Do Harpia ao Mosaicode a Evolução de um Ambiente de Programação Visual

Luan Luiz Gonçalves[1], **Flávio Luiz Schiavoni**[1]

[1] Arts Lab in Interfaces, Computers, and Everything Else - ALICE
Federal University of São João del-Rei - UFSJ
São João del-Rei - MG

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

Abstract. *The Harpia programming environment was an important tool to develop Computer Vision applications and it became obsolete since its dependencies became deprecated. This paper presents the code refactoring of this tool and the evolution of it to a new programming environment called Mosaicode.*

Resumo. *O ambiente de programação Harpia era uma importante ferramenta para o desenvolvimento de aplicações de visão computacional que tornou-se obsoleto devido a suas dependências terem se tornado depreciadas. Este artigo apresenta a refatoração do código desta ferramenta e a sua evolução para um novo ambiente de programação, chamado Mosaicode.*

1. Introdução

O Harpia é um ambiente de programação visual que funciona como um **Gerador de código** para aplicações no domínio de processamento de imagens voltado para auxílio na educação, treinamento, implementação e gerenciamento de sistemas de Visão Computacional. Esta ferramenta foi criada em 2003 dentro do edital CT-INFO 2003 - Software Livre da FINEP pelo grupo de pesquisa multidisciplinar S2i da Universidade Federal de Santa Catarina (UFSC) e esteve presente nos repositórios oficiais do Debian e Ubuntu até aproximadamente 2009 quando seu projeto foi descontinuado. Apesar de não existir uma necessidade evidente de manutenção em seu código, a mesma possuía dependências que tiveram seu desenvolvimento descontinuado e por este motivo encontrava-se inoperante para os sistemas atuais. O presente artigo apresenta a evolução do Harpia e sua transformação na ferramenta Mosaicode a partir de uma iniciativa de torná-la operante novamente.

Como outros ambientes de programação voltados para a área de Processamento de Sinais, o Harpia utiliza o paradigma de programação visual onde a programação é feita criando **Diagramas** por meio da combinação de **Blocos** e suas **Conexões**. Cada Bloco representa uma funcionalidade do sistema e é capaz de gerar o trecho de código relacionado a esta funcionalidade onde o código gerado pode ser configurado por meio de Propriedades que definem seu comportamento. As propriedades configuradas desta maneira são chamadas de propriedades estáticas e são definidas em tempo de programação. As Conexões entre os Blocos são feitas por meio de suas Portas e servem para trocar dados entre os trechos de códigos distintos, o que permite uma configuração do código gerado em tempo de execução. Um Diagrama desta ferramenta pode ser visto na Figura 1.

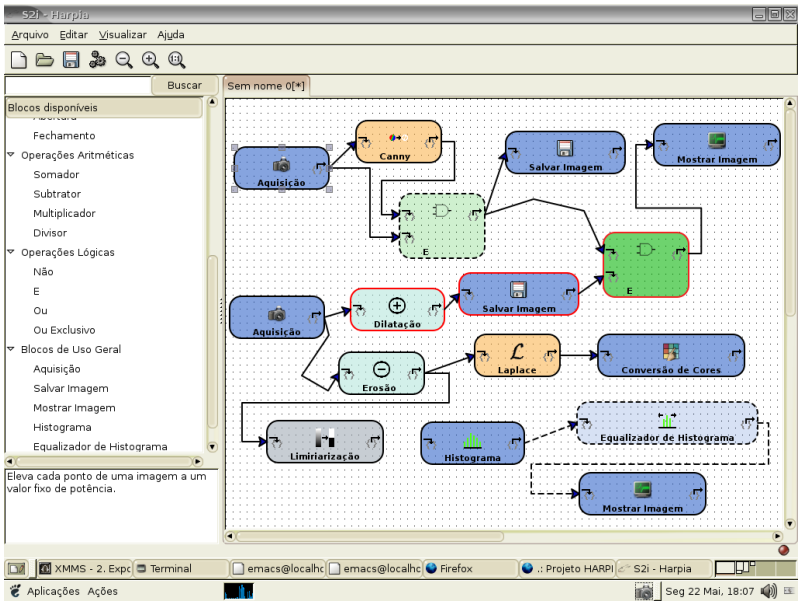

Figura 1. Diagrama feito no ambiente de programação Harpia.

