Análise e classificação de Linguagens de Programação Musical[1]

Rodrigo Ramos de Araujo[2], José Mauro da Silva Sandy[3], Elder José Reioli Cirilo[4] e Flávio Luiz Schiavoni[5]

Departamento de Computação

Universidade Federal de São João del-Rei | Brasil

Resumo: As linguagens de programação musical datam dos primórdios da computação e sofreram - e ainda sofrem - uma grande influência da evolução e pesquisa na área de Linguagens de Programação. Esta influência resultou em um ecossistema de linguagens com diferentes paradigmas, mas sob o mesmo domínio, a Computação Musical. Neste artigo, apresentamos as questões históricas da evolução

[1] *Study and evaluation of music programming languages*. Submetido em: 15/08/2018. Aprovado em: 30/09/2018.

[2] Graduando em Ciência da Computação pela Universidade Federal de São João del-Rei pesquisa atualmente linguagens de programação musical no laboratório ALICE (Arts Lab in Interface, Computers, and Else). E-mail: rodrigoraraujo94@gmail.com

[3] Possui graduação em Ciência da Computação pela Universidade Presidente Antônio Carlos (2010), com especialização em Arquitetura de Sistemas Distribuídos e Gerenciamento de Projetos. Atualmente é mestrando em Ciência da Computação pela Universidade Federal de São João del-Rei onde pesquisa as possíveis colaborações em código para processos criativos distribuídos no laboratório ALICE (Arts Lab in Interface, Computers, and Else). E-mail: jmsandy@gmail.com

[4] Professor do Departamento de Ciência da Computação da Universidade Federal de São João del-Rei, atuou como pesquisador visitante na Universidade de Waterloo (Canadá) durante um período de 4 anos e como pós-doutorando no Laboratório de Engenharia de Software da PUC-Rio. Possui graduação em Ciência da Computação pela Universidade Federal de Juiz de Fora (2006). Obteve seu mestrado em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2008) e doutorado em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2012). Tem experiência na área de Engenharia de Software, com ênfase em sistemas de software configuráveis, linguagens específicas de domínio, desenvolvimento de software orientado a features, programação por usuários finais e engenharia de software experimental. E-mail: elder@ufsj.edu.br

[5] Professor Adjunto da Universidade Federal de São João del-Rei no Departamento de Computação atua como docente efetivo no Programa de Pós Graduação em Ciência da Computação (PPGCC) e no Programa Interdepartamental de Pós-Graduação Interdisciplinar em Artes, Urbanidades e Sustentabilidade (PIPAUS) desta mesma instituição onde coordena o ALICE (Arts Lab in Interface, Computers, and Else), o Grupo de Estudos em Arte Digital Colaborativa e a Orchidea (Orchestra of Ideas). Possui graduação em Ciência da Computação pela Universidade Estadual de Maringá (1999), especialização em Desenvolvimento para WEB pela Universidade Estadual de Maringá (2004), mestrado em Ciência da Computação pela Universidade Estadual de Maringá (2007) e doutorado em Ciências da Computação pela Universidade de São Paulo na Área de Computação Musical (2013). E-mail: fls@ufsj.edu.br

destas linguagens, as suas questões técnicas e de desenvolvimento e também uma análise e avaliação das mesmas levando em consideração a facilidade de uso e critérios como legibilidade, expressividade e facilidade de escrita e também a influência da comunidade no desenvolvimento das mesmas. Por fim, apresentamos uma discussão sobre esta análise e avaliação que pode auxiliar artistas e programadores.

Palavras-chave: Computação Musical, Linguagens de Programação, Linguagens Específicas de Domínio;

Abstract: Music programming languages date back to the early days of computing and have suffered and still suffer a major influence from evolution and research in the area of Programming Languages. This influence resulted in an ecosystem of languages with different paradigms but under the same domain, the Computer Music. In this article, we present the historical questions and the evolution of these languages, their technical and developmental issues and also an analysis and evaluation of them, taking into account the ease of use and criteria such as readability, expressiveness and writeability and also the influence of the community in the development of them. Finally, we present a discussion about this analysis and evaluation that can help artists / programmers in the adoption of these languages.

Keywords: Computer Music, Programming Language, Domain Specific language;

*** * ***

A Computação Musical é uma área da Ciência da Computação interdisciplinar que envolve uma gama de conceitos computacionais como: Interação Humano Computador, Inteligência Artificial, Processamento de Sinais Digitais e Linguagens de Programação. O crescente interesse em Computação Musical está relacionado ao desenvolvimento tecnológico, especialmente auxiliando a aplicação de recursos sonoros para a construção de Jogos e Sistemas Interativos ou para a promoção da Acessibilidade e Arte Digital. A incorporação de áudio gerado eletronicamente em sistemas de software proporciona consequentemente um maior interesse em metodologias de desenvolvimento de softwares e a programação direcionadas para a expressão da síntese e processamento de áudio, culminando no surgimento de diversas soluções computacionais não-convencionais. Este desenvolvimento tecnológico alavancou também algumas práticas criativas baseadas nesta tecnologia permitindo o surgimento de novas áreas artísticas musicais como, por exemplo, as práticas musicais ubíquas (KELLER, 2018). As práticas artísticas que podem fazer uso de linguagens de programação musical serão melhor detalhadas na próxima Seção deste documento.

A programação musical pode ser feita por meio de linguagens de propósito geral (*GPL - General Purpose Language*), como Java, Python ou C, que permitem o desenvolvimento de sistemas de software para esta finalidade, desde que sejam combinadas com recursos específicos para o domínio de computação musical, proporcionando ao ambiente responsável pela execução a capacidade de acessar os dispositivos de som do computador (SCHIAVONI; GOULART; QUEIROZ, 2012). Estes recursos específicos costumam ser encontrados em bibliotecas e APIs específicas para o domínio de aplicação musical e muitas vezes dependem de sua disponibilidade para determinados sistemas operacionais, plataforma ou arquitetura.

Outra possibilidade para o desenvolvimento de aplicações musicais é a utilização de linguagens específicas de domínio (*DSL - Domain Specific Language*) que são linguagens elaboradas com base nos conceitos voltados para uma área específica de atuação. Por esta razão, linguagens específicas de domínio para a computação musical trazem mais do que a capacidade de acessar os dispositivos de som do computador e incluem diversas estruturas já prontas para a criação e prototipação de aplicações musicais. Isto torna a utilização destas linguagens mais adequada para a criação de arte digital, em específico as que envolvem computação musical. Este segundo grupo de linguagens, que é o foco do presente trabalho, possui diversas características que os tornam especialmente atraentes para o desenvolvimento de aplicações computacionais voltados para a música.

Um exemplo destas características é a possibilidade de propiciar o processamento em tempo real, fato este imprescindível para a sincronização de eventos musicais e que nem sempre se encontra disponível em linguagens de propósito geral. A linguagem Java, por exemplo, que é uma linguagem de propósito geral que possui API para o desenvolvimento de aplicações musicais, traz uma grande dificuldade para processamento em tempo real, especialmente devido a seu Garbage Collector interromper a máquina virtual de tempos em tempos.

Especificamente, as linguagens de programação musical permitem dar ao computador voz e ouvidos. Diversos campos fazem uso constante de linguagens de programação musical, como: a área de processamento de sinais para música, engenharia de áudio, sonificação, produção musical, criação de novos instrumentos musicais, composição musical apoiada por computador, composição algorítmica, análise musical apoiada por computador, criação de instalação artísticas, sistemas multimídias, e realidade virtual. As linguagens de programação musical permitem a manipulação de dados musicais tanto simbólicos, como MIDI (*Musical Instrument Device Interfac*e), MusicXML ou ABC e música simbólica em geral, quanto direto, como *streams* de áudio. A discussão e utilização de linguagens de programação musical é comum nas Artes, especialmente nas áreas que envolvem música e tecnologia, e em alguns nichos da computação.

