Integração de DAW com Infraestruturas Remotas de Processamento de Áudio

Carlos E. C. F. Batista
LAViD/UFPB
João Pessoa, Brasil
bidu@lavid.ufpb.br

Gabriel B. F. Andrade
LAViD/UFPB
João Pessoa, Brasil
gabriel.ferraz@lavid.ufpb.br

Geovana M. S. Lima
LAViD/UFPB
João Pessoa, Brasil
geovana.lima@lavid.ufpb.br

Caio M. C. Guedes
Music.AI
João Pessoa, Brasil
caio.marcelo@moises.ai

ABSTRACT

This article introduces Orchestrator Pad, an open-source audio processing plug-in built using JUCE, designed to facilitate access to the capabilities of the Music.AI platform. This service offers advanced features such as stem separation, automatic mastering, and various other forms of audio manipulation through a REST API. It supports the creation of custom workflows that integrate its services, seamlessly invoked by the plug-in using a C++ wrapper library developed in parallel. After remote audio processing, the resulting files can be played within any compatible DAW through the plug-in. This paper details the plug-in's architecture and core functionalities and discusses the advantages of incorporating AI-driven workflows into music production environments. The project repository provides the necessary code and configuration for building the Orchestrator Pad plug-in as either a standalone application or a VST plug-in.

KEYWORDS

Plug-in VST, Processamento de áudio remoto, Biblioteca C++, Ferramentas de código aberto

1 INTRODUÇÃO

No processo de produção musical contemporânea, as *Digital Audio Workstations* (DAW) desempenham um papel central ao proporcionar um ambiente integrado para gravação, edição, mixagem e masterização de áudio. Produtores utilizam uma variedade de *plug-ins*, baseados em formatos como VST, AU, AAX [1], para expandir as capacidades das DAW, incluindo efeitos, instrumentos virtuais e ferramentas de processamento avançadas. Hoje um produtor necessita não apenas de uma DAW robusta, mas também de um conjunto diversificado de *plug-ins* que atendam às suas necessidades criativas e técnicas.

Desde o surgimento dos primeiros padrões de desenvolvimento de *plug-ins* de áudio [1], houve uma evolução significativa na forma como esses *plug-ins* são desenvolvidos, distribuídos e instalados, com plataformas digitais oferecendo acesso imediato e atualizações automáticas, através de serviços com autenticação e canais de comunicação seguros.

In: XXII Workshop de Ferramentas e Aplicações (WFA 2024). Anais Estendidos do XXX Simpósio Brasileiro de Sistemas Multimídia e Web (WFA'2024). Juiz de Fora/MG, Brazil. Porto Alegre: Brazilian Computer Society, 2024.

ISSN: 2596-1683

Gradualmente surgiram plataformas onde os *plug-ins* não apenas eram distribuídos e validados remotamente, mas também tinham parte de sua execução realizada em servidores remotos. Esta tendência tem se intensificado com a emergência de serviços de manipulação de áudio remoto baseados em aprendizado de máquina e outras técnicas computacionalmente intensivas. Estes serviços proporcionam funcionalidades avançadas através de plataformas acessíveis via web, aplicativos dedicados, *plug-ins* e API, aumentando a flexibilidade e a eficiência para os usuários. Plataformas como Music.AI [2], BandLab [3], LALAL.AI [4] e outras oferecem processamento remoto de áudio, permitindo que usuários acessem funcionalidades avançadas de manipulação sonora fornecidas por servidores dedicados.

O presente artigo apresenta o *Orchestrator Pad*, um *plug-in* VST de código aberto desenvolvido usando o framework JUCE [5], com o objetivo de facilitar o acesso às capacidades da API REST da plataforma Music.AI [2]. Esta plataforma oferece uma série de funcionalidades avançadas para análise e manipulação de áudio, incluindo separação de *stems* (*demixing*), redução de ruído de fundo, *upsampling*, masterização automática, transcrição de letras etc. A separação de *stems*, por exemplo, permite isolar diferentes elementos de uma faixa de áudio, como vocais, bateria e instrumentos. Tais funcionalidades podem ser integradas para definir *workflows* (fluxos de trabalho) usados para processar arquivos de áudio de entrada, gerando um ou mais arquivos de áudio como resultado [2].