Esta ferramenta possui um grande potencial para o ensino e programação de Processamento Digitais de Imagens além de possuir características claramente atrativas para outras áreas da computação como Ensino de Computação, Linguagens de Programação, Realidade Virtual, Visão computacional, Computação Musical e Arte Digital. Por esta razão, em 2014, pesquisadores do Departamento de Computação da Universidade Federal de São João del-Rei (UFSJ), entraram em contato com o grupo de desenvolvimento inicial, e de posse do código-fonte, iniciaram uma refatoração de código do Harpia com o objetivo de retomar o projeto e torná-lo novamente operante. Esta refatoração do código visava resolver as dependências quebradas e também projetar uma infra-estrutura arquitetural para a ferramenta por meio de um modelo formal para garantir uma melhor integração entre seus componentes [Garlan and Shaw 1994].

2. Evolução da ferramenta

A evolução da ferramenta partiu de, inicialmente, torná-la operante. Além disto, a modificação para corrigir a dependência quebrada serviu para aprimorar o conhecimento sobre o código-fonte e sobre o funcionamento da mesma.

2.1. Mudança na persistência

O Harpia é feito em Python e utiliza para a persistência em XML a biblioteca Amara, que está descontinuada. A modificação desta implementação envolveu uma modificação brusca no código pois a dependência de arquivos XML estava distribuída em diversas classes do projeto, o que tornava o acoplamento desta dependência bastante forte. A refatoração do código e a adoção da nova biblioteca para esta funcionalidade iniciou-se pelo isolamento de todos os métodos de persistência em uma única classe *proxy* responsável pela persistência XML. Esta classe traz para o sistema todas as funcionalidades da nova dependência e todas as classes que trabalham com persistência chamam métodos desta classe do sistema [Gamma et al. 1994].

Além da adoção de uma classe *proxy* como um único ponto de acoplamento com a persistência XML, foi adotada outra biblioteca para substituir a Amara. Neste ponto da refatoração optou-se por escolher a biblioteca Beautiful Soup para este fim por ser uma biblioteca bastante madura e cujo adoção ocorre por diversos outros projetos.

2.2. Refatoração da GUI

A GUI do Harpia é baseada na ferramenta Glade que traz diversas vantagens como o desenvolvimento rápido de componentes e telas por definir as mesmas em arquivos XML. Porém, tal estratégia de desenvolvimento implicava na distribuição dos arquivos XML com as definições de GUI e impedia que telas criadas dinamicamente fossem criadas usando o Glade. Cada Bloco da ferramenta necessitava de um arquivo Glade com a definição de sua janela de Propriedades e a alteração nas propriedades de um Bloco ou a criação de um novo Bloco implicava na utilização de diversas ferramentas. Isto tornava o desenvolvimento de novos blocos trabalhoso e não aproveitava a característica principal do Glade para o desenvolvimento de GUI que é a prototipação rápida. Além disto, o sistema dependia de uma versão antiga e descontinuada da ferramenta Glade.

Decidiu-se neste ponto alterar toda a interface gráfica do sistema para algo mais flexível que o Glade e que permitisse a criação de telas em tempo de execução para a configuração das propriedades dos Blocos. Tinha-se também uma preocupação em a) diminuir a dependência do sistema de uma biblioteca de GUI, b) isolar as classes de GUI de maneira a diminuir o impacto sobre alterações futuras nas mesmas e c) separar a definição dos Blocos e processamentos de sua representação gráfica.

