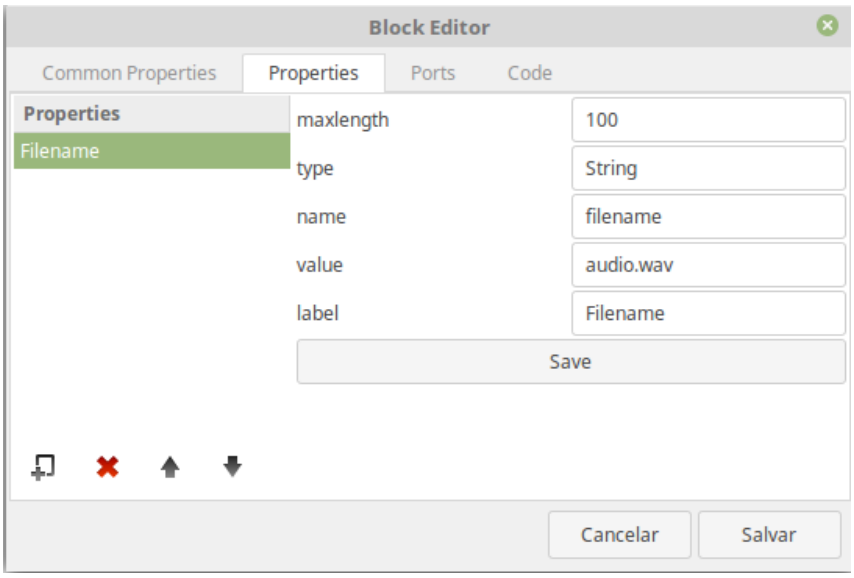

Figura 18: Editor de Blocos: propriedades

Na Figura 19, é criado uma porta de entrada do tipo *Sound*, que recebe o sinal de áudio a ser gravado no arquivo de áudio.

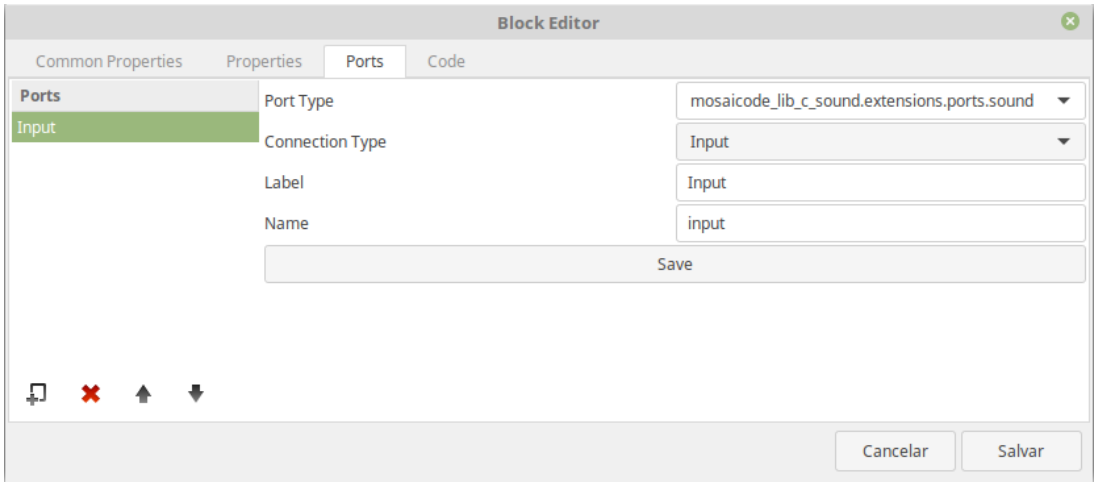

Figura 19: Editor de Blocos: portas

A Figura 20 mostra um exemplo que define a Parte de Código *Setup* do Bloco *Record*. Neste exemplo é usado o *wildcard* para definir o *id* do Bloco e o *wildcard* para definir o nome do arquivo de saída (propriedade do bloco).