Para dar suporte ao desenvolvimento do *plug-in* e proporcionar uma nova abordagem para acessar as funcionalidades do Music.AI, foi desenvolvida uma API C++. Esta escolha se fundamenta na posição dominante da linguagem na indústria para o desenvolvimento de *plug-ins* de áudio, onde a performance e o controle sobre os recursos de *hardware* são importantes [1].

O *Orchestrator Pad* se posiciona como um robusto código base para o desenvolvimento de aplicações e *plug-ins* de áudio que explorem os recursos da plataforma Music.AI. O código é disponibilizado sob a licença MIT, garantindo liberdade para modificações e distribuições futuras. Este trabalho detalha a arquitetura do *plug-in*, suas funcionalidades principais e explora os benefícios de se integrar fluxos de trabalho baseados em aprendizagem de máquina nos ambientes de produção musical.

Este artigo está estruturado da seguinte forma: a seguir é detalhada a API C++ que realiza a interface com os *endpoints* oferecidos pela API REST da plataforma Music.AI. A terceira

seção descreve o *plug-in Orchestrator Pad*, abordando sua arquitetura de *software* e principais funcionalidades. A última seção oferece conclusões relacionadas ao desenvolvimento e teste do *plug-in* e discute novas funcionalidades para versões futuras.

2 API C++

A API C++ foi desenvolvida com o intuito de facilitar o acesso aos recursos da plataforma Music.AI. Ela encapsula as requisições REST e retorna objetos que contém as informações recebidas.

O *CMake* [6] foi empregado para configurar parâmetros de compilação e gerenciar dependências do projeto de software, assegurando sua portabilidade. A biblioteca *cpr* [7] foi escolhida por sua capacidade de realizar requisições HTTP de forma simples e eficaz, essencial para a comunicação com a API REST da plataforma Music.AI. Finalmente, a biblioteca *nlohmann-json* [8] foi fundamental para manipular e gerenciar dados no formato JSON, para o tratamento de informações recebidas e enviadas pela API.

Para evitar dependências excessivas no uso de JSON fora do escopo da API, foi necessário mapear e abstrair os dados recebidos para formatos que melhor representassem as informações usando recursos nativos do C++. As classes desenvolvidas incluem:

- *Application*: informações relacionadas à chave de acesso utilizada, para identificação do usuário solicitante em todas as requisições.
- *Workflow*: representa um *workflow* (fluxo de trabalho), com ID, nome, slug e descrição. Um *workflow* pode ser criado na plataforma Music.AI a partir da integração dos múltiplos serviços oferecidos em uma sequência que pode resultar em um ou mais arquivos de áudio.
- *WorkflowParams*: encapsula a URL de entrada de um *workflow*.
- *Job* (tarefa): encapsula um processo na plataforma, contendo informações como entrada de dados, o *workflow* a ser utilizado no processamento e as saídas geradas.
- *Load*: armazena URLs para *upload* e *download* de arquivos.
- *Metadata*: armazena os metadados de uma tarefa.
- *RequestResult*: armazena os resultados do processamento de uma tarefa.
- *JSONUtils*: auxilia na conversão entre JSON e as classes criadas.
- *MusicAIException*: exceções lançadas em caso de falhas nas requisições.

Por fim, a classe MusicAI integra a lógica central da API, oferecendo funções como:

- *getApplication*: obtém informações sobre a chave de acesso em uso.
- *getUpload*: recupera URLs para upload e download de arquivos.
- *getWorkflows*: obtém todos os *workflows* (ou um número específico) vinculados a uma chave.
- *createJob*: cria uma nova tarefa.
- *getJob*: obtém uma tarefa específica.
- *getJobs*: obtém todas as tarefas associadas à chave de acesso.
- *deleteJob*: exclui uma tarefa.
- *uploadFile*: envia um arquivo de áudio para o servidor.
- *getSession*: cria uma sessão para verificar uma tarefa.
- *monitorJob*: verifica o status de uma tarefa.
- *downloadFromJob*: baixa os resultados de uma tarefa para um diretório.
- *downloadFromResult*: baixa os resultados de um objeto *RequestResult* para um diretório.

