Estudo e avaliação de Linguagens de Programação Musical

Rodrigo R. Araújo, Flávio L. Schiavoni,José Mauro da S. Sandy, Elder José R. Cirilo

[1]Departamento de Ciência da Computação
Universidade Federal do São João del Rei (UFSJ)

rodrigoraraujo94@gmail.com, fls@ufsj.edu.br,

jmsandy@gmail.com, elder@ufsj.edu.br

Abstract. *Music programming languages date back to the early days of computing and have suffered and still suffer a major influence from evolution and researchs in the area of Programming Languages. This influence resulted in an ecosystem of languages with different paradigms but under the same domain, Computer Music. In this article, we present the historical questions of 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. Finally, we present a discussion about this analysis and evaluation that can help artists / programmers in the adoption of these languages.*

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 histórias da evolução 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. Por fim, apresentamos uma discussão sobre esta análise e avaliação que pode auxiliar artistas / programadores na adoção destas linguagens.*

1. Introdução

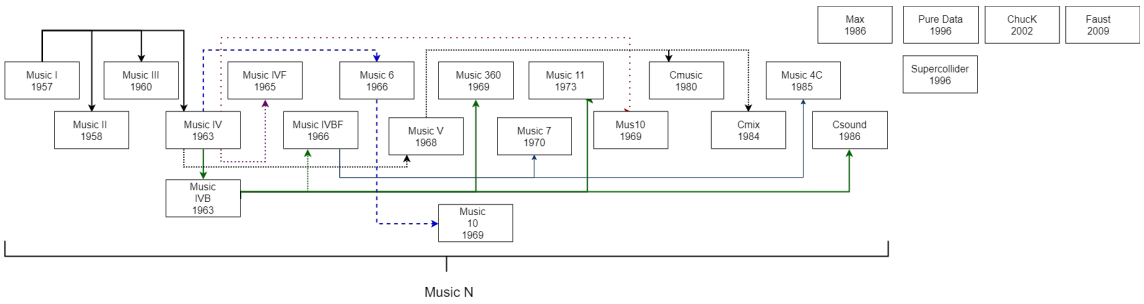

Figura 1. *Timeline*- Linguagens de Programação musical

Computação Musical é uma área da Ciência da Computação interdisciplinar que envolve uma gama de conceitos computacionais como: Interação Humano Computa-

dor [Dix 2009], Inteligência Artificial [Russell and Norvig 2016], Processamento de Sinais Digitais e Linguagens de Programação. O crescente interesse em Computação Musical está relacionado ao desenvolvimento tecnológico, especialmente na 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.

Esta programação musical pode ser feita por meio de linguagens de propósito geral (*GPL - General Propose Language*) que permitem o desenvolvimento de sistemas de software para este propósito desde que aumentadas com recursos específicos para esta área que proporcionam ao ambiente de execução a capacidade de acessar os dispositivos de som do computador [Schiavoni et al. 2012]. Há também a possibilidade de se utilizar as linguagens específicas de domínio (*DSL - Domain Specific Language*) que são linguagens elaboradas com base nos conceitos desenvolvidos em arte digital e mais especificamente em computação musical. Este segundo grupo de linguagens, que é o foco do presente trabalho, possui diversas características que as tornam especialmente atraentes para o desenvolvimento de aplicações computacionais para a música. Um exemplo destas características é a capacidade de processamento em tempo real, imprescindível para a sincronização de eventos musicais e nem sempre disponível em linguagens de propósito geral.

Especificamente, as linguagens de programação musical permitem dar ao computador voz e ouvidos. Diversos campos fazem uso constante destas linguagens de programação 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, sistemas multimídias, e realidade virtual. 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 e na Música Ubíqua.

É importante salientar que as linguagens de programação musical são quase tão antigas quanto as primeiras linguagens de programação. Music I [Mathews et al. 1969], a linguagem de programação que pioneiramente possibilitou o uso de computadores para a síntese de som, foi concebida em 1957 por Max Mathews [Mathews et al. 1969]. Esta linguagem de programação deu origem a família de linguagens *Music N* [Di Nunzio 2008]. Um breve histórico destas linguagens será apresentado na Seção 2 deste documento.

No entanto, a discussão concentra-se na utilização das linguagens para a criação artística, e pouco vem sendo abordado na literatura sobre seus elementos conceituais, paradigmas e evolução do projeto. No geral, as principais linguagens de programação 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 currículos. Por esta razão, a motivação deste trabalho é apresentar uma caracterização e comparação das principais linguagens de programação musical, conforme apresentado na Seção 3, incluindo suas características, passos iniciais para sua utilização e a relação entre estas linguagens e seus

usuários.

