Extending Open-Source Audio Embedded System: A Pedal Pi Components Study Case

P. da Silva, L. Costa, R. Pereira, and S. Jucá

***Abstract*—Audio plugins processing equipment that serve beginner instrumentalists have limitations regarding of extension for physical control and integration with other equipment, limiting the execution and the creation performances. Open-source projects allow their adaptation or extension by modifying the code or by integrating components that comply with some specification. Pedal Pi, a low-cost open-source DIY project, enables the development of software components for producing HMIs through a specified API. In this paper, components are developed that allow the control of the Pedal Pi by means of applications for mobile devices. For this, a bibliographic review of open-source solutions for the embedded processing of audio plugins is performed, an analysis of the extension points offered by the Pedal Pi software architecture is realized, the control components developed are presented and are discussed the results and the contribution relationships among the open-source musical projects.**

***Index Terms*—Audio plugins, Digital Signal Processing, Pedal Pi.**

I. Introdução

EQUIPAMENTOS musicais para Processamento Digital de Sinais (DSP) tendem a ser vistos por instrumentistas iniciantes como uma possibilidade econômica para o seu desenvolvimento musical. O mercado de ferramentas para a música oferece dispositivos em versões de entrada — equipamentos básicos com custo reduzido — e em versões mais avançadas. O fácil acesso a uma quantidade relevante de *plug-ins* de áudio — implementações de efeitos desenvolvidas como extensões modulares [1] — de qualidade por um preço baixo — comparado a equipamentos analógicos similares — é um dos principais motivos para quem adquire tais equipamentos.

Entretanto, quando há um desenvolvimento musical, instrumentistas sentem a necessidade de substituir o equipamento. Limitações decorrentes da impossibilidade de um controle amplo resultam em restrições que afetam as *performances* criativa e musical de seus usuários. Equipamentos mais completos permitem uma extensão de controle através do padrão MIDI (*Musical Instrument Digital Interface*) [2]. Equipamentos de entrada não possuem suporte a esse padrão, seja pelo limite do custo de produção ou mesmo pela estratégia de mercado adotado pelas empresas.

Como alternativa a produtos comerciais de código fechado, iniciativas da comunidade de software livre, através da discussão, do compartilhamento de experiências e de código-fonte, incentivam a otimização e o reaproveitamento do código compartilhado [3] [4]. Nesse sentido, o AudioPint [5] oferece uma plataforma robusta para manipulação de efeitos e sintetização com Pure Data com enfoque na criação musical obtida através do controle por diversos equipamentos físicos; o equipamento MOD [6] concede a controle de *plugins* de áudio por meio de um protocolo aberto desenvolvido e o Zynthian provê uma plataforma totalmente aberta para sintetizadores [7], possibilitando a programação para situações específicas.

De forma similar, o projeto Pedal Pi é uma plataforma acessível que visa proporcionar uma solução DIY (*Do it yourself*, faça você mesmo) para uso de *plugins* de áudio com qualidade já consolidada [8]. Essa plataforma oferece uma API (do inglês, *Application Programming Interface*) para o desenvolvimento de Interfaces Homem-Máquina (IHM) para controle de suas configurações, onde é possível desenvolver componentes para controle aproveitando os pinos GPIO (*General Purpose Input/Output*) utilizados no sistema.

Dando continuidade ao projeto Pedal Pi, este trabalho propõe componentes que permitam o usuário controlá-lo, de forma a respeitar as características "custo acessível" e "faça você mesmo" da proposta apresentada no trabalho original. São desenvolvidas IHMs para o componente Raspberry – P0 e são elaborados componentes que oferecem serviços para controle utilizando os protocolos serial e REST.

O desenvolvimento deste trabalho foi composto pelas seguintes etapas: i) revisão bibliográfica de trabalhos relacionados; iii) estudo e enumeração dos recursos de controle e extensão oferecidos pela arquitetura do Pedal Pi; iii) implementação de recursos de controle e extensão; iv) resultados e discussão dentre as soluções de código-aberto (*open-source*) analisadas; e, v) resenha com as considerações finais e agradecimentos.