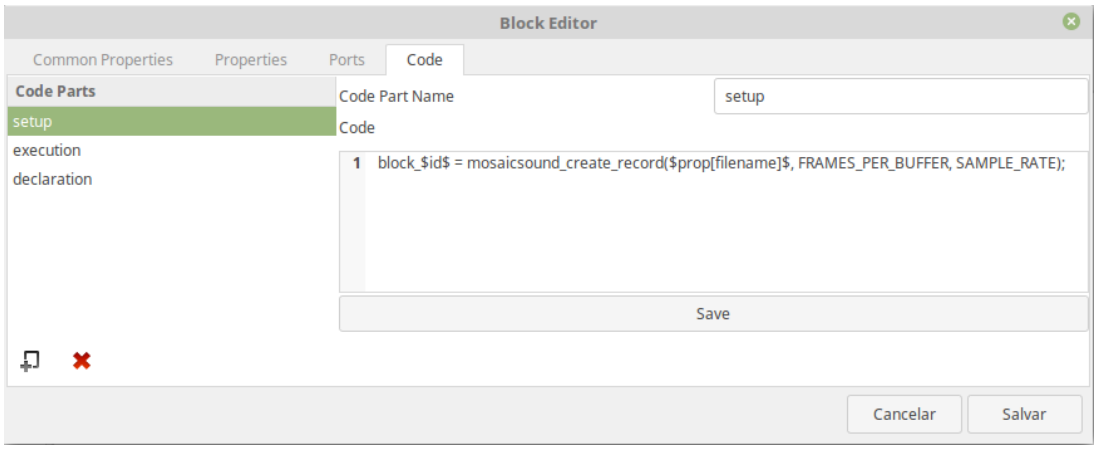

Figura 20: Editor de Blocos: código

O código com a implementação do código da biblioteca de blocos – conjunto de blocos, portas e padrão de código – está disponível no GitHub[3].

[3] Disponível em <https://github.com/Mosaicode/mosaicode-c-sound>

4 Resultados

Este capítulo apresenta os resultados obtidos por este projeto.

4.1 Biblioteca *libmosaic-sound*

A primeira contribuição deste trabalho é uma biblioteca para o desenvolvimento de aplicações musicais, intitulada *libmosaic-sound*. Essa biblioteca foi desenvolvida na linguagem de programação C baseado na API PortAudio e na biblioteca *libsndfile*, implementando recursos que permitem artistas digitais desenvolverem aplicações de áudio e trabalharem com Sound Design, exigindo menos esforço de programação. O desenvolvimento da biblioteca *libmosaic-sound* facilitou a construção do conjunto de blocos, oferecendo uma estrutura adequada e reduzindo o esforço necessário para a construção dos blocos.

Essa biblioteca também pode ser usada sem o *Mosaicode*, como as bibliotecas de programação para a linguagem C. Ela traz implementado recursos que a API PortAudio não oferece e usa uma estrutura que o reduz o esforço para desenvolver aplicações de áudio. Junto com a biblioteca há exemplos usando todos os Tipo Abstratos de Dados definidos, que pode ajudar o usuário a compreender o uso de cada um, além de ter ajudado para o desenvolvimento do Padrão de Código no *Mosaicode*.

4.2 Conjunto de blocos para o ambiente Mosaicode

Este trabalho resultou em um conjunto de blocos para aplicação de áudio no *Mosaicode* que define uma Linguagem de Programação Visual. Com essa VPL simplificamos desenvolvimento de aplicações para o domínio de Computação Musical, permitindo gerar aplicações de áudio e trabalhar com Sound Design arrastando e conectando blocos, sendo assim, aumentado a acessibilidade aos artistas digitais.

A VPL construída traz todos os recursos oferecido pela biblioteca *libmosaic-sound*, abrangendo fontes sonoras em forma de onda simples e capacitando a construção de sínteses de áudio, efeitos sonoros e envelopes, para a geração de sons complexos. É possível implementar os exemplos clássicos de sínteses como AM, FM, Aditiva e Subtrativas e praticar as demais técnicas de Computação Musical, sem preocupar com sintaxe de código e decorar comandos, apenas arrastando e conectando blocos. O usuário também tem a opção de obter o código-fonte da aplicação gerado a partir do diagrama, tendo total liberdade pra modificar, estudar e usar da forma que desejar.