Os tópicos abaixo conduzem a uma discussão sobre programação musical apresentando elementos específicos dessas linguagem de programação, analisando paradigmas e critérios de programação em comparação quanto ao atual momento dessas linguagens e sua relação de seus usuários.

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 [Miletto et al. 2004]. 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 IBM704 através de uma linguagem de programação chamada Music I, o que mudou o modo de se elaborar som [Mathews et al. 1969].

O advento da linguagem de programação Music I, criada por Max Mathews, data dos primórdios da computação e marca o início da Computação Musical. A Music I é provavelmente uma das primeiras linguagens de programação de terceira geração da história dos computadores. Nascida como um experimento, ela tem uma linha evolutiva duradoura, como mostra a Figura 1. Posteriormente, aplicações como a manipulação simbólica da música, composição e live coding passaram a ser possíveis com o advento e modernização de novas linguagens de programação musical.

2.1. Estruturas Elementares das Linguagens de Programação Musical

Mesmo após décadas de pesquisa, praticamente todas as linguagens de programação musical modernas podem ser consideras evoluções da família Music-N. Elas utilizam o mesmo paradigma de programação da pioneira Music-1, isto é, empregam 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. assim como também, as conexões de *ugens* (fluxo de áudio) podem ser direcionadas para vertedouros: saída de som do computador; arquivos; GUIs; analisadores.

2.2. Sumário das Linguagens de Programação Musical

É apresentado a seguir um sumário das características de projeto das linguagens de programação mais representativas e o impacto das mesmas na Computação Musical

Csound Projeto iniciado no MIT por Barry Vercoe em 1985 derivado do Music 11 também escrito por Barry, 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 característica da Linguagem C para criar inicialmente uma transcrição do Music 11 para C. Por ser uma linguagem baseada em C [Kernighan and Ritchie 2017] o Csound manteve seus paradigmas de programação sendo uma linguagem estruturada e procedimental e os seus tipos dos dados mantém-se Estáticos e de fraca tipagem como no C, 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. Já há algumas versões o CSound ganhou a capacidade de validação de código em tempo real, o que permite que o mesmo seja utilizado para programação em *live coding*.

Max/MSP Diferente de outras linguagens citadas neste trabalho, o Max é um *software* proprietário, mantido pela Cycling'74 [1]. Escrito por Miller Puckette inicialmente para o IRCAM (*Institut de Recherche et Coordination Acoustique/Musique*) em Paris, em meados de 1986[Favreau and Rowe 1986]. Em 1989, licencia para Opcode System, gerando no ano seguinte a primeira versão comercial do Max com as extensões desenvolvidas por David Zicarelli [Zicarelli 2002]. Em 1997 a Opcode System move a licença para Cycling'74 fundada por Zicarelli. Em 1997, aproveitando-se de sua característica de ser modular e extensível, é lançada a extensão MSP, pacote capaz de manipular sinais de áudio em tempo real, ativada no momento em que se inicia o *software* até o momento em que é fechado, tornando o agora Max/MSP uma linguagem para processamento de sinais com capacidade de controle por MIDI e outros controles em hardware. O Max/MSP é uma linguagem de programação multiparadigma, interpretada em máquina virtual[Myers 1990], reconhecida por sua programação visual, multimídia, com os tipos de dados estáticos ou dinâmicos referente aos tipos de entrada inseridas. A facilidade de alterar a ordem das ugens em tempo real faz com que este ambiente seja bastante utilizado para performances de *live coding*.

Pure Data O Pure data, ou Pd, é uma linguagem de programação visual assim como Max, também desenvolvida por Miller Puckette durante durante a década de 90[Puckette 1996]. O Pd é baseado na ideia de blocos/caixas 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 no desenvolvimento do Max 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. O Pd é extensível por meio de plugins chamados *externals* e isto permitiu que ao passar dos anos outras funcionalidades fossem adicionadas como o poder de gerar imagens 3D, vídeos ou até mesmo controlar hardwares. 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. Esta linguagem também é amplamente utilizada para programação *live coding*.