A Seção Programação Musical deste trabalho irá fazer uma breve apresentação da área de

computação musical. No Brasil, esta discussão encontra-se bastante centrada a Comissão Especial em Computação Musical da Sociedade Brasileira de Computação[6] e no Simpósio Brasileiro de Computação Musical[7]. Outros eventos possuem espaço para esta discussão, como as Seções de Interfaces e de Sonologia do Congresso da Associação Nacional de Pesquisa e Pós-graduação em Música (ANPPOM) e o Workshop de Música Ubíqua (Ubimus).

É importante salientar que as linguagens de programação musical são quase tão antigas quanto as primeiras linguagens de programação. O Music I (MATHEWS et al., 1969) foi um programa de síntese musical concebido em 1957 por Max Mathews e que foi seguido de uma família de linguagens sendo que a linguagem Music 3, pioneiramente, possibilitou o uso de computadores para realizar síntese de som (LAZZARINI, 2013). Esta linguagem de programação deu origem a família de linguagens denominada: *Music N*.

Há diversas formas de abordar a discussão quando se analisa linguagens de programação com propósitos artísticos. A discussão deste artigo concentra-se na utilização das linguagens de programação musical abordando seus elementos conceituais, paradigmas e evolução do projeto, adoção pela comunidade em geral e espaço científico para pesquisa e utilização destas linguagens. Em aspectos gerais, as principais linguagens de programação musicais são desconhecidas nas demais áreas afins da computação e boa parte dos cursos de Ciência da Computação no Brasil não as citam ou as aplicam em seus planos de curso. O mesmo ocorre em cursos de Música que utilizam suporte computacional para execução de tarefas, mas não envolvem o aluno na criação de novas plataformas e aplicativos para o fazer musical. Por fim, temos toda uma comunidade de leigos que poderiam desenvolver atividades musicais apoiadas por computadores caso tivessem conhecimento e acesso a linguagens de programação musicais. Por esta razão, a motivação deste trabalho é apresentar uma caracterização e comparação das principais linguagens voltadas para a programação musical incluindo suas características, passos iniciais para sua utilização e a relação entre elas e seus usuários.

Os tópicos abaixo conduzem a uma discussão sobre programação musical apresentando elementos específicos destas linguagens de programação, analisando seus paradigmas e critérios de programação em comparação com o atual momento que estas se encontram e sua relação com seus usuários.

1. Programação Musical

A programação musical é a área da computação voltada para a manipulação de dados musicais no

6 Informações sobre a comissão disponível em: http://www.sbc.org.br/14-comissoes/385-computacao-musical
7 O Acervo do SBCM disponível em: http://compmus.ime.usp.br/sbcm/

computador. Estes dados musicais podem ser dados de música simbólica e/ou áudio. A programação musical inclui adicionalmente o conhecimento necessário para a conversão e armazenamento da música em dispositivos computacionais, seja na forma de dados simbólicos ou na forma de áudio. A seguir, serão apresentados os principais domínios destinados a aplicação das linguagens de programação musical.

Manipulação de dados musicais: A manipulação de dados musicais na programação musical refere-se aos meios de armazenamento e representação de informações musicais simbólicas que são utilizadas na produção do som. A representação musical simbólica no computador pode ser exemplificada pelos padrões MIDI (Musical Instrument Digital Interface) e / ou MusicXML. Tais padrões suportam a representação simbólica de partituras e/ou de músicas que podem ser tocadas em sintetizadores. Como os dados armazenados nestes padrões são simbólicos, ou seja, representam eventos musicais como notas e tempos, é possível manipulá-los, analisá-los e alterá-los programaticamente. Já o armazenamento musical em formato de áudio está atrelado ao processamento digital de sinais. Neste caso, a programação musical permite a produção de *streams* de áudio por meio de técnicas de síntese de som ou de efeitos de áudio. Também é possível a análise de áudios para recuperação de informações musicais como: estilos; tempo; tonalidade; ornamentos; e afins (CUTHBERT; ARIZA, 2010).

Composição: Uma área artística que comumente utiliza linguagens de programação musical é a composição. Por meio da programação musical é possível trabalhar com geração musical utilizando técnicas de estocásticas, aleatoriedade e análise de outras composições. Na criação musical é possível a produção de arranjos, combinações e permutações de material musical já existente em novas peças musicais.

Instrumentos Musicais: Uma aplicação comum das linguagens de programação musical é a da criação de instrumentos musicais. Este é um domínio de aplicação que envolve a combinação de sensores que capturam gestos e movimentos do músico para posterior aplicação de algoritmos e modelos de síntese sonora para mapear os dados de entrada em uma saída sonora. Esta área de aplicação de linguagens de programação musical depende fortemente da capacidade de processamento em tempo real para apresentações ao vivo e performances (MIRANDA; WANDERLEY, 2006).

Orquestras de computadores portáteis: A Música costuma ser uma prática artística realizada por grupos e a música computacional não se afastou deste costume, fazendo uso de redes de computadores e de protocolos como o OSC (Open Sound Control) para a prática artística coletiva. Desde os primórdios da computação musical grupos se organizam para criar música no computador, prática esta que culminou na criação de Orquestras de Laptops, as chamadas *Laptop Orchestras* (TRUEMAN et al., 2006).

2. Um breve histórico das linguagens de programação musical

Os primeiros registros de emissão e reprodução de som de forma eletrônica datam de 1876 com a invenção do telefone por Alexandre Graham Bell. Quase um século após o telefone ter sido criado os avanços tecnológicos relacionados a produção de som e música tornaram possível a produção de som totalmente digital em um IBM 704 através de um programa de computador chamado Music I (MILETTO et al., 2004). Este programa foi seguido por uma família de linguagens de programação e o surgimento de uma primeira linguagem destinada para a criação musical no computador mudou o modo de se pensar e elaborar som (MATHEWS et al., 1969) e, consequentemente, a música.

O advento da família de linguagens de programação Music N, criada por Max Mathews, data dos primórdios da computação e é uma possível marca do início da Computação Musical. Nascida como um experimento, ela tem uma linha evolutiva duradoura, como demonstra a Tabela 1. Posteriormente, aplicações como a manipulação simbólica da música e composição passaram a ser possíveis com o advento e modernização de novas linguagens de programação musical. Estas linguagens evocam a curiosidade e despertam questionamentos quanto a capacidade de aplicações para atender usuários na criação de projetos artísticos digitais.

Ano	Versão	Linguagem de Programação
1957	Music I	Assembler (704)
1958	Music II	Assembler (704)
1960	Music III	Assembler (7090)
1963	Music IV	Assembler (7094)
1963	Music IVB	Assembler Befap (7094)
1965	Music IVF	Fortran
1966	Music IVBF	Fortran
1966	Music 6	Fortran
1968	Music V	Fortran
1969	Music 360	Assembler (IBM360), Fortran
1969	Music 10	Assembler (PDP-10)
1970	Music 7	Fortran
1973	Music 11	Assembler (PDP-11), Fortran
1977	Mus10	Asembler (DEC KL 10), Algol
1980	Cmusic	C, Csh, Cpp
1984	Cmix	Assembler (IBM 730), C, MinC
1985	Music 4C	C
1985	Csound	C

Tab. 1 – Histórico aproximado de Linguagens Music N

3. Estruturas elementares das linguagens de programação musical

Mesmo após décadas de pesquisa, praticamente todas as linguagens de programação musical modernas podem ser consideradas, de certa forma, evoluções da família Music-N. Elas utilizam o mesmo paradigma de programação da pioneira Music-1, isto é, uma programação essencialmente orientada a objetos que emprega estruturas elementares das unidades básicas geradoras (*Unit Generators - ugens*) de som e de processamento. As *ugens* podem ser configuradas e conectadas para a criação de músicas mais complexas permitindo com isto uma programação modular baseada em componentes ou unidades fundamentais de processamento. Isto permitiu a Music-N e permite a suas sucessoras, uma capacidade de abstrair funcionalidades e conceitos musicais em unidades pequenas que podem ser combinadas ou conectadas para a criação de aplicações mais complexas. Estas conexões de *ugens*, que podem ser entendidas como fluxos de áudio ou de outros dados, podem ser direcionadas para vertedouros como a saída de som do computador; arquivos, GUIs, analisadores ou outros módulos.