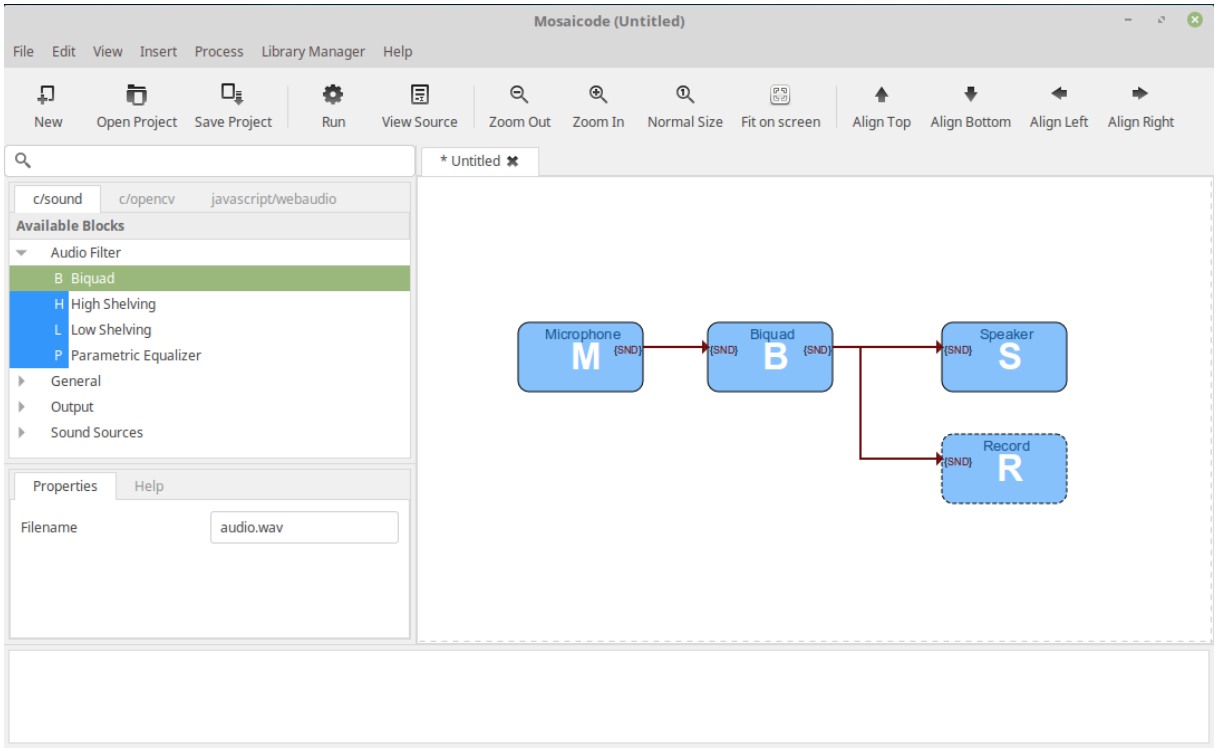

Figura 21: Exemplo de diagrama no *Mosaicode* usando o conjunto de blocos construídos neste trabalho.