Supercollider O Supercollider, também chamado SC, foi desenvolvido por James McCartney e é um software livre lançado em 1996 [McCartney 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. O scsynth mais de 400 unidades geradoras para análise, síntese e processamento que podem ser programadas por meio da sclang que também pode ser utilizada para

[1] Site disponível em: `https://cycling74.com`.

gerenciar a agenda de eventos para produção de som ou criar as GUIs(*graphical user interface*). A linguagem possui ainda interface externa podendo enviar e receber mensagens OSC da rede e MIDI de controladores em Hardware. Supercollider é uma linguagem de programação Orientada a objetos cuja estrutura foi influenciada pela linguagem Small-Talke que também pode operar de maneira funcional devido a mesma ter sido também influenciada pela linguagem Lisp.Por isto, dizemos que a mesma é multiparadigma, com tipagem de dados forte porém dinâmica. Os elementos desta linguagem foram escritos em C++ e seu ambiente é multiplataforma, capaz de executar *live coding* por ser compilada em máquina virtual.

ChucK Uma das mais recentes Linguagens de programação musical a obter sucesso, foi desenvolvida por Ge Wang e Perry R. Cook na Universidade de Princenton em 2002. Usada para auxiliar na execução do projeto PLOrk(*Princeton Laptop Orchestra*)[Trueman et al. 2006] esta linguagem está presente também na implementação do *software* Smule entre outras aplicações. O projeto é um *software* livre disponível pela GPL e permanece aberto. 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. Essa execução em tempo real torna possível *live coding* e programação em tempo real, uma das principais características do ChucK.

FAUST FAUST (*Functional AUdio STream*) é uma linguagem de programação puramente funcional, desenvolvida por Yann Orlarey, Dominique Fober e Stephane Letz [Orlarey et al. 2009] em 2002 na Universidade de Lyon, na França. É a linguagem mais recente dentre as apresentadas neste trabalho. Distribuída com a licença GPL, ela 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. FAUST é uma linguagem textual, mas orientada a diagrama de blocos algébricos, construídos através da composição de funções[Orlarey et al. 2002]. A Linguagem tem os dados com tipagem Forte e Estáticos, exigindo um compilador / gerador de código específico.

Tabela 1. Comparação das Linguagens e seus paradigmas de programação

Linguagens	Modelo de Execução	Paradigmas	Tipo de dados
Csound	Compilado	Estruturada, Procedimental, Orientada por fluxo	Estático, fraco
Max/MSP	Interpretado	Multiparadigma, Visual, Orientada a objetos	Estático ou Dinâmicos
Pure Data	Interpretado	Multiparadigma, Visual, Estruturada	Estático ou Dinâmicos
Supercollider	Interpretado	Multiparadigma, Orientada a objeto, funcional	Dinâmico, forte
ChucK	Interpretado	Multiparadigma, Orientada a objeto, Procedimental	Estático,forte
FAUST	Geração de código	Funcional	Estático, fraco

3. Avaliação das linguagens musicais

Uma vez apresentadas as questões históricas das linguagens de programação musical e uma comparação técnica entre elas, pretendemos analisar de que modo as linguagens mais representativas se aderem a critérios de avaliação como legibilidade, capacidade de escrita e confiabilidade e também o relacionamento da comunidade com o projeto das linguagens estudas.

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 caracteriza-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, Max/MSP, Pure Data, Supercollider, ChucK, e FAUST.

- QP_1: De que modo as linguagens de programação musical estão aderentes as critérios de avaliação de legibilidade, capacidade de escrita e confiabilidade?
- QP_2: Como a comunidade de se relaciona com estas linguagens de programação musical?

3.1. QP_1: Avaliação Técnica das Linguagens de Programação Musical

As linguagens apresentadas foram todas construídas seguindo o mesmo paradigma da linguagem Music N e sua construção por *ugens*. 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 1.

A primeira questão de projeto está relacionado 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.

Outra questão que pode influenciar no uso, ensino e aprendizagem destas linguagens está nos paradigmas que influenciaram suas criações. Pure Data e Max/MSP, são linguagens visuais programadas por meio de "caixinhas" e "cordinhas". Chuck e Supercollider são orientadas a objetos enquanto CSound é uma linguagem procedimental. Já FAUST é uma linguagem totalmente funcional.

Além desta questão, podemos ainda comparar as linguagens de acordo com seus tipos de dados. Pure Data e Max/MSP possuem tipos estáticos ou dinâmicos, FAUST e Csound possuem tipagem estática e fraca enquanto ChucK possui tipagem estática forte e Supercollider possui tipagem dinâmica e forte.

Apesar de ser bastante controverso classificar linguagens de programação, os critérios a seguir são normalmente os mais aceitos entre os métodos acadêmicos utilizados para este fim.

Legibilidade Linguagens visuais, como Pure Data e Max/MSP, 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. Outro ponto de conflito na leitura do código 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.