4. Sumário das linguagens de programação musical

4.1. Csound

Uma das linguagens sucessoras da Music-N e que ainda se encontra em atividade é o **Csound.** Este projeto foi iniciado no MIT por Barry Vercoe em 1985 e é derivado do Music 11 também escrito por Barry (BOULANGER, 2000). O Csound continuou seu desenvolvimento nos anos 2000 liderado por John Fitch na Universidade de Bath (VERCOE et al., 1986). Diferentemente de seu antecessor, o Music 11 escrito em Assembler, Barry se aproveitou da portabilidade da Linguagem C para criar inicialmente uma transcrição do Music 11 para C.

Por ser uma linguagem baseada em C (KERNIGHAN; RITCHIE, 2017) o Csound segue o mesmo paradigma de programação sendo também uma linguagem estruturada e orientada a objetos e os seus tipos dos dados mantém-se estáticos e de tipagem forte, concentrando-se no fluxo dos dados. Csound é um *software* livre disponível pela LGPL (*GNU Lesser General Public License*), multiplataforma capaz de rodar em diversos em ambientes, como Linux, Windows e Mac. A última versão do Csound, chamada Csound-6.11, conta com milhares de geradores de unidades tendo como ponto forte ser modular e extensível pelo usuário.

HarmonicAdditiveSynthesis1.csd

```
<CsoundSynthesizer>

<CsOptions>
-odac -M0 -+rtmidi=virtual -dm0
;DISABLE VIRTUAL MIDI FOR EXTERNAL MIDI CONTROL
</CsOptions>

<CsInstruments>

sr        =    44100    ;SAMPLE RATE
ksmps          =   32   ;NUMBER OF AUDIO SAMPLES IN EACH CONTROL CYCLE
nchnls         =   2    ;NUMBER OF CHANNELS (2=STEREO)
0dbfs          =   1    ;MAXIMUM SOUND INTENSITY LEVEL

;FLTK INTERFACE CODE;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
FLcolor  255, 255, 255, 0, 0, 0
;               LABEL                  | WIDTH | HEIGHT | X | Y
        FLpanel "Additive Synthesis 1",  500,    500,    0,   0

;SWITCHES                                                        ON | OFF | TYPE | WIDTH | HEIGHT | X  | Y | OPCODE | INS | STARTTIM | DUR   p4  p5  p6  p7  p8  p9  p10 p11 p12 p13 p14 p15 p16 p17 p18 p19 p20 p21 p22 p23 p24 p25 p26 p27 p28 p29 p30 p31 p32 p33
gkFLTK_MIDI,ihFLTK_MIDI      FLbutton "ON (FLTK) / OFF (MIDI)",    1,    0,    22,    180,     25,    5,    5,    0,     1,      0,      3600
FLsetColor2 255, 255, 50, ihFLTK_MIDI    ;SET SECONDARY COLOUR TO YELLOW
gkReset,ihReset            FLbutton "Reset",     1,     0,     21,     80,     22,  415,    420,    0,      4,      0,      0.01
```

Fig. 1 – Exemplo de programa Csound

4.2. Pure Data

O Pure data, ou Pd, é uma linguagem de programação visual desenvolvida por Miller Puckette durante durante a década de 90 (PUCKETTE, 1996). O Pure Data é um ambiente de programação visual cujas unidades geradoras (*ugens*) são representadas por blocos, caixinhas que se conectam, podendo processar dados de áudio, matemáticos, gráficos e mais. Os blocos do Pd são chamados de "objetos", mas esta designação não remete ao paradigma de programação Orientação à Objetos. No processo de elaboração do Pd, Puckette trouxe consigo todas as experiências acumuladas em projetos anteriores para lançar o Pd como *software* livre distribuído pela Licença BSD (*Berkeley Software Distribution*). O Pd é uma linguagem multiparadigma e interpretada em máquina virtual que também possui os tipos de dados estáticos ou dinâmicos referentes aos tipos de entrada inseridas. A implementação do Pd utiliza a linguagem tcl/tk para sua GUI e a linguagem C para seu *engine* de som e suas *ugens* internas. GUI e engine de som se conectam por rede em um modelo cliente/servidor que permite que uma máquina execute a interface gráfica do Pd enquanto outra máquina execute seu ambiente de execução.

Este ambiente / linguagem de programação é extensível por meio de plugins, chamados *externals*,

o que permitiu que com o passar do tempo outras funcionalidades fossem adicionadas aos processamentos originais, como o poder de gerar imagens 3D, vídeos ou até mesmo controlar hardware. O Pure Data pode ainda ser estendido utilizando códigos desenvolvidos no próprio Pd, que podem ser encapsulados em novos objetos e reutilizados em outras programações.

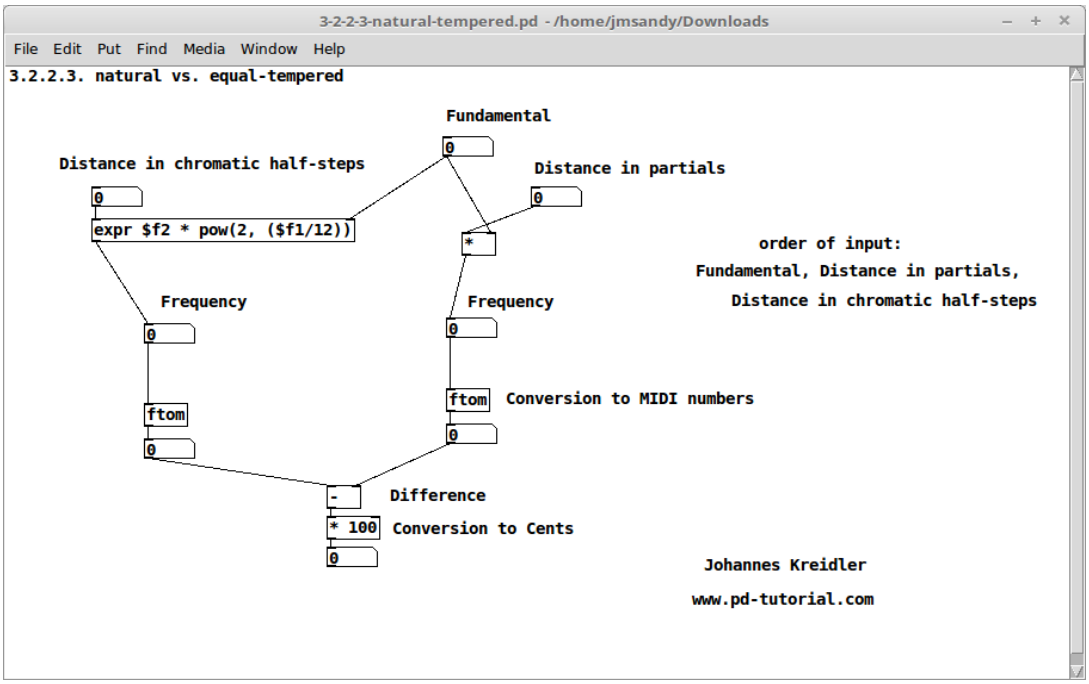

Fig. 2 – Interface e blocos do Pd

4.3. Supercollider

O Supercollider, também chamado SC, foi desenvolvido por James McCartney e é um software livre lançado em 1996 e distribuído pela GPL (GNU General Public License) a partir de 2002 (MCCARTNEY, 2002). O SC é constituído por três componentes principais: um servidor de áudio em tempo real, chamado scsynth, uma linguagem de programação interpretada, chamada sclang e um editor de sclang, chamado scide. No servidor scsynth encontram-se mais de 400 unidades geradoras para análise, síntese e processamento de som que podem ser programadas por meio da sclang. A linguagem sclang que também pode ser utilizada para gerenciar a agenda de eventos para produção de som ou criar Interfaces gráficas para o usuário (*GUI*). A linguagem sclang possui ainda interface externa podendo enviar e receber mensagens OSC da rede e MIDI de controladores externos. Supercollider é uma linguagem de programação Orientada a objetos cuja estrutura foi influenciada pela linguagem SmallTalk (GOLDBERG; ROBSON, 1983) e que também pode operar de maneira funcional devido a mesma ter sido também influenciada pela linguagem Lisp (STEELE, 1990). Por isto, dizemos que a mesma é multiparadigma, com tipagem de dados forte, porém dinâmica. Os elementos dessa linguagem