II. Revisão bibliográfica

Processamento Digital de Sinais (DSP) consiste no estudo da representação de sinais por modelos matemáticos e do processamento de sinais por sistemas discretos no tempo [9]. Através da utilização de métodos desse domínio é possível a comunicação entre equipamentos por meio de transmissão de dados sem fio [10], extrair de sinais ópticos a composição química de estrelas [11] e diagnosticar várias condições cardíacas através da análise de padrões ou aberrações em eletrocardiógrafos [12]. No contexto musical, trabalhos incluem criação de modelos matemáticos para modelagem de amplificadores valvulados de guitarra [13], simulação de instrumentos acústicos por meio de captadores de guitarras [14] e efeitos de áudio (como *tremolo* e *sustain* [15]).

P. M. M. da Silva and L. S. Costa are with the Master Program in Computer Science, Federal Institute of Ceará, 60040-215, Brazil. E-mail: mateus.moura@ppgcc.ifce.edu.br, leonardo.costa@ppgcc.ifce.edu.br

R. I. S. Pereira is with Federal University of Paraíba, 58059-900, Brazil, E-mail: renata@dee.ufc.br

S. C. S. Jucá is with Federal Institute of Ceará, Maracanaú Campus, 61939-140, Brazil, E-mail: sandrojuca@ifce.edu.br

Equipamentos embarcados para processamento de áudio utilizam em sua maioria *Processadores* de Sinais Digitais — microprocessadores especificamente construídos para lidar com tarefas de Processamento Digital de Sinal, como filtro digital e análises de Fourier [16]. Ao passo que a utilização de processadores específicos auxilia o desenvolvimento e a confiabilidade de um equipamento, limita a modificação por terceiros ao restringir o desenvolvimento para aqueles que possuem acesso à kits de desenvolvimento, sendo em grande parte profissionais que atuam na área.

Como alternativa, a utilização de processadores de propósito geral para realizar operações com áudio popularizou-se a partir do aumento da capacidade de execução de instruções, possibilitando a execução de programas para processamento de áudio simultaneamente a outros com *performance* satisfatória. Nesse ponto, a utilização e o desenvolvimento de ferramentas para esta área torna-se mais evidente, em especial softwares *open-source*, dada a facilidade e a liberdade de codificação. Ainda, a miniaturização também trazida com a evolução tecnológica permitiu embarcar aplicações e ferramentas desenvolvidas para tais processadores.

Nessa perspectiva, Audio Pint oferece uma plataforma robusta para invenção musical [5]. Utiliza *hardware* de computadores pessoais — placa-mãe com processador x86 — para a execução do sistema operacional Linux e dos *plugins* de áudio de arquitetura Pure Data e LADSPA. Pode ser ainda controlado por periféricos HID, Serial e USB por intermédio da biblioteca *hidio*. Foi montado em um *case* de plástico *Pelican* para a locomoção do equipamento.

Com uma perspectiva mais comercial, MOD [6] — posteriormente renomeado para MOD Quadra — oferece um equipamento robusto para processamento de *plugins* de áudio LV^2 aos pés do instrumentista. Utiliza processador de arquitetura ARM na placa-mãe desenvolvida para o equipamento. Sua versão posterior, MOD Duo, foi financiada pela comunidade através de uma campanha de financiamento coletivo [17]. Atualmente possui suporte à *plugins* de áudio LV^2, Max/MSP e Pure Data e pode ser gerenciada com uma interface *web* (mod-ui) ou com controladores sob o protocolo *control-chain*. Os códigos códigos desenvolvidos, exceto o *firmware*, estão sob licença aberta (GPLv3).

No eixo de sintetização de som, Zynthian oferece uma plataforma aberta, configurável e atualizável [7]. O equipamento, entretanto, é preparado para executar diversas *Engines*, inclusive a desenvolvida por MOD (mod-host/mod-ui). O *hardware* é composto com Raspberry Pi 3 e com uma interface de áudio de alta qualidade (HifiBerry DAC+). São vendidos kits do *hardware*, mas a comunidade também ensina como montá-lo — existem diversas variações com outras interfaces de áudio [18]. O controle desse equipamento por outros pode ocorrer tanto utilizando sua interface MIDI física [19], quanto através da USB por meio do protocolo *midi-over-usb* [20].