A API C++ padroniza e simplifica significativamente a interação com a plataforma ao encapsular as requisições possíveis e definir objetos que representam as informações de resultado. A proposta se destaca em um cenário onde poucos serviços disponibilizam API para manipulação de áudio utilizando processamento remoto. Alguns dos serviços que oferecem API C++ focam em funções mais específicas e limitadas, como otimização de áudio e tratamento de voz - o krisp.ai [9] se especializa na remoção de ruído de fundo em chamadas e gravações, enquanto o sonicAPI.com [10] oferece funcionalidades para processamento de áudio, como transformação de texto em fala e reconhecimento de voz. Em contraste, a API C++ desenvolvida para o Music.AI se distingue por possibilitar o acesso a um conjunto mais amplo e versátil de ferramentas de manipulação de áudio.

3 ORCHESTRATOR PAD

O *plug-in Orchestrator Pad* (Figura 1) foi desenvolvido utilizando a API C++ apresentada na seção anterior e o framework JUCE [5], que é amplamente utilizado para criação de *plug-ins* de áudio. A arquitetura do *plug-in* é composta por várias classes que desempenham funções específicas, como a interface gráfica do usuário, manipulação de áudio e interação com a API Music.AI.

As principais classes que compõem a arquitetura do *plug-in* são:

- *AudioProcessor*: é responsável pelo processamento de áudio - implementação padrão do JUCE para roteamento dos fluxos de áudio entre o *plug-in* e a DAW.
- *AudioProcessorEditor*: é a principal responsável pela interface do usuário. Esta classe é responsável por inicializar e gerenciar todos os componentes visuais do *plug-in*.
- *ConfigWindow*: é uma janela de diálogo usada para configurar a chave da API. Ela exibe campos de entrada e botões para salvar e cancelar a configuração. A chave da API é salva em um arquivo chamado *music_ai.apikey* no diretório de trabalho atual.

- *ContentComponent*: é responsável pela reprodução dos arquivos de áudio gerados pelo processamento da tarefa. Ela utiliza *juce::AudioAppComponent* para gerenciar o áudio e *juce::AudioTransportSource* para controlar a reprodução. O estado da reprodução é gerenciado pela enumeração *TransportState*, que inclui estados como *Stopped, Playing, Paused*, etc. A classe também implementa métodos para manipulação de volume e panorama, além de controle de progresso.

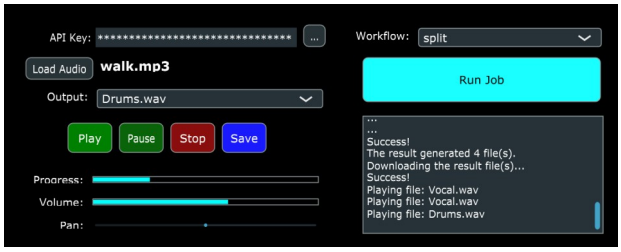

Figura 1: **Interface gráfica do *Orchestrator pad***

A partir da Figura 1 identificam-se os elementos da interface do *plug-in Orchestrator pad* e então discutir como suas funcionalidades são exploradas. Os seguintes elementos são apresentados:

- Chave de Acesso (*API Key*): Permite inserir uma chave de acesso para autenticação com a API do Music.AI. A chave é definida em uma janela exibida através do botão "...".
- Botão "*Load Audio*": Permite carregar um arquivo de áudio (no exemplo, "*walk.mp3*") para processamento.
- Seleção de *Workflow*: Permite selecionar um dos *workflows* vinculados à conta da chave utilizada, para utilização na criação de uma tarefa.
- Botão "*Run Job*": Habilitado após a carga do arquivo de áudio a ser processado, inicia a execução nos servidores da plataforma de uma tarefa onde o *workflow* selecionado será aplicado ao arquivo carregado.
- Seleção de Saída: Habilitado após a realização da tarefa, permite selecionar um dos arquivos de saída da execução. No exemplo, "*Drums.wav*" está selecionado como o arquivo de saída.
- Botões de Controle de Reprodução: Habilitados após a realização da tarefa, são botões para reproduzir, pausar, parar e salvar o áudio processado.
- Barra de Progresso: Permite visualizar e definir o ponto de reprodução da saída selecionada.
- Barra de Volume: Permite definir o volume de reprodução da saída selecionada.
- Barra de Panorama: Permite definir o panorama (distribuição nos canais estéreo) da reprodução da saída selecionada.
- Console de Status: Exibe mensagens de estado da realização da tarefa e da reprodução dos arquivos de resultado.