foram escritos em C++ e seu ambiente é multiplataforma por ser compilada em máquina virtual.

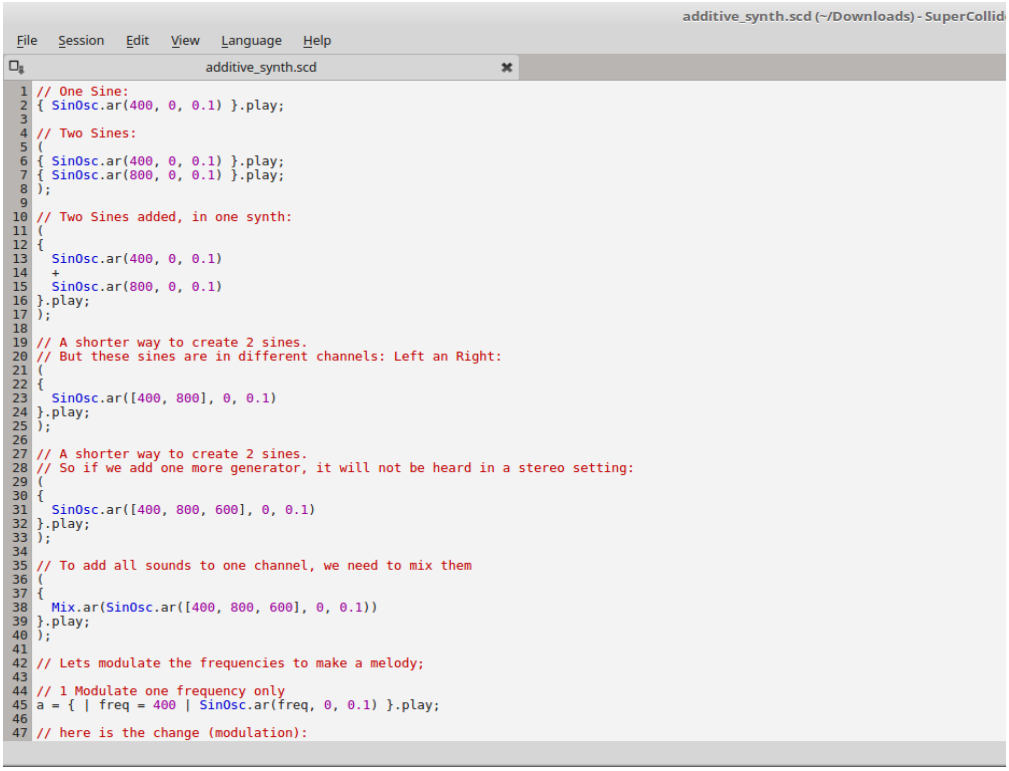

Fig. 3 – Interface e códigos do Supercollider

4.4. ChucK

Uma das mais recentes Linguagens de programação musical a ganhar certo destaque chama-se ChucK e foi desenvolvida por Ge Wang e Perry R. Cook na Universidade de Princeton em 2002. Esta linguagem surgiu como ferramenta de apoio ao ensino e a criação de meta-instrumentos do projeto de orquestra de laptops desta universidade, a PLOrk (*Princeton Laptop Orchestra*), e está presente também na implementação do *software* Smule e de outras aplicações (TRUEMAN et al., 2006). O projeto desta linguagem é um *software* livre disponível pela GPL e que permanece aberto desde a sua criação. O ChucK é uma linguagem multiparadigma, vagamente orientada a objetos, do tipo C, com os tipos de dados fortemente estruturados. A linguagem é compilada com instruções executadas em tempo real em uma máquina virtual. Esta linguagem possui uma sintaxe bastante diferente de outras linguagens de programação devido ao fato de suas atribuições serem feitas de acordo com a metáfora de uma conexão entre trechos de código. Assim, não atribuímos um valor 220 a uma variável float cf, como ocorre em C com a sintaxe **(float cf = 220;)**, mas conectamos o valor 220 a uma variável float cf usando, para isto, o conector **=>**, como apresentado na linha 9 da Figura 5, com a sintaxe **(220 => float cf;)**. Este operador (->), chamado ChucK é, segundo os autores, massivamente sobrecarregado e

pode realizar inúmeras operações distintas dependendo sempre do tipo dos dados envolvidos na operação, o que pode simplificar o entendimento e a abstração da programação nesta linguagem.

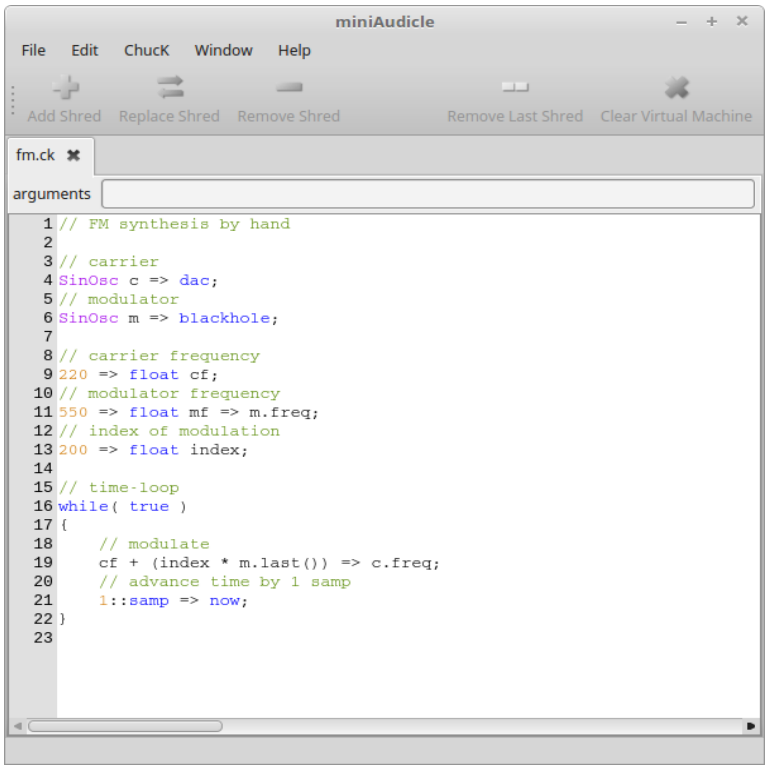

Fig. 4 – Interface e códigos ChucK no miniAudicle

4.5. FAUST

FAUST (*Functional AUdio STream*) é uma linguagem de programação puramente funcional, desenvolvida por Yann Orlarey, Dominique Fober e Stephane Letz (ORLAREY; FOBER; LETZ, 2009) em 2002 na Universidade de Lyon, na França. É a linguagem mais recente dentre as apresentadas neste trabalho. FAUST é distribuída com a licença GPL, e diferencia-se das outras linguagens pela capacidade de gerar códigos para outras linguagens. Um programa FAUST processa sinais através de funções matemáticas que transformam os sinais de entrada em saída. Assim, a linguagem FAUST é uma linguagem textual, mas orientada a diagrama de blocos algébricos, construídos através da composição de funções (ORLAREY; FOBER; LETZ, 2002). A Linguagem tem os dados com tipagem Forte e Estáticos, exigindo um compilador / gerador de código específico. Esta linguagem é capaz de gerar aplicações prontas de áudio, como páginas web baseando-se na API webaudio em javascript ou plugins de efeitos em C para a plataforma LV2 do Linux utilizando, para isto, a mesma sequência de código, mas geradores diferentes de saída. Um outro gerador pode ter como saída um diagrama SVG da conexão dos blocos, por exemplo. É importante notar que FAUST pode ser compilada diretamente

para um executável via LLVM JIT (e.g. Faust Live, faust opcodes, objetos faust MaxMSP), e não somente para outras linguagens. Graças a isto, FAUST se tornou uma linguagem extremamente poderosa como ferramenta para a criação de aplicações musicais.