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 é um tanto comprometida para programadores acostumados com mais liberdade na escolha de nomes de variáveis. Contudo, tal definição auxilia programadores experientes a ler o código já que pelo nome de uma variável é possível saber seu tipo.

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.

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 $=>$. Assim, ao invés de atribuir o valor 5 a uma variável inteira a, o programador "chucka" este valor por meio da expressão $5 => inta;$. O FAUST é bastante legível para programadores acostumados com linguagens funcionais e não aparenta ter maiores questões de legibilidade.

Simplicidade global A simplicidade global tanto do Pure Data quanto do Max/MSP é um tanto comprometida dado que o programador precisa saber o nome dos objetos ("caixinhas") e não há uma paleta para auxiliar a programação. 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 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.

Tipos de Dados A presença de facilitadores adequados a definir tipos de dados e estruturas é um auxilio 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 como Csound, Chuck e FAUST, porém Max/MSP,Pd e Supercollider são capazes de usar essas mesmas estruturas para elaboração de projetos mais complexos. A Tabela1 já apresentou uma comparação quanto ao tipo de dados destas linguagens.

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 das linguagens visuais. Pure Data e Max/MSP possuem o fluxo de áudio e controle separado e a criação de laços de repetição, contadores ou incrementos pode ser bastante complexa. 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*.

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. Tais abstrações muitas vezes extrapolam a produção de som, como no Max/MSP e no Pure Data, e permitem a utilização destas linguagens para criação e processamento de vídeo, por exemplo.

Expressividade A expressividade computacional nestas linguagens está novamente, associada ao domínio de aplicação em questão onde as mesmas são consideradas altamente expressivas. Vale notar aqui que a experiência pessoal dos autores demonstra que muitos usuários / programadores geralmente aprende 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 em quais linguagens e não foi possível realizar um mapeamento da expressividade computacional musical nestas linguagens.

Verificação de Tipos As questões sobre tipos já foram apresentadas na Tabela 1, no entanto, cabe aqui trazer uma consideração. Pure Data e Max/MSP possuem 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.

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.

3.2. QP_2: Linguagens de Programação Musical e a Comunidade

Um dos quesitos para iniciar e se manter interessado a aprendizagem de uma linguagem de programação vai além dos quesitos técnicos e das capacidades computacionais

da linguagem. Esta seção apresenta uma analise do esforço necessário para se iniciar a utilização das linguagens de programação musical estudadas e a relação das mesmas com as comunidades acadêmica e não-acadêmica.

3.2.1. Passos Iniciais para Utilização das Linguagens

Um meio possível para determinar uma boa escolha ao adotar uma nova linguagem de programação é caracterizar quais as que melhor se aderem as restrições e características 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, o que impediu a avaliação do ambiente Max/MSP.

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.

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*, videoaulas, 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, o que a tornou a linguagem de mais difícil iniciação. Vale ressaltar que ao utilizar o FAUST através da plataforma online que é disponibilizada estes problemas não acontecem, mas que este fato pode dificultar sua utilização em um primeiro momento para usuários que queiram ter o ambiente em suas máquinas.

3.2.2. Relação: Linguagens e usuários

É possível analisar também o ambiente de divulgação destas linguagens utilizando para isto 3 aspectos: a divulgação na comunidade não acadêmica, por meio das redes sociais, 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.

Comunidades não-acadêmicas Uma das variáveis presentes na popularização e distribuição dos *softwares* é o *marketing buzz*, correspondente a divulgação do produto pelos seus utilizadores. Para analisar a popularização destas linguagens em comunidades não apenas acadêmicas coletamos dados das linguagens de Programação musical citadas em grupos e páginas do Facebook. Entre as linguagens a que apresentou o maior número de seguidores/membros foi o Max/MSP com aproximadamente 16.000 membros[2], seguido pelo Pure Data com aproximadamente 12.000 membros[3]. O grupo criado para o Supercollider se aproxima de 5.000 membros[4], e o ChucK com pouco mais de 200 membros [5]. FAUST não possui Grupo mas a página da linguagem conta com aproximadamente 400 seguidores[6]. O único grupo de Csound encontrado é italiano e conta com cerca de 360 usuários[7]. É necessário ressaltar que nenhuma página ou grupo é gerenciada pelos criadores de alguma dessas linguagens.

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*; 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*.

Comunidades científicas Por fim, podemos analisar a utilização e presença destas 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 Max/MSP teve algumas conferências isoladas em 2009 e 2011 e o FAUST teve em 2017 sua primeira conferência. Infelizmente não obtivemos o número de submissões / aceites para estes eventos.