Aproveitando-se da disponibilidade de sistemas embarcados recentes de baixo custo, POP, acrônimo de Pedal Programável Aberto (*Programmable Open Pedal*), utiliza C.H.I.P., um SE de USD 9.00 com interface de áudio embutida [21]. O projeto é desenvolvido com uma perspectiva *maker*. Uma interface de configuração encontra-se em estágio inicial: o controle dos *plugins* de áudio LV^2 ocorre com *scripts* escritos em *shell script*.

Por fim, o Pedal Pi desenvolveu uma ferramenta de baixo custo para o processamento de *plugins* de áudio LV^2 [8]. O sistema pode ser executado tanto em um Raspberry Pi, quanto em conjunto com um computador de propósito geral, permitindo aproveitar o poder de processamento de um sistema computacional e a fácil customização de controle provida pelos pinos GPIO oferecidas pelo sistema embarcado. A arquitetura desenvolvida permite uma fácil expansão do equipamento através da integração de componentes que utilizem sua API. Por essa facilidade, foi escolhido utilizá-lo. Detalhes necessários para o entendimento de sua arquitetura são expostos na seção a seguir.

III. Análise da arquitetura do Pedal Pi

A arquitetura do Pedal Pi foi projetada para a comunicação de módulos lógicos (Figura 1). Conforme [8], "o intuito da abstração aplicada na arquitetura do projeto foi auxiliar os processos de evolução e de manutenção do *software*". A apresentação organizada permite a adição de novos componentes. Nesta sessão, os módulos da arquitetura são apresentados e avaliados com foco nos aspectos de extensibilidade.

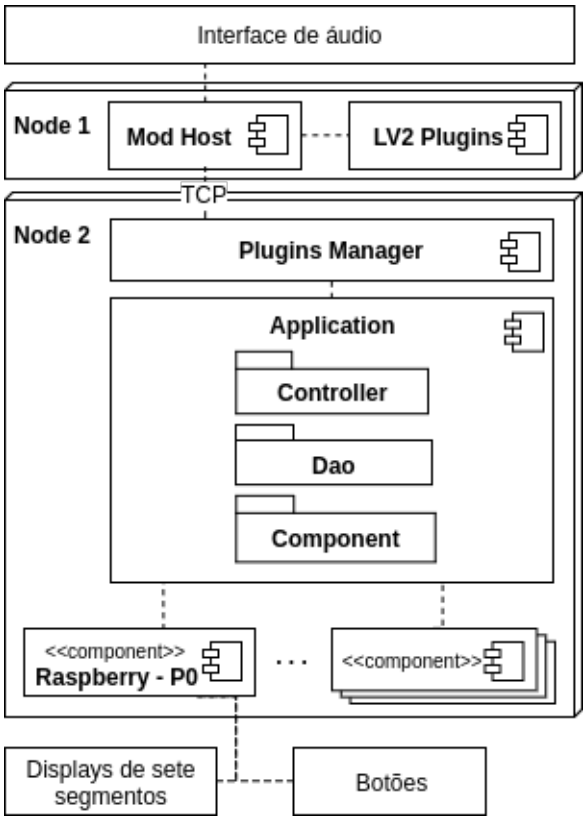

Fig. 1. Visão geral da arquitetura. Adaptado de [8].

A. Plugins de Áudio e Gerenciamento – Plugins Manager

No projeto original, a tecnologia escolhida para o fornecimento de *plugins* de áudio do Pedal Pi foi a LV^2 (acrônimo

de *Linux Audio Developer's Simple Plugin API* - LADSPA - *version 2*). LV^2 é um padrão aberto para o desenvolvimento *plugins* de áudio [22] e possui bibliotecas amplamente conhecidas na comunidade *open-source* musical, como Calf Plugins e Guitarix.

Em Linux Audio Conference (LAC), possivelmente a principal conferência para a divulgação de trabalhos relacionados à música para o Sistema Operacional Linux, há continuamente a produção de trabalhos envolvendo essa tecnologia. Exemplos de trabalhos compreendem o desenvolvimento de tecnologias para facilitar a criação de *plugins* de áudio [23], extensões do protocolo que auxiliam no desenvolvimento de aplicações como persistência do estado, sincronização de tempo e controle transparente de *plugins* sobre a internet [24].