Fig. 5 – Interface e código - FAUST

5. Comparação de linguagens

A primeira comparação que podemos trazer, entre as linguagens aqui apresentadas é em relação a plataforma de funcionamento. Conforme ilustrado pela Tabela 2, as linguagens aqui apresentadas funcionam nas plataformas Linux, Windows e Mac e ainda possuem possibilidade de funcionamento para as plataformas móveis Android, iOS e web.

Linguagens	Plataformas					
	GNU/Linux	Windows	Mac OS	IOS	Android	Web
Csound	x	x	x	x	x	x
Pure Data	x	x	x	x	x	
Supercollider	x	x	x	x*		
ChucK	x	x	x			
FAUST	x	x	x			x
*- plataforma fechada, funcional apenas para desenvolvedores.						

Tab. 2 - Comparação das Linguagens - Plataformas

Além disso, todas as linguagens aqui apresentadas podem ser estendidas por seus usuários desenvolvedores. Graças a isto, conforme apresentado na Tabela 3, todas as linguagens possuem

capacidade de interação com eventos MIDI e mensagens OSC e possuem capacidade de desenvolver interfaces gráficas de usuários. Além disto, boa parte destas linguagens permitem também a criação e processamento de dados de imagem e vídeo.

Linguagens	Extensões			
	MIDI	OSC	GUI	Video
Csound	x	x	x	
Pure Data	x	x	x	x
Supercollider	x	x	x	x
ChucK	x	x	x	x
FAUST	x	x	x	

Tab. 3 - Comparação das Linguagens - Extensões

6. Avaliação das linguagens musicais

Uma vez apresentadas as questões históricas das linguagens de programação musical e uma comparação técnica entre elas, também classificamos as linguagens de programação por suas funcionalidades, correspondentes a um conjunto de características capazes de definir como ela opera e soluciona problemas, denominada classificação de paradigmas de programação. Iremos ainda analisar de que modo estas linguagens se aderem a critérios de avaliação como legibilidade, capacidade de escrita e confiabilidade e também o relacionamento da comunidade com o projeto das linguagens estudadas.

Para isto, será utilizada a organização proposta pela metodologia Goal/Question/Metric (GQM) (BASILI, 1992). De acordo com esta metodologia, o escopo do estudo pode ser sumarizado como a seguir. Adicionalmente, baseado no objetivo do estudo, é apresentado a seguir as questões de pesquisa.

Analisar Linguagens de Programação Musical
com o propósito de caracterizá-las
com respeito aos critérios de legibilidade, capacidade de escrita,
confiabilidade e atuação da comunidade
do ponto de vista dos programadores/músicos
no contexto das linguagens de programação Csound,
Pure Data, Supercollider, ChucK, e FAUST.

- QP1: De que modo as linguagens de programação musical estão aderentes aos critérios de avaliação de legibilidade, capacidade de escrita e confiabilidade?

- QP2: Como a comunidade se relaciona com estas linguagens de programação musical?

As linguagens apresentadas foram todas construídas seguindo o mesmo paradigma essencialmente orientado a objetos da linguagem Music N e sua construção por *ugens,* com exceção de FAUST que, por ser funcional, não se encaixa de maneira tão precisa no paradigma orientado a objetos. Também atuam diretamente sobre o mesmo domínio que é a computação musical e suas atividades. No entanto, do ponto de vista da construção destas linguagens, as mesmas são bastante distintas. Tais características são apresentadas na Tabela 4.

Linguagens	**Modelo de Execução**	**Paradigmas**	**Tipos de Dados**
Csound	Compilado	Estruturada, Orientada a Objetos, Orientada por fluxo	Estático, forte
Pure Data	Interpretado	Multiparadigma, Visual, Estruturada	Estático ou Dinâmicos
Supercollider	Interpretado	Multiparadigma, Orientada a objeto, funcional	Dinâmico, forte
ChucK	Interpretado	Interpretado & Multiparadigma, Orientada a objeto, Procedimental	Estático,forte
FAUST	Compilado/ Geração de código	Funcional	Estático,forte

Tab. 4 - Comparação das Linguagens e seus paradigmas de programação

A primeira questão de projeto está relacionada às mesmas serem linguagens interpretadas ou compiladas. Tal questão está associada diretamente à capacidade de execução em tempo real das aplicações desenvolvidas. As linguagens interpretadas podem garantir um ambiente de execução que garanta a alta disponibilidade do processamento assim como o agendamento das tarefas dentro de um intervalo de tempo compatível com a taxa de amostragem do som gerado. Tal garantia é menor para linguagens compiladas ou geradoras de código. Neste caso, o programador precisa ter conhecimento para configurar seu sistema operacional para a execução em tempo real de maneira a garantir o processamento do áudio sem interrupção. Por outro lado, linguagens compiladas atuam diretamente sobre o Sistema operacional e podem ter um custo mais leve de execução por não necessitar do ambiente de interpretação. Por esta razão, ser compilada ou executada não pode ser considerado um atributo positivo ou negativo, mas como uma característica que pode ser explorada e avaliada no momento de optar por uma ou outra linguagem de programação.

Outra questão que pode influenciar no uso, ensino e aprendizagem destas linguagens está nos paradigmas que influenciaram suas criações. Pure Data é uma linguagem visual programada por meio de "caixinhas" e "cordinhas". ChucK, Csound e Supercollider são orientadas a objetos. Já FAUST é

uma linguagem totalmente funcional. Linguagens de programação visual podem tornar as ugens mais claras para o programador e com isto ser mais simples para programadores pouco experientes. Já programadores experientes, que estão acostumados a pensar em funções e objetos como componentes conectáveis de um código podem achar mais simples a abordagem textual.

Além desta questão, podemos ainda comparar as linguagens de acordo com seus tipos de dados. Pure Data possui tipos estáticos ou dinâmicos, FAUST possui tipagem estática e fraca enquanto ChucK e Csound possui tipagem estática forte e Supercollider possui tipagem dinâmica e forte. A tipagem de dados pode ser também associada a facilidade de programação já que uma tipagem fraca pode permitir diferentes conexões de códigos sem apresentar erro de uso de tipo incorreto. Certamente estes erros, se reportados com mensagens claras pelo compilador / interpretador, pode auxiliar o programador a desenvolver o código desejado.

6.1. Legibilidade

Legibilidade é definida como a capacidade dos programas serem lidos e compreendidos. As Linguagens apresentadas neste trabalho apresentam uma vantagem nesse quesito por se encontrarem em um domínio específico da computação. Assim, a legibilidade das linguagens pode estar associada a um conhecimento prévio do domínio, seus algoritmos clássicos e conhecidos, e também a uma lógica de programação bastante atrelada ao domínio.