3.1 Usando o *plug-in*

O *Orchestrator Pad* foi desenvolvido principalmente para ser utilizado em conjunto com uma DAW (Figura 2), funcionando como um *plug-in*, conforme explicado anteriormente. As opções de compilação oferecidas pelo projeto de *software* permitem a criação não só de um *plug-in* VST3, mas também de uma aplicação autônoma, capaz de ser executada em múltiplas plataformas. Apesar dessa flexibilidade, o uso como *plug-in* em uma DAW é considerado o cenário padrão para a ferramenta. Portanto, este contexto será utilizado para apresentar a jornada do usuário.

Para usar o *plug-in Orchestrator Pad*, o usuário primeiro deve carregar e selecionar o *plug-in* dentro de uma DAW (compatível com VST3 [1]), vinculando-o a uma trilha de um projeto. Com o *plug-in* carregado, o usuário poderá definir a chave de acesso para a API da plataforma (que é salva). A manipulação parte do carregamento de um arquivo de áudio local - a atual versão ainda não permite o tratamento de fluxos de áudio vindo da DAW para o *plug-in*.

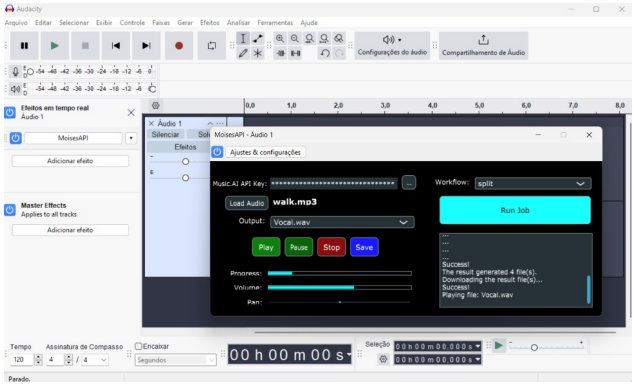

Figura 2: ***Orchestrator pad* usado em conjunto com a DAW Audacity.**

Após o carregamento do arquivo, o usuário precisa selecionar um dos *workflows* disponíveis. Esses *workflows* estão vinculados à conta do usuário na plataforma Music.AI e são configurados previamente. Cada *workflow* integra um conjunto de funcionalidades específicas de processamento de áudio, como separação de *stems*, transcrição de letras, masterização automática ou otimização vocal. A atual versão do *plug-in* trata apenas *workflows* que geram saídas no formato de arquivo de áudio.

Com o arquivo de áudio carregado e o *workflow* escolhido, o usuário pressiona o botão "*Run Job*" para iniciar a tarefa. O *plug-in* então envia as informações para a plataforma realizar o

processamento de acordo com o que é definido no *workflow*. Durante esse processo, o console do *plug-in* exibe informações sobre o estado da execução.

Depois da conclusão do processamento, a interface ativa o componente que permite a seleção dos arquivos de áudio gerados (saída). Os controles *play*, *pause* e *stop* possibilitam a manipulação da reprodução do áudio selecionado. O usuário também tem a opção de salvar o arquivo de áudio selecionado em uma pasta externa.

Durante a reprodução, os usuários podem visualizar e ajustar o progresso, volume e o panorama utilizando os sliders oferecidos pelo *plug-in*. O resultado da reprodução é vinculado à trilha associada ao *plug-in* na DAW (Figura 2), podendo ser posteriormente manipulado e reproduzido utilizando os recursos da própria DAW e outros *plug-ins*. O vídeo de teste do *Orchestrator pad*[1] apresenta a visualização completa do processo de uso do plug-in.

4 CONCLUSÕES E TRABALHOS FUTUROS

O *Orchestrator Pad* é um *plug-in* que viabiliza a integração das funcionalidades da API Music.AI dentro das ferramentas de edição de áudio. Os testes realizados demonstraram que o *plug-in* opera com sucesso tanto em modo autônomo quanto integrado a uma DAW, garantindo sua versatilidade e eficácia em diferentes cenários de uso.

Os códigos-fonte da API em C++ e do *plug-in* estão disponíveis em um repositório público no GitHub[2], sob uma licença de código aberto. Isso permite que desenvolvedores e entusiastas contribuam para o contínuo aprimoramento da ferramenta. O desenvolvimento do *Orchestrator Pad* continua após o lançamento inicial, com a incorporação de novas funcionalidades e melhorias.