Para o controle do servidor de som JACK e o gerenciamento dos *plugins* de áudio, Pedal Pi utiliza *mod-host* [25] — *host* LV^2 para o servidor de som JACK. Como a biblioteca *Mod-host* permite um controle por *sockets* TCP, para realizar um controle em alto nível de abstração, foi desenvolvido a biblioteca Plugins Manager.

Plugins Manager, com o auxílio da biblioteca *liblilv*, disponibiliza uma forma de controle e organização lógica de *plugins* de áudio através de uma API escrita em *python 3*.

Em sua abstração, instâncias dos *plugins* de áudio com seus parâmetros configurados e suas conexões estabelecidas ficam armazenadas em *pedalboards*. *Pedalboards* são agrupados em bancos. Com caráter ilustrativo, a Figura 2 relaciona a abstração apresentada com equipamentos físicos. Ainda, a Figura 3 exemplifica como utilizar esta abstração: é estabelecido uma conexão entre a biblioteca e mod-host, é criado o *pedalboard Beatles song* e adionado no *Bank 1*, é instanciado um *plugin* de áudio em *Beatles song*.

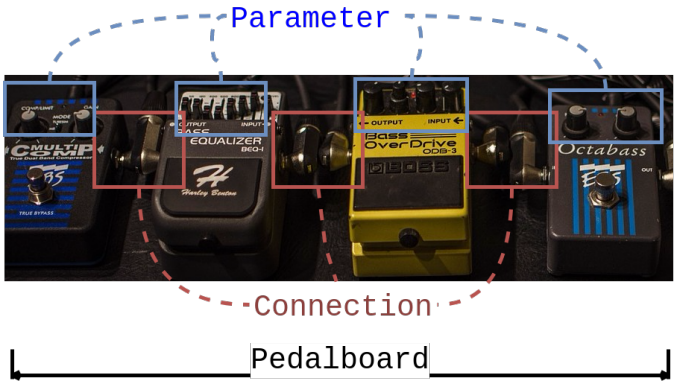

Fig. 2. Representação de *Parameter*, *Connection* e *Pedalboard* em um *pedalboard* físico para baixo elétrico. Os pedais de baixo representam instâncias de *plugins* de áudio. Adaptado de [26].

A vinculação entre um *pedalboard* e o mod-host (Figura 3, linha 16) faz com que mudanças no *pedalboard* reflitam de forma transparente no *host* de áudio. Para a realização deste feito, o padrão de projetos *Observer* [27] foi aplicado. Essa estratégia possibilita que os programadores desenvolvam códigos mais limpos ao estrutura-lo considerando somente o nível de abstração necessário [28]. Outro recurso de Plugins Manager que utiliza desta mesma estratégia é o módulo *autosaver*, responsável por salvar configurações automaticamente.

A estrutura fornecida pela utilização do *design pattern Observer* (através da classe UpdatesObserver) serviu como ponto de extensão da biblioteca. Além dos usos supracitados, é possível ainda alterar Plugins Manager para integrar outros *hosts* de áudio. Por exemplo, é possível substituir mod-host por Carla, host que oferece um suporte mais amplo de arquiteturas de *plugins* de áudio.

```
1  manager = BanksManager()
2
3  # Connecting with mod_host
4  mod_host = ModHost('localhost')
5  mod_host.connect()
6
7  manager.register(mod_host)
8
9  # Create bank and pedalboard
10 bank = Bank('Bank 1')
11 manager.append(bank)
12 pedalboard = Pedalboard('Beatles song')
13 bank.append(pedalboard)
14
15 # Link pedalboard with mod-host
16 mod_host.pedalboard = pedalboard
17
18 # Audio plugins instances
19 builder = Lv2EffectBuilder()
20 reverb = builder.build(
21    'http://calf.sourceforge.net/plugins/Reverb'
22 )
23 pedalboard.append(reverb)
24 reverb.params[0].value = 5
25
26 # Connections
27 sys_effect = SystemEffect('system',
       ↪ ['capture_1'], ['playback_1'])
28 pedalboard.connect(sys_effect.outputs[0],
       ↪ reverb.inputs[0])
29 pedalboard.connect(reverb.outputs[0],
       ↪ sys_effect.inputs[0])
```

Fig. 3. Conexão com mod-host, criação e gerenciamento de configurações.