Linguagens visuais, como o Pure Data, tendem a ter códigos simples de ser entendidos em um primeiro momento, mas que podem se tornar caóticos com a adição de dezenas ou centenas de objetos. A possibilidade de encapsular parte do código do usuário em novos objetos pode auxiliar o entendimento e melhorar a legibilidade do código podendo causar também outros problemas como a redefinição de objetos do sistema caso não se respeite o sistema de nomes global ou o ocultamento de um código encapsulado. Outro ponto de conflito na leitura do código no Pd é que nem sempre conseguimos saber como ocorre a sincronização de uma mensagem de áudio ou de controle do caso de a saída de um objeto estar conectada a outros 2 objetos. Esta ordem depende da ordem de inclusão do objeto no *patch* sendo que esta informação não está disponível de maneira simples no ambiente.

O Csound, por questões históricas, possui nome definido para tipos de variáveis e também parâmetros de compilação passados no próprio corpo do programa. Isto torna sua legibilidade um tanto comprometida para programadores acostumados com mais liberdade na escolha de nomes de variáveis, mas traz todos os parâmetros de configuração de maneira visível ao usuário. Além disto, esta definição auxilia programadores experientes a ler o código e identificar o tipo de uma variável pelo seu nome.

O Supercollider também é bastante legível, principalmente quando os programadores utilizam orientação a objeto no código, e possui uma característica interessante. O código do SC é enviado ao servidor para sua execução e isto pode ser feito de maneira não linear, ou seja, o início do código pode ser a última parte do mesmo a ser enviada ao servidor. Isto torna a programação extremamente flexível, mas pode comprometer a legibilidade do código e até mesmo impedir a troca de código entre programadores. Para saber a ordem de execução de uma peça em SC pode ser necessário muitas linhas de comentário e um pouco de treino para a performance.

O código em ChucK é bastante legível após uma leitura rápida em sua sintaxe apesar de o mesmo fugir drasticamente da forma de escrever código em diversas outras linguagens de programação. A questão mais aparente desta sintaxe está no fato de o ChucK se basear em conectar (em tradução livre: "chuckar"!) processamentos e variáveis por meio do conector ChucK (=>). Assim, ao invés de atribuir o valor 5 a uma variável inteira a, o programador "chucka" este valor por meio da expressão (5 => int a;).

Já a linguagem FAUST é bastante legível para programadores acostumados com linguagens funcionais e não aparenta ter maiores questões de legibilidade. No entanto, vale lembrar que não é comum aprender linguagens de programação funcionais na computação e que esta família de linguagens é mais comumente utilizada por engenheiros.

6.2. Simplicidade global

A simplicidade global do Pure Data é um tanto comprometida dado que o programador precisa saber o nome dos objetos ("caixinhas") e não há uma paleta para auxiliar a programação. A primeira impressão de programar nestes ambientes é de tatear no escuro e tentar acertar o nome de objetos que nem sempre são o que se espera. Além disto, a capacidade de estes ambientes serem estendidos por meio de plugins pode comprometer esta simplicidade para casos onde o programador não sabe se um dado objeto está ou não presente em seu ambiente de programação antes de tentar fazer uso do mesmo.

Já no Csound, a simplicidade global compete com a quantidade de *ugens* que esta linguagem possui. O fato de possuir milhares de geradores auxilia o programador, que raramente precisará criar um *ugen*, mas complica a simplicidade da linguagem e aumenta sua complexidade pois isto implica em conhecer estas unidades.

A linguagem Supercollider, apesar de possuir diversos *ugens*, possui uma simplicidade global mais evidente, em parte por utilizar interfaces de programação comum e outros recursos de linguagens orientadas a objetos. Por fim, ChucK e FAUST não possuem questões maiores em relação a sua

simplicidade global.

6.3. Tipos de Dados

A presença de facilitadores adequados a definir tipos de dados e estruturas é um auxílio notável a legibilidade (SEBESTA, 2003). As linguagens de programação musical apresentam um número distinto e característicos de tipos de dados e estruturas. Principalmente por serem DSL's e trabalharem em prol de domínio inicialmente restrito estas linguagens são capazes de usar essas mesmas estruturas para elaboração de projetos mais complexos. A Tabela 4 já apresentou uma comparação quanto ao tipo de dados destas linguagens. Em geral, essas linguagens possuem tipos que utilizados para representar música simbólica, como dados discretos unitários, eventos e notas musicais, e também representar fluxos de áudio, como conjuntos de amostras de áudio (array 1D). As linguagens que possuem capacidade de processamento de imagem e vídeo também manipulam amostras para estes tipos de dados como array 2D e 3D.

6.4. Sintaxe

Todas as linguagens aqui apresentadas possuem um fluxo de áudio contínuo que pode ser alterado pela configuração de seus geradores por um sinal de controle. Este sinal de controle pode ser discreto, baseado em eventos, ou contínuo, quando segue um fluxo como o do áudio. Por esta razão, é como se todas as linguagens tivessem embutido em sua programação um laço de repetição para os fluxos de áudio que são configurados pelo programador, mas que não dependem da sua instanciação explícita. Entendendo que este conceito é presente em todas, a sintaxe das linguagens textuais aqui analisadas são bastante claras, mesmo tendo diferenças de outras linguagens de programação, conforme apresentado anteriormente. Aqui cabe um comentário sobre a sintaxe de linguagens visuais. Pure Data possui o fluxo de áudio e controle separado sendo que o fluxo de áudio funciona em um laço de repetição interno cuja execução é atrelada a um ligar / desligar do processamento de áudio em sua GUI. Por esta razão, esta repetição não pode ser explicitamente utilizada para o processamento de sinais de controle e a criação de laços de repetição, contadores ou incrementos pode ser bastante complexa ou não intuitiva. Além disto, os objetos desta linguagem possuem entradas, chamadas de *inlets*, que podem ser "quentes" ou "frias" sendo que a alteração do valor de um inlet frio altera o estado interno do objeto mas não propaga esta alteração para os próximos objetos do fluxo conectados em suas saídas, chamadas de *outlets*.

6.5. Capacidade de Escrita

A capacidade de escrita em todas as linguagens parece ser bastante satisfatória sendo necessário notar que esta capacidade está também associada ao conhecimento específico do domínio em questão. Assim, para avaliar qual destas linguagens seria mais simples implementar, por exemplo, um sintetizador FM, é necessário partir da ideia que o usuário / programador conhece o conceito e sabe implementar este tipo de sintetizador. Vale destacar que todas as linguagens aqui apresentadas possuem capacidade para a criação de abstrações como tipos de dados complexos compostos de tipos de dados mais simples ou primitivos. Tais abstrações muitas vezes extrapolam a produção de som, como no Pure Data, e permitem a utilização destas linguagens para criação e processamento de vídeo, por exemplo. Isto amplia em muito a capacidade de escrita destas linguagens, mas pode comprometer, como comentado anteriormente, sua legibilidade.

6.6. Expressividade

A expressividade computacional nestas linguagens está novamente, associada ao domínio de aplicação em questão onde as mesmas são consideradas altamente expressivas. Por esta razão, o critério de expressividade da linguagem se torna bastante difícil de se avaliar já que este conceito facilmente se confunde com o conceito de expressividade musical ou a expressividade de um instrumento musical construído usando esta linguagem que, certamente, não trata da mesma questão. Vale notar que a experiência pessoal dos autores demonstra que muitos usuários/programadores geralmente aprendem mais de uma destas linguagens por achar que determinadas tarefas são melhores realizadas em uma ou outra linguagem de programação. Infelizmente não há consenso sobre quais tarefas são mais simples e não foi possível realizar no contexto deste artigo um mapeamento da expressividade computacional musical nestas linguagens.