A Figura 21 apresenta um exemplo usando o conjunto de blocos construídos neste trabalho. Nesse exemplo temos a aplicação do filtro *lowpass* (Biquad) em um sinal de áudio capturado por um microfone (Microphone), a saída do filtro é direcionada para o alto-falante (speaker) e gravado em um arquivo de áudio (Record). O Código-fonte 6 apresenta o código gerado a partir do diagrama desse exemplo.

```
1  #include <mosaic-sound.h>
2  #include <portaudio.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6
7  #define NUM_SECONDS 12
8  #define SAMPLE_RATE 44100
9  #define FRAMES_PER_BUFFER 256
10
11 mosaicsound_mic_t *block_1;
12 mosaicsound_biquad_t *block_2;
13 mosaicsound_record_t *block_3;
14 mosaicsound_speaker_t *block_4;
15
16 static int mosaicsound_callback(const void *inputBuffer, void *
       outputBuffer,
17                                 unsigned long framesPerBuffer,
```

```
                                    const PaStreamCallbackTimeInfo *
    timeInfo,
                                    PaStreamCallbackFlags statusFlags,
                                    void *userData) {
  float *in = (float *)inputBuffer;
  float *out = (float *)outputBuffer;

  (void)timeInfo; /* Prevent unused variable warnings. */
  (void)statusFlags;
  (void)userData;

  block_1->process(block_1, in);
  block_2->process(block_2);
  block_3->process(block_3);
  block_4->process(block_4, out);

  return paContinue;
}

static void mosaicsound_finished(void *data) { printf("Stream
    Completed!\n"); }

int main(int argc, char *argv[]) {
  block_1 = mosaicsound_create_mic(FRAMES_PER_BUFFER);
  block_2 = mosaicsound_create_biquad(1, 2, FRAMES_PER_BUFFER);
  block_2->sampleRate = SAMPLE_RATE;
  block_2->cutOff = 3000.0;
  block_2->slope = 0.1;
  block_3 = mosaicsound_create_record("examples/record_mic_lowpass
    .wav", FRAMES_PER_BUFFER, 44100);
  block_4 = mosaicsound_create_speaker(FRAMES_PER_BUFFER);

  block_2->input = block_1->output;
  block_3->input = block_2->output;
  block_4->input = block_2->output;

  void *stream = mosaicsound_initialize(SAMPLE_RATE,
    FRAMES_PER_BUFFER);

  printf("Recording until the Enter key is pressed.\n");
  getchar();

  mosaicsound_terminate(stream);
  return 0;
}
```

Código-fonte 6: Código gerado pelo *Mosaicode* a partir do diagrama da Figura 21.

No Código-fonte 6, o nome das variáveis não é muito intuitivo, tendo como prefixo a palavra "block_" seguido de uma número de identificação do block (id). Esse nome dificulta a interpretação do código, porém já está sendo implementado uma otimização na geração de código para gerar um código mais legível.

4.3 Contribuições científicas

Este trabalho também resultou em um conjunto de blocos para aplicações de áudio no *Mosaicode*, que gera código na linguagem Javascript baseado na API WebAudio[1]. Além da implementação desse conjunto, houve a publicação do artigo "Web Audio application development with Mosaicode" (SCHIAVONI; GONÇALVES; GOMES, 2017), publicado nos Anais do Simpósio Brasileiro de Computação Musical em setembro de 2017. O artigo discute o desenvolvimento de aplicação Web Audio com o *Mosaicode*, utilizando o conjuntos de blocos construído.

Também foi publicado o artigo "Programação musical para a web com o *Mosaicode*" (SCHIAVONI; GONÇALVES, 2017b), que apresenta o ambiente do *Mosaicode* para a criação de aplicações musicais voltado para Web, publicado nos Anais da Associação Nacional de Pesquisa e Pós-Graduação em Música em setembro de 2017. Esse conjunto de blocos em Javascript/WebAudio foi construído como protótipo funcional para este projeto e acabou colaborando para *Mosaicode* com esse novo conjunto de blocos, permitindo gerar aplicação web para o domínio de Computação Musical.

Outra publicação resultado deste trabalho foi do artigo "From Virtual Reality to Digital Arts with *Mosaicode*" (SCHIAVONI; GONÇALVES, 2017a), publicado nos Anais do *Symposium on Virtual and Augmented Reality* em novembro de 2017. O artigo discute o ambiente do *Mosaicode* para gerar aplicações para os domínios específicos da Arte Digital, apresentando uma taxonomia para desenvolvimento de projetos voltados para Arte Digital.