Diante desta decisão, optou-se por utilizar a API Gtk em sua versão 3.22 para a refatoração da interface gráfica do sistema. Para isolar esta dependência analisou-se que a estratégia utilizada para a persistência XML de criar uma classe *proxy* não seria eficiente pois significaria a reescrita de todos os componentes Gtk. Por esta razão, a utilização da API Gtk foi feita por meio de classes que estendem a API Gtk e fornecem uma interface simples para os demais componentes do sistema. Isto tornou as dependências do projeto mais fracas onde uma modificação de um componente implica na modificação de um trecho reduzido de código. Portanto o impacto de futuras modificações como alterações na versão do Gtk foi reduzido. A nova GUI pode ser vista na Figura 2.

2.3. Refatoração das extensões e Mudança na Geração de código

O conjunto inicial de Blocos da ferramenta Harpia era focados no domínio de visão computacional e era baseado na biblioteca OpenCV em sua versão 2.4. Esta versão de biblioteca também estava descontinuada sendo que o padrão atual da mesma é a versão 3.4 e os códigos são incompatíveis entre estas versões. Para permitir que o código gerado fosse atualizado para esta versão da biblioteca, todos os Blocos definidos na ferramenta tiveram que ser reescritos para a versão mais nova do OpenCV.

A definição de um Bloco neste momento dependia de: a) um arquivo Glade para sua tela de propriedades, b) um arquivo XML para definir a ajuda, c) um arquivo de imagem para seu ícone, d) um arquivo Python para fazer a conexão entre Glade, XML e a ferramenta e e) a alteração do gerador de código da ferramenta. Os Bloco da ferramenta estavam definidos internamente no código da mesma e a criação de novos Blocos com novas funcionalidades dependia de uma alteração no código fonte do próprio Harpia.

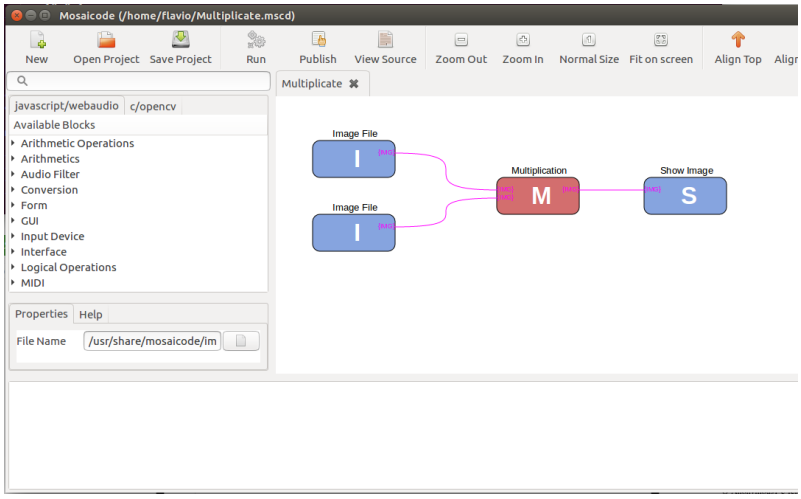

Figura 2. Diagrama feito no ambiente de programação Mosaicode.

Para simplificar a evolução da ferramenta e garantir uma maior flexibilidade e adequação do código gerado, a ferramenta foi alterada de maneira a desassociar o código dos blocos de programação do código da ferramenta.

Como estratégia de simplificar a alteração e criação de novos Blocos, neste momento optou-se por não mais ter ícones nos Blocos mas apenas uma identificação visual por cor e uma letra. Também optou-se por definir no próprio Bloco sua Ajuda, Propriedades, Portas e Código a ser gerado por este bloco. Isto reduziu a definição de um Bloco para uma única classe Python contendo campos definidos por dicionários e nenhuma método em especial. As Portas de um Bloco também foram definidas em uma classe Python contendo tanto sua definição quanto sua geração de código. Por fim, definiu-se um Padrão de código que significa a estrutura do código-fonte a ser gerado por um conjunto de blocos.