6.7. Verificação de Tipos

As questões sobre tipos já foram apresentadas na Tabela 4, no entanto, cabe aqui trazer uma consideração. Como apresentado anteriormente, esta questão envolve também mensagens claras do compilador / interpretador quanto a uma atribuição de tipo incorreto. Pure Data possui tipos de dados e controle sendo que os tipos de controle podem ser variados. Por isto, a verificação de tipos ocorre apenas na execução sendo possível "conectar" objetos cujos tipos não são compatíveis. As mensagens de erro, em alguns casos, ocorrem apenas na execução e não é possível saber, em tempo de

desenvolvimento, se os objetos estão corretamente conectados.

6.8. Tratamento de Exceção

Talvez pelo fato de este tipo de tratamento ser novo na computação, apenas Supercollider traz sintaxe explícita para tratamentos de exceção. A linguagem FAUST faz amplo uso de tratamentos em seu código gerado, mas não encontramos como declarar explicitamente um tratamento de exceção nesta linguagem.

6.9. Passos iniciais para utilização das linguagens

Um meio possível para determinar uma boa escolha ao adotar uma nova linguagem de programação é caracterizar qual a aderência que a mesma possui em relação às restrições e características de problemas de implementação impostas pelas necessidades dos usuários. Neste contexto, as linguagens até aqui apresentadas foram avaliadas para determinar quais suas licenças, quais apresentam maior facilidade de instalação e possuem documentação facilmente acessível para auxiliar neste processo iniciatório. Para esta análise foi definido que a linguagem deve funcionar em plataformas Linux.

A **Licença** foi um quesito inicial de avaliação pois a partir da licença pudemos analisar se as linguagens estão ou não disponíveis gratuitamente com instalador e código-fonte. Este quesito influencia fortemente a portabilidade do código-fonte e a disponibilização da linguagem em diversos Sistemas operacionais além de influenciar a participação da comunidade nos processos de documentação e divulgação das linguagens.

O processo de **Instalação** foi feito em máquinas virtuais diferentes com o intuito de realizar a instalação em ambientes que não apresentavam nenhuma dependência e resquícios de instalações anteriores, e assim, validar o processo de forma mais consistente. Para esta etapa o sistema operacional Linux Ubuntu 18.4 foi utilizado devido a sua vasta adoção e proximidade com o usuário final. De maneira geral, a maioria das linguagens apresentaram grande facilidade no processo de instalação, sendo necessário apenas a utilização da ferramenta de gestão de pacotes do sistema operacional. Isto permitiu avaliar como **Fácil** a instalação das linguagens Csound, Pure Data, Supercollider e ChucK. A linguagem FAUST não se enquadrou neste critério pois sua instalação básica não é complicada, mas a mesma requer uma série de outras dependências, o que dificulta o processo de utilização de maneira rápida. Por esta razão, a Instalação do FAUST foi considerada de dificuldade **Média**. Entretanto, a mesma possui uma plataforma online que possibilita o usuário utilizar a linguagem sem a necessidade de instalação, o

que não simplifica a instalação, mas serve como alternativa para sua utilização.

A **Documentação** de todas as linguagens analisadas podem ser consideradas satisfatórias e há grande material disponível na Internet que facilitam a aprendizagem de forma rápida e consistente através de tutoriais, *howto's*, video-aulas, etc. Um ponto negativo observado é que não é tão fácil encontrar material que apresente todo o processo de compilação da linguagem FAUST para as diferentes plataformas que ele atua. Esta falta de documentação nos fez classificá-la como uma linguagem de mais difícil iniciação.

7. Relação: Linguagens e usuários

É possível analisar também o ambiente de divulgação destas linguagens utilizando para isto 2 aspectos: a participação da comunidade de desenvolvimento, por meio dos repositórios de código, e a participação da comunidade acadêmica por meio de publicações em meios científicos.

7.1. Comunidades de desenvolvimento

Outro mecanismo utilizado para analisar a relação entre os desenvolvedores e seus usuários, são os sites de *hosting service*. Existem três unidades que foram analisadas *Forks*(cópia de repositórios), *branches* (ramificações de atualizações de repositório) e *Pull requests* (pedidos de colaboração para mudanças em um repositório).

Começando pelo Csound obtemos 90 *Forks*, pouco menos de 15 *branches* e 2 *Pull requests* em aberto; O Pure Data com 78 *Forks* com 23 *branches* e 55 *Pull requests* já o Supercollider 377 *Forks* com 49 *branches* e 42 *Pull requests*, em seguida analisamos o ChucK com 57 *Forks*, 41 branches e 8 *Pull requests* e por fim o FAUST com 60 *Forks*, 28 *branches* e 3 *Pull requests*.

Linguagens	*Forks*	*Branches*	*Pull request*
Csound	90	15	2
Pure Data	78	23	55
Supercollider	377	49	42
ChucK	57	41	8
FAUST	60	28	3

Tab. 5 - Análise dos repositórios das linguagens de Programação e a participação da comunidade no desenvolvimento das mesmas

7.2. Comunidades científicas

Por fim, pudemos analisar a utilização e presença dessas linguagens no meio acadêmico. O Csound é uma linguagem que possui maior amparo da academia e possui tanto uma conferência acadêmica anual, a Csound Conference, quanto uma revista acadêmica, chamada Csound Journal. O Pure Data também conta com uma conferência quase anual, a PdCon. O Supercollider teve uma conferência em 2016 para celebrar os 20 anos da linguagem, chamada Source 2016. O FAUST teve em 2017 sua primeira conferência. Infelizmente não obtivemos o número de submissões / aceites para estes eventos.

Linguagens	Evento ou revista
Csound	Csound Conference (Anual) e Csound Journal
Pure Data	PdCon (Anual)
Supercollider	Source 2016
ChucK	Não identificada
FAUST	FAUST 2017

Tab. 6- Comparação das Linguagens e sua repercussão acadêmica

8. Discussões

Supercollider e Csound lideram a participação ativa em seus repositórios de código conforme dados apresentados na Seção 7. Esses dados podem indicar uma inclinação maior dos programadores mais experientes para utilizar linguagens textuais. Relacionando essas linguagens ao histórico da programação musical podemos indicar que o Csound tenha uma leve vantagem em relação às outras por fazer parte da família Music N e ter mais tempo em uso comparado às outras linguagens de software livre. No entanto, as questões de legibilidades apresentadas na Seção 6 pode torná-la menos atrativa para novos programadores se comparada com o Supercollider, que surpreende com 4 vezes mais Forks que o Csound.

A linguagem ChucK e FAUST, mais novas e com menos tempo de utilização, parecem estar ganhando espaço entre usuários programadores. FAUST possui uma capacidade muito interessante devido ao fato de gerar código para diversas plataformas. No entanto, a programação funcional não é tão comum para programadores e isto pode tornar sua adoção um tanto comprometida.

A preocupação com os usuários parece ser uma característica comum a todas as linguagens aqui analisadas e pode ser vista na análise apresentada na Seção 7. As linguagens aqui apresentadas se demonstraram bastante amigáveis a usuários iniciantes por possuírem ampla documentação, pacotes

simples de instalação e licença clara de uso.

A existência de congressos e revistas dedicadas a estas linguagens, apresentadas na Seção 7, demonstram ainda que as mesmas possuem uma forte relação com a academia e que as pesquisas feitas neste domínio estão sendo publicadas e divulgadas com uma certa preocupação e rigor acadêmico.

Conclusão

Este trabalho se propôs a trazer uma visão geral sobre linguagens de programação musical partindo de uma apresentação da área passando por questões históricas, aspectos técnicos, critérios de avaliação até chegar uma análise da relação entre estas linguagens e seus usuários. Esta visão geral demonstra que a existência de linguagens de programação musicais que utilizam paradigmas tão distintos é um aspecto central para esta área.