4. 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

[2]Grupo do Max/MSP disponível em: `https://www.facebook.com/groups/maxmspjitter/`.

[3]Grupo do Pd disponível em: `https://www.facebook.com/groups/4729684494/`.

[4]Grupo do SC disponível em: `https://www.facebook.com/groups/supercollider/`.

[5]Grupo do ChucK em: `https://www.facebook.com/groups/1593843507578422/`.

[6]Grupo do FAUST disponível em: `https://www.facebook.com/GRAMEfaust/`.

[7]Grupo do CS disponível em: `https://www.facebook.com/groups/49295609818/`.

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, tornado-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.

Limitações do Estudo 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 CMusice CMix,que estão presentes na Figura 1 mas cuja utilização não se mostra atualmente presente. Também não foi incluído o ambiente de programação visual Eyesweb.Este ambiente é similar ao Pure Data em alguns aspectos e pode que ser utilizado para computação musical mas possui o foco em processamento de imagens e visão computacional. Por fim, também não foram abordados os ambiente de programação visual OpenMusic– ambiente focado em composição e que funciona sobre a linguagem LISP – e a linguagem Nyquist– implementa o processamento de áudio e música sobre a linguagem LISP.

Agradecimentos Os autores gostariam de agradecer a bolsa de iniciação científica cedida pela UFSJ / CNPq / FAPEMIG para a realização desta pesquisa e também ao ALICE (Arts Lab in Interfaces, Computers, and Else). A presente pesquisa encontra-se no contexto da criação de um ambiente de programação visual para a área das Artes digitais chamado Mosaicode, desenvolvida no ALICE. 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 mostrado-se eficiente para a criação de aplicações musicais[Schiavoni and 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. (1992). Software modeling and measurement: The goal/question/metric paradigm. Technical report, University of Maryland at College Park, College Park, MD, USA.

Di Nunzio, A. (2008). *Genesi, sviluppo e diffusione del software"Music N"nella storia della composizione informatica.* PhD thesis, Bologna University - Philosophy and Letters.

Dix, A. (2009). Human-computer interaction. In *Encyclopedia of database systems*, pages 1327–1331. Springer.

Favreau, E., F. M. K. O. P. P. P. M. and Rowe, R. (1986). Software developments for the 4x real-time system. *International Computer Music Conference. San Francisco: International Computer Music Association*, pages 43–46.

Kernighan, B. and Ritchie, D. M. (2017). *The C programming language*. Prentice hall.

Mathews, M. V., Miller, J. E., Moore, F. R., Pierce, J. R., and Risset, J.-C. (1969). *The technology of computer music*, volume 969. MIT press Cambridge.

McCartney, J. (1996). Supercollider, a new real time synthesis language. In *ICMC*.

McCartney, J. (2002). Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68.

Miletto, E. M., Costalonga, L. L., Flores, L. V., Fritsch, E. F., Pimenta, M. S., and Vicari, R. M. (2004). Introdução à computação musical. In *IV Congresso Brasileiro de Computação*.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123.

Orlarey, Y., Fober, D., and Letz, S. (2002). An algebra for block diagram languages. In *Proceedings of International Computer Music Conference*, pages 542–547. Citeseer.

Orlarey, Y., Fober, D., and Letz, S. (2009). Faust: an efficient functional approach to dsp programming. *New Computational Paradigms for Computer Music*, 290:14.

Puckette, M. S. (1996). Pure data. In *International Computer Music Conference.*, volume 1997, pages 224–227, San Francisco. International Computer Music Association.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Schiavoni, F. L. and Gonçalves, L. L. (2017). Teste de usabilidade do sistema mosaicode. In *Anais [do] IV Workshop de Iniciação Científica em Sistemas de Informação (WICSI)*, pages 5–8, Lavras - MG - Brazil.

Schiavoni, F. L., Goulart, A. J. H., and Queiroz, M. (2012). Apis para o desenvolvimento de aplicações de áudio. *Seminário Música Ciência Tecnologia*, 1(4).

Sebesta, R. W. (2003). *Conceitos de Linguagem de Programação. 5a Edição*. Editora Bookman Companhia.

Trueman, D., Cook, P. R., Smallwood, S., and Wang, G. (2006). Plork: The princeton laptop orchestra, year 1. In *ICMC*.

Vercoe, B. et al. (1986). Csound. *The CSound Manual Version*, 3.

Zicarelli, D. (2002). How i learned to love a program that does nothing. *Computer Music Journal*, 26(4):44–51.