O modelo inicial de geração de código utilizava a concatenação de Textos para a montagem do código final. Por esta razão, a mesma dependia de códigos específicos para concatenar as propriedades dos Blocos. Nesta mudança, adotou-se um modelo de geração de código baseado em substituição de palavras-chaves, como no Framework Jakarta Velocity [Harrop 2004], permitindo com isto que o modelo de código gerado fosse desassociado do código da ferramenta e pudesse ser alterado sem a modificação do código-fonte da mesma. Isto permitiu que a ferramenta sofresse alterações em seus pontos de variação e que tornou possível desassociar o Harpia da geração de código para o domínio específico de Visão Computacional ou mesmo para a linguagem C.

2.4. Criação de um Meta-Modelo

Com a mudança no modo de representar os Blocos, a Geração de código na ferramenta passou a ser definidos de maneira abstrata por um conjunto de **Classes de Modelo**. Diagrama, Bloco, Conexão, Porta, Propriedade e Padrão de código passaram a compor um conjunto de classes com o Meta-Modelo do Sistema, o que permitiu modificar o sistema em tempo de execução adicionando ou removendo instâncias destas classes. Com isto, o sistema passou a trabalhar com extensões de seu Meta-modelo.

Tendo um meta-modelo comum para a representação da GUI ou mesmo para a

persistência, foi possível definir extensões para o ambiente como um conjunto novo de Blocos, Portas e Padrões de Código partindo de um meta-modelo genérico que pode ser representado tanto como classes Python quanto como arquivos XML. Foi adicionado ao sistema a capacidade de carregar extensões em tempo de execução permitindo assim desassociar a ferramenta de suas extensões e torná-la uma ferramenta genérica para geração de código baseada em modelos. O carregamento da ferramenta se dá com o carregamento das extensões na seguinte ordem: Classes Python instaladas com o sistema, Arquivos XML instalados com o sistema e Arquivos XML no espaço de usuário. Assim, se o usuário quiser alterar e personalizar Blocos em sua instância da ferramenta, o mesmo consegue fazer de maneira a alterar a instalação da ferramenta apenas para seu usuário e sem depender de acesso especial para isto.

3. Mudança arquitetural e o surgimento do Mosaicode

Para completar a modificação do sistema na refatoração da GUI, foram criadas também classes de controle para evitar que o sistema dependa totalmente das GUIs para seu funcionamento permitindo que seja possível realizar diversas tarefas passando argumentos ao programa em linha de comando. Tanto GUI quanto Persistência passaram a ter dependência mais fraca de bibliotecas externas. Para isto, foi utilizado o padrão MVC (*model, view and control*) para a refatoração do código, separando o desenvolvimento do software em camadas e facilitando a alteração da interface gráfica sem alterar a parte funcional da aplicação [Krasner et al. 1988], conforme ilustrado pela Figura 3.

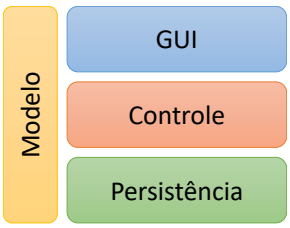

Figura 3. Arquitetura da ferramenta inspirada no modelo em camadas e na arquitetura MVC.

A **Camada de Persistência** é responsável por salvar e carregar os Diagramas feitos na ferramenta e também por carregar e persistir as extensões da ferramenta (Blocos, Propriedades, Portas e Padrões de código). Com isto, toda a persistência XML da ferramenta está isolada nesta camada, o que permite alterar facilmente esta dependência futuramente. A camada de persistência utiliza a camada de Modelo e seus métodos são chamados exclusivamente pela camada de Controle.

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

A **Camada de GUI** define a interface de usuário e é baseada no GTK versão 3. Todos os componentes gráficos do ambiente estendem uma classe Gtk e especializa esta

classe para as necessidades da ferramenta. Assim, temos classes Menu, Janela Principal, Barra de Ferramentas e assim por diante. Os componentes gráficos como Blocos, Conexões e Diagramas baseiam-se na biblioteca Goocanvas. A configuração de propriedades de um Bloco é genérica e configurável a partir da descrição de Propriedades de cada Bloco.