Para as próximas versões do *Orchestrator Pad*, são planejadas várias melhorias e novas funcionalidades que visam otimizar a experiência do usuário e ampliar as capacidades do *plug-in*. Uma nova interface gráfica que incorpora de forma mais intuitiva e eficiente o fluxo de trabalho utilizado na manipulação do *plug-in* está em desenvolvimento. Esta interface redesenhada facilitará a navegação e a utilização das funcionalidades oferecidas, proporcionando uma experiência de usuário mais agradável e produtiva. Outra melhoria significativa será a implementação da funcionalidade de arrastar e soltar (*drag and drop*) tanto para a entrada quanto para a saída de arquivos de áudio.

Será desenvolvida a capacidade de tratamento de fluxos de áudio de entrada diretamente da DAW, com comunicação em tempo real com a API REST usando *streaming* de áudio. Por fim, serão adicionadas funcionalidades de visualização e edição dos *workflows* diretamente no *plug-in*, oferecendo aos usuários uma flexibilidade maior para configurar e ajustar seus processos de áudio de acordo com suas necessidades específicas.

O processamento remoto de áudio tem impulsionado ferramentas cada vez mais relevantes para a produção musical moderna, permitindo a execução de tarefas computacionalmente intensivas em ambientes remotos, o que otimiza o fluxo de trabalho e expande as possibilidades criativas.

Uma parte essencial do projeto é a análise contínua do estado da arte, abrangendo desde sistemas mais antigos, como o *AudioGridder* [11], que foca na distribuição da execução de plugins, passando por sistemas baseados em *WebAudio* [12], até as plataformas mais avançadas, como o HARP [13] que realiza processamento de *deep learning* remoto. Este estudo tem como objetivo fornecer uma visão mais precisa sobre o uso do processamento remoto na produção musical e a estruturação dessas soluções sob a perspectiva da arquitetura de software. Já há uma compreensão inicial de que essas soluções são bastante heterogêneas, refletindo a diversidade de funcionalidades disponíveis. Parte dessa análise e compreensão será consolidada em um artigo, que apresentará um panorama abrangente sobre o tema.

1 Vídeo de demonstração acessível através da URL: https://tinyurl.com/opadvideo

2 https://github.com/moiseslabs/orchestrator-pad/

REFERÊNCIAS

[1] PIRKLE, Will. Designing audio effect plugins in C++: for AAX, AU, and VST3 with DSP theory. Routledge, 2019.

[2] Music.AI: Advanced AI Audio Processing. Disponível em: <https://music.ai/>. Acesso em Julho de 2024.

[3] BANDLAB. Disponível em: <https://www.bandlab.com/>. Acesso em Julho de 2024.

[4] LALAL.AI. Tools and API. Disponível em: <https://www.lalal.ai/pt-br/tools-and-api/>. Acesso em Julho de 2024.

[5] JUCE: The Cross-Platform C++ Framework. Disponível em: <https://juce.com/>. Acesso em Julho de 2024.

[6] CMake: Cross-Platform Make. Disponível em: <https://cmake.org/>. Acesso em Julho de 2024.

[7] CRAWL, Kevin. cpr - C++ Requests: Curl for People, a spiritual port of Python Requests. Disponível em: <https://github.com/libcpr/cpr>. Acesso em Julho de 2024.

[8] LOHMANN, Niels. JSON for Modern C++. Disponível em: <https://github.com/nlohmann/json>. Acesso em Julho de 2024.

[9] SONICAPI. Documentação. Disponível em: https://sonicapi.com/docs. Acesso em Julho de 2024.

[10] KRISP.AI. Documentação do SDK. Disponível em: https://sdk-docs.krisp.ai/docs/getting-started. Acesso em Julho de 2024.

[11] AUDIOGRIDDER. AudioGridder. Disponível em: https://audiogridder.com. Acesso em Julho de 2024.

[12] REN, Shihong; LETZ, Stephane; ORLAREY, Yann; MICHON, Romain; FOBER, Dominique et al. Using Faust DSL to Develop Custom, Sample Accurate DSP Code and Audio Plugins for the Web Browser. Journal of the Audio Engineering Society, v. 68, n. 10, p. 703-716, 2020.

[13] GARCIA, Hugo Flores et al. HARP: Bringing Deep Learning to the DAW with Hosted, Asynchronous, Remote Processing. 37th Conference on Neural Information Processing Systems (NeurIPS 2023)