Esse projeto também colaborou para o desenvolvimento do *Mosaicode*. Foi feito o "Teste de Usabilidade do Sistema *Mosaicode*" (SCHIAVONI; GONÇALVES, 2017c), publicado como artigo nos Anais do III Workshop de Iniciação Científica em Sistemas de Informação/XII Simpósio Brasileiro de Sistemas de Informação em Junho de 2017. Esse teste avalia a usabilidade do sistema, apresentando sugestões para melhorar a usabilidade do sistema e identifica problemas no sistema a serem solucionados. A maioria das sugestões já foram implementadas e os problemas identificados já foram corrigidos.

Também foi discutido sobre o desenvolvimento de treinamentos de percepção musical utilizando a ferramenta *Mosaicode*. Essa discussão foi realizada no artigo "Percepção

[1] Conjunto de blocos disponível em <https://github.com/Mosaicode/mosaicode-javascript-webaudio>

Musical apoiada por computador: Além das alturas e tempos" (GONÇALVES; SCHIAVONI, 2017), publicado no XIII Congresso de Produção Científica e Acadêmica da UFSJ.

5 Conclusão

Este trabalho propôs a construção de um conjunto de blocos para aplicação de áudio, dentro do ambiente de programação visual *Mosaicode*. Essa construção dentro do *Mosaicode* permite a geração de código-fonte de aplicações a partir de diagramas compostos por Blocos e Conexões, tornando o Sound Design mais acessível aos artistas digitais.

A primeira etapa deste projeto exigiu muita leitura e pesquisa, com intuito de avaliar a opinião de autores para definir a linguagem de programação e API de áudio adequada para o projeto, oferecendo os recursos necessários e tornando o desenvolvimento mais fácil. Esse esforço de pesquisa e leitura, também foi necessário para definir os recursos necessários para um DSL/VPL que suprem as necessidades dos artistas digitais no desenvolvimento de aplicações para o domínio de Computação Musical. Além disso, a investigação das ferramentas Pure Data e da biblioteca Gibberish contribuiu muito para definir esses recursos.

Na segunda etapa foi apresentado a construção de uma biblioteca, que foi utilizada para auxiliar na construção dos blocos do *Mosaicode* também para facilitar o desenvolvimento de aplicações de áudio. A estrutura da biblioteca remete a manipulação de blocos e conexões do *Mosaicode*, para gerar aplicações, como se cada TAD fosse os blocos e a atribuições entre *outputs* e *inputs* fossem as conexões. Essa estrutura também reduziu bastante a quantidade de linha que os usuários precisam escrever para gerar uma aplicação de áudio, comparado com a API PortAudio, principalmente porque a API fornece apenas a manipulação das interfaces de entradas e saídas exigindo ao usuário implementar o processamento dos dados lidos/escritos pelas interfaces, para gerar a aplicação que deseja.

Para o desenvolvimento da segunda etapa houve dificuldade para implementação das técnicas de Computação Musical e para a construção de uma biblioteca. Além da contribuição com a biblioteca, houve a contribuição para o crescimento próprio ao adquirir todos esses conhecimentos e colocar em prática.

Na terceira etapa foi apresentado a construção de um conjunto de blocos no *Mosaicode* para trabalhar com Sound Design. Esse conjunto de blocos baseou-se na biblioteca *libmosaic-sound*, que facilitou bastante o desenvolvimento dos blocos por meio do reuso de código e oferecendo a geração de um código-fonte menor, abstraindo os recursos em TAD's, tornando mais fácil de entender e associar cada trecho aos componentes do diagrama. Essa construção dos blocos resultou em uma VPL para o domínio de Computação Digital e, por ter sido desenvolvida no *Mosaicode*, permite a geração do código-fonte

para ser estudado, modificado e usado da forma que quiser. Além disso, contribui para o *Mosaicode* com mais um conjunto de blocos para trabalhar na área de Computação Musical.