A **Camada de Modelo** representa os componentes do sistema como Diagrama, Porta, Conexões, Blocos e Padrão de Código. Um Bloco pode possuir Propriedades, Portas de Entrada e de Saída e uma Conexão será obrigatoriamente uma relação entre um Bloco origem (Source) e um Bloco destino (Sink) e suas respectivas portas.

Neste ponto da refatoração pudemos notar que esta versão da ferramenta seguia as premissas do Harpia mas já não tinha mais código em comum com a ferramenta inicial e também não se destinava mais ao domínio exclusivo da Visão computacional. Por esta razão, a mesma foi rebatizada com o nome de **Mosaicode**, apresentada na Figura 2.

Além de redesenhar a ferramenta e torná-la novamente operante, a evolução do Harpia para o Mosaicode contou também com a adição de novas funcionalidades como a capacidade de alinhar os blocos, novo layout de Blocos e Conexões e a documentação, ajuda de blocos por diagramas de exemplo e extensão do Meta-modelo e adição de comentários no Diagrama.

Outra funcionalidade adicionada foi a incorporação de um servidor Web que permite publicar o código gerado localmente para simplificar a colaboração e cooperação em criações colaborativas.

3.1. Extensões do Mosaicode

Uma vez que o ambiente de programação Mosaicode permite a utilização de novas extensões, alguns projetos de desenvolvimento em paralelo passaram a ocorrer para criar neste ambiente um ferramental adequado para o desenvolvimento de aplicações voltadas para o domínio de Arte Digital, Realidade Virtual e Visão Computacional.

- **Web Art:** Foi criada uma extensão para a criação de WebArt e Computação Musical baseada na linguagem Javascript e na biblioteca webaudio do HTML5. Esta extensão envolve Blocos de HTML reativos conectados por meio do padrão de projeto Observador. Estes Blocos possuem diversas funcionalidades para síntese musical, criação de sons, Criação de formulários HTML, Criação de elementos gráficos para a web, captura de sensores por páginas em dispositivos móveis, síntese de imagem com SVG e Canvas, entre outras [Gomes et al. 2019].
- **Síntese de imagens:** Baseado na biblioteca open source OpenGL na linguagem C, criou-se uma extensão para a síntese de imagem em 2D e 3D. Esta extensão permite a definição de elementos básicos da síntese de imagens, como Quadrado, Cubo, Círculo, Reta, Esfera, e também a transformação destes elementos como a escala, rotação e translação [Gomes 2019].
- **Síntese de som:** Baseado nas bibliotecas PortAudio e LibSoundFile foi criada uma extensão para processamento e síntese de som. Esta extensão possui Blocos como ruído branco, osciladores, operadores aritméticos de sinal, Ganho, entre outros e utiliza a biblioteca PortAudio para a captura e reprodução de áudio. Além disto, utiliza a biblioteca Libsoundfile para a leitura e escrita de arquivos de som [Luiz Gonçalves 2017].

- **Processamento de Imagens e Visão computacional:** Como relatado anteriormente, foi criado uma extensão para processamento de Imagem e Visão computacional baseado na biblioteca OpenCV 3.4 na linguagem C++. Esta biblioteca possui Blocos para captura de imagens por câmera ou por arquivos e também diversas operações aritméticas ou morfológicas de imagens [Resende 2019].
- **Outras extensões:** Encontra-se em desenvolvimento uma extensão para a criação de GUIs baseada na linguagem C++ e na biblioteca Gtk+3. Também está sendo desenvolvido uma extensão para a comunicação em rede e uma extensão para a implementação de controladores físicos como controladores MIDI e joysticks.

3.2. Estendendo o ambiente por plugins

Além de permitir que o ambiente seja modificado por seus usuários por meio da modificação e criação de novas extensões, foi criado também um modelo de extensão do próprio ambiente por meio de *plugins*[Syeed et al. 2015]. Um *plugin* do Mosaicode é um trecho de código distribuído a parte do ambiente de programação que altera o comportamento do mesmo adicionando novas funcionalidades. A API de *plugin* do Mosaicode permite que um *plugin* crie novos Menus, instale-se na Barra de Ferramentas do sistema e tenha acesso a todas as classes da camada de controle do ambiente.