Apresentadas as principais características de cada linguagem de programação e seus históricos e suas influências, critérios e abstrações, foram avaliados os passos para um primeiro contato com estas linguagens para entender como isso pode ou não influenciar o futuro programador/músico a escolher/adotar uma determinada linguagem. Depois apresentamos números referentes à comunidade de usuários e sua participação na elaboração dos projetos.

Percebe-se uma necessidade dos programadores / músicos de ser parte ativa do processo de criação assumindo para si o papel de desenvolvedor de suas aplicações, composições, instrumentos e instalações. Podemos ver estas linguagens como unidade de conhecimento capazes de gerar representatividade em um grupo, e por isso nossa análise passou pela sua interação com o seu usuário, tornando-o presente nesta pesquisa como um fator determinante de análise das mesmas.

Considerar que linguagens do final dos anos 50 repercutem na formação de áudio digital atual, demonstra o progresso em uma área de pesquisa da computação datada de muito longe e que influenciam o modelo de programação de áudio e outras mídias hoje ou nos próximos anos.

Há ainda, no contexto de programação musical, outras linguagens e ambientes de programação que não foram incluídos neste estudo devido a compatibilidade com o escopo do mesmo. Entre estas linguagens estão linguagens históricas com CMusic (MOORE, 1983) e CMix (GARTON; TOPPER, 1997), cuja utilização não se mostra atualmente presente. No caso da linguagem CMix, a mesma continuou na forma de sua versão mais atual e em tempo real chamada RTCMix (Real-Time). Também não foi incluído os ambientes de programação visual Eyesweb (CAMURRI et al., 2004) e Max/MSP. Estes ambientes são similares ao Pure Data em alguns aspectos, porém, o Max/MSP é proprietário, e por isto, fora do escopo desta pesquisa. Já o ambiente de programação Eyesweb pode ser utilizado para computação musical, mas possui o foco em processamento de imagens e visão computacional. Por fim,

também não foram abordados os ambientes de programação visual OpenMusic (BRESSON; AGON; ASSAYAG, 2011) -- ambiente focado em composição e que funciona sobre a linguagem LISP -- e a linguagem Nyquist (DANNENBERG, 1997) -- que implementa o processamento de áudio e música sobre a linguagem LISP.

A pesquisa acadêmica e a presença da mesma não é sentida apenas nas revistas e congressos especialistas de cada linguagem mas também em revistas e eventos da área de computação musical, música ou computação em geral. Pretende-se como um desdobramento mapear sistematicamente a literatura e investigar a presença das mais diversas linguagens em publicações científicas como as bases indexadas de publicações de periódicos e anais de conferências.

Agradecimentos

Os autores gostariam de agradecer a bolsa de iniciação científica cedida pela Universidade Federal de São João del-Rei para a realização desta pesquisa e também a o grupo de pesquisa ALICE (Arts Lab in Interfaces, Computers, and Everything Else).

A presente pesquisa encontra-se no contexto da criação de um ambiente de programação visual para a área das Artes digitais chamado Mosaicode. Este ambiente de programação visual possui suporte para a criação de artes sonoras e visuais de maneira a simplificar o desenvolvimento especialmente para programadores leigos tendo mostrando-se eficiente para a criação de aplicações musicais (SCHIAVONI; GONÇALVES, 2017). O Mosaicode difere das linguagens apresentadas neste artigo por ser uma ferramenta de geração de código a partir de diagramas e se utilizar de GPLs com APIs específicas para a construção de aplicações artísticas. É intenção dos desenvolvedores deste projeto evoluir este ambiente de forma a transformá-lo em uma linguagem de programação para as artes. Assim, contextualizar este ambiente entre as linguagens de programação musical também é parte da motivação deste artigo.

Referências

BASILI, V. R. *Software modeling and measurement: the Goal/Question/Metric paradigm*. [S.l.], 1992.

BOULANGER, Richard Charles et al. (Ed.). *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*. MIT press, 2000.

BRESSON, J.; AGON, C.; ASSAYAG, G. *Openmusic: visual programming environment for music composition,analysis and research*. In: ACM. Proceedings of the 19th ACM international conference on Multimedia. [S.l.],2011. p. 743–746.

CAMURRI, A. et al. *Toward real-time multimodal processing: Eyesweb 4.0*. In: CITESEER. Proc. Artificial Intelligence and the Simulation of Behaviour (AISB) 2004 Convention: Motion, Emotion and

Cognition,. [S.l.], 2004. p. 22–26.

CUTHBERT, M. S.; ARIZA, C. *Music21: A toolkit for computer-aided musicology and symbolic music data.* In: International Simposium on Music Information Retrieve. [S.l.: s.n.], 2010. p. 637–642.

DANNENBERG, R. B. *Machine tongues XIX: Nyquist, a language for composition and sound synthesis.* Computer Music Journal, JSTOR, v. 21, n. 3, p. 50–60, 1997.

GARTON, B.; TOPPER, D. *Rtcmix-using cmix in real time.* In: ICMC. [S.l.: s.n.], 1997.

GOLDBERG, A.; ROBSON, D. *Smalltalk-80: the language and its implementation.* [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1983.

KELLER, D. *Challenges for a second decade of Ubimus research: Knowledge transfer in Ubimus activities.* Música Hodie, v. 18, p. 147–165, 2018.

KERNIGHAN, B.; RITCHIE, D. M. *The C programming language.* [S.l.]: Prentice hall, 2017.

LAZZARINI, Victor. *The development of computer music programming systems.* Journal of New Music Research, v. 42, n. 1, p. 97-110, 2013.

MATHEWS, M. V.; MILLER, J. E.; MOORE, F. R.; PIERCE, J. R.; RISSET, J.-C. T*he technology of computer music.* [S.l.]: MIT press Cambridge, 1969. v. 969.

MCCARTNEY, J. *Rethinking the computer music language: Supercollider.* Computer Music Journal, MIT Press, v. 26, n. 4, p. 61–68, 2002.

MILETTO, E. M. et al. *Introdução à computação musical.* In: IV Congresso Brasileiro de Computação. [S.l.:s.n.], 2004.

MIRANDA, E. R.; WANDERLEY, M. M. *New digital musical instruments: control and interaction beyond the keyboard.* [S.l.]: AR Editions, Inc., 2006. v. 21.

MOORE, F. *Introduction to music synthesis using Cmusic.* Computer Audio Research Laboratory, University of California, San Diego, 1983.

MYERS, B. A. *Taxonomies of visual programming and program visualization.* Journal of Visual Languages & Computing, Elsevier, v. 1, n. 1, p. 97–123, 1990.

ORLAREY, Y.; FOBER, D.; LETZ, S. *An algebra for block diagram languages.* In: CITESEER. Proceedings of International Computer Music Conference. [S.l.], 2002. p. 542–547

ORLAREY, Y. *FAUST: an efficient functional approach to DSP programming.* New Computational Paradigms for Computer Music, Editions Delatour, Paris, France, v. 290, p. 14, 2009.

PUCKETTE, M. S. *Pure data.* In: International Computer Music Conference. San Francisco: International Computer Music Association, 1996. v. 1997, p. 224–227.

SCHIAVONI, F. L.; GOULART, A. J. H.; QUEIROZ, M. *APIs para o desenvolvimento de aplicações de áudio.* Seminário Música Ciência Tecnologia, v. 1, n. 4, 2012.

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.

SEBESTA, R. W. *Conceitos de Linguagem de Programação.* 5a Edição. [S.l.]: Editora Bookman Companhia, 2003.

STEELE, G. Common *LISP: the language.* [S.l.]: Elsevier, 1990.

TRUEMAN, D.; COOK, P. R.; SMALLWOOD, S.; WANG, G. *PLOrk: The princeton laptop orchestra, year 1.* In: ICMC. [S.l.: s.n.], 2006.

VERCOE, B. et al. *Csound. The Csound Manual Version*, v. 3, 1986.