O desafio da terceira etapa foi implementação do padrão de código para o *Mosaicode*. Primeiramente, foi construído vários exemplos de códigos usando a *libmosaic-sound*, esses exemplos foram estudados afim de entender cada parte de código e definir quais partes são fixas no padrão de código e quais partes são geradas pelo *Mosaicode*. A construção da biblioteca de blocos acrescentou conhecimentos de como funciona a geração de código, fazendo entender como é possível fazer com que uma algoritmo interprete um diagrama e gere outro algoritmo.

Esse projeto também contribuiu para o desenvolvimento do *Mosaicode*, que tem passado por uma refatoração de código buscando melhorar a sua estrutura para tornar mais fácil a sua manutenção e extensão. Também contribuiu com um Teste de Usabilidade, melhorando a usabilidade do sistema por meio de correções de problemas e implementação de novas funcionalidades. Esses trabalhos, com objetivo de melhorar o *Mosaicode*, também contribuíram com publicações de artigos, que foram oportunidades para divulgar a ferramenta, compartilhando os nossos trabalhos para profissionais em eventos importantes para as áreas da Ciência da Computação e Música no Brasil. Além disso, foram momentos de troca de conhecimentos e experiências únicas.

5.1 Trabalho futuros

Como trabalhos futuros pretende-se rever a lista de recursos definidas buscando oferecer novos recursos, ampliando a biblioteca e o conjunto de blocos para aplicação de áudio. Também pretende-se ligar este projeto a outros projetos desenvolvidos pela equipe de desenvolvimento do *Mosaicode*. Existem trabalhos em desenvolvimento para construir conjuntos de blocos para oferecer recursos de Processamento Digital de Imagem, Visão Computacional, Inteligência Artificial, Redes de Computadores e Realidade Virtual. A intenção é que todos essas trabalhos se conectem oferecendo recursos para gerar aplicações para os domínios específicos da Arte Digital.

Referências

ANDERSEN, L. O. *Program analysis and specialization for the C programming language*. Tese (Doutorado) — University of Cophenhagen, 1994. Citado na página 29.

BRINKMANN, P. et al. Embedding pure data with libpd. In: CITESEER. *Proceedings of the Pure Data Convention*. [S.l.], 2011. Citado na página 20.

CAMURRI, A. et al. Eyesweb: Toward gesture and affect recognition in interactive dance and music systems. *Computer Music Journal*, MIT Press, 2000. Citado na página 21.

DEURSEN, A. V.; KLINT, P. Domain-specific language design requires feature descriptions. *CIT. Journal of computing and information technology*, SRCE-Sveučilišni računski centar, 2002. Citado na página 12.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, ACM, 2000. Citado na página 22.

DODGE, C.; JERSE, T. A. *Computer Music: Synthesis, Composition and Performance*. 2nd. ed. [S.l.]: Macmillan Library Reference, 1997. Citado na página 26.

GOMES, A.; HENRIQUES, J.; MENDES, A. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. *Educação, Formação & Tecnologias-ISSN 1646-933X*, 2008. Citado na página 12.

GONÇALVES, L. L.; SCHIAVONI, F. L. Percepção musical apoiada por computador: além das alturas e tempos. In: *Anais do XV Congresso de Produção Científica da Universidade Federal de São João Del Rei*. São João del-Rei - MG - Brazil: [s.n.], 2017. v. 1, n. 1, p. 1–8. Citado na página 44.

GRAU, O. *Virtual Art: from illusion to immersion*. [S.l.]: MIT press, 2003. Citado na página 23.

GRONBACK, R. C. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. [S.l.]: Addison-Wesley, 2009. Citado na página 12.

GUDWIN, R. R. Linguagens de programação. *Campinas: DCA/FEEC/UNICAMP*, 1997. Citado na página 22.

HAEBERLI, P. E. Conman: A visual programming language for interactive graphics. *SIGGRAPH Comput. Graph.*, ACM, 1988. Citado na página 12.

HILS, D. D. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, Elsevier, 1992. Citado na página 12.

IAZZETTA, F. A música, o corpo e as máquinas. *Revista Opus*, p. 27–44, 1997. Citado na página 12.