Para isto, as classes de controle foram modificadas e diversos métodos de notificação de alteração do ambiente foram transformados para funcionar como o padrão de projeto Observador de maneira que os controles do Mosaicode consigam notificar qualquer classe interessada em saber de qualquer modificação do ambiente. Também foi criado um Carregador de novos *plugins* e um primeiro *plugin* do ambiente que permite ao usuário adicionar, alterar e remover Blocos, Portas e Padrões de código.

3.3. Adoção de ferramentas e tecnologias

Para garantir a manutenção e evolução desta ferramenta, algumas ferramentas e tecnologias foram adotadas e incorporadas no desenvolvimento da mesma.

- **Código-fonte disponível:** Desde o início deste trabalho, todas as modificações no projeto são mantidas no repositório Github e as Extensões e *Plugins* são mantidos em repositórios separados do código-fonte do ambiente.
- **Testes Unitários:** Foi adotado o pytest para a criação de testes unitários para todo o código da ferramenta, seus plugins e extensões.
- **Documentação online:** A ferramenta, plugins e extensões estão com seus códigos documentados utilizando a ferramenta sphinx e o padrão docstring.
- **Padrão de código:** O desenvolvimento também segue o padrão de código PEP8 e utiliza a ferramenta coverage para verificar se o sistema está íntegro, documentado e testado antes do lançamento de uma nova versão.
- **Empacotamento e distribuição:** a ferramenta Mosaicode, suas extensões e plugins estão empacotadas disponíveis no repositório pypi.
- **Internacionalização**: o módulo gettext foi utilizado para definir as strings contidas na ferramenta Mosaicode, oferecendo o suporte a mais de um idioma.

4. Lições aprendidas

Quando procurou-se a ferramenta Harpia no repositório Ubuntu / Debian esperava-se ter encontrado-a disponível e operante e foi uma certa surpresa ter descoberto que a mesma

estava obsoleta e que esta obsolescência tinha sido causada pela obsolescência de uma de suas dependências. Esta talvez tenha sido a primeira lição aprendida. Muitas vezes evita-se "reinventar a roda" e reescrever um código que já existe e está disponível em alguma biblioteca, framework ou componente. No entanto, nem sempre atenta-se que adotar uma biblioteca significa acoplar um código de terceiro e depender da existência desta biblioteca para garantir a longevidade do próprio *software*. Certamente esta lição não indica que sempre deve-se desenvolver novo código que já existe mas que deve-se sempre 1) avaliar a longevidade da biblioteca antes de criar a dependência e 2) tentar garantir um baixo acoplamento para esta dependência de maneira que a modificação da dependência e alteração do código não resulte em outra ferramenta, como aconteceu neste nosso exemplo. Certamente é necessário avaliar o **impacto que uma dependência** pode causar na evolução do Sistema antes de adotar a biblioteca como dependência.

Outra lição aprendida é que a manutenção de um sistema nem sempre ocorre por falha de programação, erro de código ou obsolescência do sistema. Manutenções ocorrem pois um software é um sistema complexo que costuma ter diversos componentes cuja manutenção depende da manutenção de todos os componentes do mesmo. Além disto, as necessidades dos usuários podem mudar ao longo do tempo e isto pode criar novos requisitos no sistema, requisitos estes que demandam manutenção. Por esta razão, pensar o desenvolvimento já tendo como certeza que o sistema irá precisar de manutenção constante implica em adotar diversas estratégias, de documentação à testes, para simplificar manutenções e garantir a longevidade do sistema.

5. Conclusão

Este artigo apresentou a refatoração do código e evolução do ambiente de programação visual Harpia. Na refatoração do código, a ferramenta foi reescrita e sua arquitetura foi modificada de maneira a trabalhar com extensões e *plugins*. Após isto, novas extensões foram desenvolvidas para atender outros domínios de aplicação além da Visão Computacional e se estendendo para o domínio da Arte Digital, Computação Musical, Computação Gráfica, Processamento Digital de Imagens, Realidade Virtual e Aumentada[Luiz Gonçalves 2017, Gonçalves and Schiavoni 2019, Gomes et al. 2019]. Isto permitiu isolar a geração de código da ferramenta inicial e adicionar à mesma outros domínios de aplicação e outras linguagens de programação. Após a ramificação, a ferramenta foi renomeada passando a se chamar Mosaicode.

No momento, o desenvolvimento deste ambiente tem se mostrado bastante frutífero e motivador para fomentar o desenvolvimento tecnológico em nosso grupo de pesquisa. Os primeiros testes feitos com usuários [Schiavoni and Gonçalves 2017] apontam que este ambiente auxilia na criação de aplicações para estes domínios de aplicação. Ultrapassando a barreira do desenvolvimento tecnológico, este projeto tem servido como objeto para investigação acadêmica e científica nas áreas interdisciplinares em que o projeto está inserido. Ao todo, em seis anos de projeto, já soma-se mais de 20 artigos publicados em conferências e em revistas acadêmicas / científicas além de relatórios, monografias e dissertações. Este trabalho tem envolvido alunos de graduação e mestrado em projetos de Iniciação Científica e trabalhos de conclusão de curso na área de Computação e mais recentemente na área de criação artística em cursos como Artes Aplicadas, Música e Artes Cênicas.

Os autores gostariam de agradecer ao CNPq (151975/2019-1), a FAPEMIG (APQ-02148-18), a Universidade Federal de São João del-Rei e aos membros do laboratório ALICE – Arts Lab in Interfaces, Computers, Education and Else – pelo apoio a esta pesquisa.

O repositório com o código do ambiente Mosaicode e mais informações sobre o projeto podem ser encontrados em `https://mosaicode.github.io/`.

Referências

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). Design patterns: elements of.

Garlan, D. and Shaw, M. (1994). An introduction to software architecture. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA.

Gomes, A., Resende, F., Gonçalves, L., and Schiavoni, F. (2019). Prototyping web instruments with mosaicode. In Schiavoni, F., Tavares, T., Constante, R., and Rossi, R., editors, *Proceedings of the 17th Brazilian Symposium on Computer Music*, pages 114–120, São João del-Rei - MG - Brazil. Sociedade Brasileira de Computação.

Gomes, A. L. N. (2019). *Extensão de Síntese de Imagens no Mosaicode para Arte Digital*. Monografia (bacharelado em ciência da computação), Universidade Federal de São João del-Rei.

Gonçalves, L. and Schiavoni, F. (2019). The development of libmosaic-sound: a library for sound design and an extension for the mosaicode programming environment. In Schiavoni, F., Tavares, T., Constante, R., and Rossi, R., editors, *Proceedings of the 17th Brazilian Symposium on Computer Music*, pages 99–105, São João del-Rei - MG - Brazil. Sociedade Brasileira de Computação.

Harrop, R. (2004). *Pro Jakarta Velocity: From Professional to Expert*. Apress.

Krasner, G. E., Pope, S. T., et al. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49.

Luiz Gonçalves, L. (2017). Sound design com o mosaicode. Monografia (bacharelado em ciência da computação), Universidade Federal de São João del-Rei.

Resende, F. R. (2019). *Processamento Digital de Imagens e Visão Computacional no ambiente Mosaicode*. Monografia (bacharelado em ciência da computação), Universidade Federal de São João del-Rei.

Schiavoni, F. L. and Gonçalves, L. L. (2017). From virtual reality to digital arts with mosaicode. In *2017 19th Symposium on Virtual and Augmented Reality (SVR)*, pages 200–206, Curitiba - PR - Brazil.

Syeed, M. M. M., Lokhman, A., Mikkonen, T., and Hammouda, I. (2015). Pluggable systems as architectural pattern: An ecosystemability perspective. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, pages 42:1–42:6, New York, NY, USA. ACM.