JIANG, Z.; ALLRED, R.; HOCHSCHILD, J. *Multi-rate digital filter for audio sample-rate conversion*. [S.l.]: Google Patents, 2002. Citado na página 19.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, ACM, 2005. Citado 2 vezes nas páginas 12 e 22.

MILETTO, E. M. et al. Introdução à computação musical. In: *IV Congresso Brasileiro de Computação*. [S.l.: s.n.], 2004. Citado 2 vezes nas páginas 12 e 24.

MORIE, J. F. Inspiring the future: Merging mass communication, art, entertainment and virtual environments. *SIGGRAPH Comput. Graph.*, ACM, 1994. Citado na página 23.

MYERS, B. A. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, Elsevier, 1990. Citado na página 22.

PAUL, C.; WERNER, C. *Digital art*. [S.l.]: Thames & Hudson London, 2003. Citado na página 23.

PORTAUDIO, C. *Portable Cross-platform Audio I/O*. 2017. Disponível em: <http://portaudio.com/>. Citado 2 vezes nas páginas 7 e 27.

PUCKETTE, M. S. et al. Pure data. In: *ICMC*. [S.l.: s.n.], 1997. Citado 2 vezes nas páginas 19 e 20.

PULLI, K. et al. Real-time computer vision with opencv. *Communications of the ACM*, ACM, 2012. Citado na página 18.

REAS, C.; FRY, B. *Processing: a programming handbook for visual designers and artists*. [S.l.]: Mit Press, 2007. Citado na página 20.

RICHARDSON, L. *Beautiful Soup Documentation*. 2015. Citado na página 15.

ROBERTS, C.; WAKEFIELD, G.; WRIGHT, M. The web browser as synthesizer and interface. In: CITESEER. *NIME*. [S.l.], 2013. Citado 3 vezes nas páginas 18, 19 e 30.

RYAN, M.-L. *Narrative As Virtual Reality: Immersion and Interactivity in Literature and Electronic Media*. [S.l.]: Johns Hopkins University Press, 2001. Citado na página 23.

SCHIAVONI, F. L.; GONÇALVES, L. L. From virtual reality to digital arts with mosaicode. In: *2017 19th Symposium on Virtual and Augmented Reality (SVR)*. Curitiba - PR - Brazil: [s.n.], 2017. p. 200–206. Citado na página 43.

SCHIAVONI, F. L.; GONÇALVES, L. L. Programação musical para a web com o mosaicode. In: *Anais do XXVII Congresso da Associação Nacional de Pesquisa e Pós-Graduação em Música*. Campinas - SP - Brazil: [s.n.], 2017. p. 1–6. Citado na página 43.

SCHIAVONI, F. L.; GONÇALVES, L. L. Teste de usabilidade do sistema mosaicode. In: *Anais [do] IV Workshop de Iniciação Científica em Sistemas de Informação (WICSI)*. Lavras - MG - Brazil: [s.n.], 2017. p. 5–8. Citado na página 43.

SCHIAVONI, F. L.; GONÇALVES, L. L.; GOMES, A. L. N. Web audio application development with mosaicode. In: *Proceedings of the 16th Brazilian Symposium on Computer Music*. São Paulo - SP - Brazil: [s.n.], 2017. p. 107–114. Citado na página 43.

SCHIAVONI, F. L.; GOULART, A. J. H.; QUEIROZ, M. Apis para o desenvolvimento de aplicações de áudio. *Seminário Música Ciência Tecnologia*, 2012. Citado 2 vezes nas páginas 19 e 29.

WANDS, B. *Art of the Digital Age*. [S.l.]: Thames & Hudson, 2007. Citado na página 23.

WRIGHT, M. et al. Supporting the sound description interchange format in the max/msp environment. In: *ICMC*. [S.l.: s.n.], 1999. Citado na página 20.

ZOLZER, U. *DAFX: Digital Audio Effects*. 2nd. ed. [S.l.]: Wiley Publishing, 2011. Citado na página 19.

ZUBEN, P. *Música e tecnologia: o som e seus novos instrumentos*. [S.l.]: Irmãos Vitale, 2004. Citado na página 12.