B. Biblioteca Application

A biblioteca Application é a parte orquestradora de Pedal Pi. A utilização de Pedal Pi como sistema se dá por meio dela: Application carrega os módulos, especifica componentes, define seu ciclo de vida e gerencia a comunicação entre componentes. Componentes são pedaços de software que oferecem (através de uma interface) serviços predefinidos e estão habilitados a se comunicarem com outros componentes [29].

A arquitetura definida para uso de componentes é peça central para a execução deste trabalho. Application possibilita o registro de componentes antes da sua inicialização (Figura 4, linha 3). Componentes podem registrar observadores em Application para que sejam informados de mudanças do estado na aplicação. Quando o sistema é encerrado (linha 8), os recursos gerenciados pelo componente (como pinos GPIO, conexões à bancos) devem ser liberados.

```
application = Application()
# Register component Raspberry - P0
application.register(RaspberryP0(application))
application.start()
try:
   pause()
except KeyboardInterrupt:
   application.stop()
```

Fig. 4. Registro de componentes na aplicação.

Application ainda define a funcionalidade de configuração corrente ao Pedal Pi.

C. Componente Raspberry - P0

O conceito de *configuração corrente* (ou configuração atual) está presente na maioria de equipamentos para processamento de áudio. De forma simplificada, a configuração corrente corresponde ao *pedalboard* que está sendo executado na aplicação, isto é, as instâncias de *plugins* que devem estar carregadas no *host* juntamente com suas conexões e seus parâmetros atribuídos corretamente. Tal ideia traz ainda o comportamento de alteração da configuração corrente, atribuído geralmente a botões (*next/before pedalboard*).

Com a ideia de oferecer um gerenciamento simplificado e de fácil construção, Raspberry - P0 foi desenvolvido. Raspberry - P0 é uma IHM composta por *displays* de sete segmentos e *footswitches* (Figura 6 – (a)). Sua utilização traz a possibilidade do usuário poder visualizar o número da configuração corrente em uso e alterá-la para a próxima configuração ou para a anterior. O código e a IHM apresentaram-se de fácil implementação. O nível de abstração alto — obtido pelo uso da biblioteca *gpiozero* — permitiu utilizar este projeto como referência para os componentes a serem desenvolvidos.

IV. Componentes

A partir da análise de como os projetos *open-source* para processamento de *plugins* oferecem meios de estender seu controle, e em especial o estudo da arquitetura do Pedal Pi, os esforços concentraram-se no desenvolvimento de componentes. Foram desenvolvidas interfaces homem-máquina para o componente Raspberry – P0. Paralelamente, foram implementadas integrações entre o Pedal Pi e outros sistemas, a partir do conceito de *abertura*.

A. Componentes de Controle

Sendo Pedal Pi um equipamento concebido com o ideal DIY, os componentes físicos que serão utilizados para sua construção poderão variar de acordo com os equipamentos que os *makers* dispõem. Dada a liberdade no desenvolvimento de interfaces para controle, protótipos para o componente Raspberry – P0 foram construídos buscando estimulá-los promovendo a MetaReciclagem e a criatividade no desenvolvimento de interfaces que se adaptem às suas necessidades (Figura 6).

No primeiro protótipo foram utilizados componentes de fácil acesso: caixa para interruptor de luz, dois pulsadores de campainha. Prevendo um futura melhoria, também foi adicionada uma chave seletora. O controle apresentou resultados desejáveis nas primeiras semanas de uso, entretanto, o desgaste no sistema de molas dos pulsadores fez com que eles não voltassem à posição inicial.

Para solucionar esse problema, foi decidido utilizar botões sem sistema físico de mola. No segundo protótipo foram utilizados *footswitches* 3DPT (botões geralmente utilizados em pedais de guitarra e baixo preparados mecanicamente para suportar pressão sobre eles) e um condulete de alumínio. A utilização de *footswitches* aumentou consideravelmente o preço do protótipo. Como alternativa é possível utilizar outros tipos de botões, sendo estes de dois estados (liga/desliga) ou momentâneos.

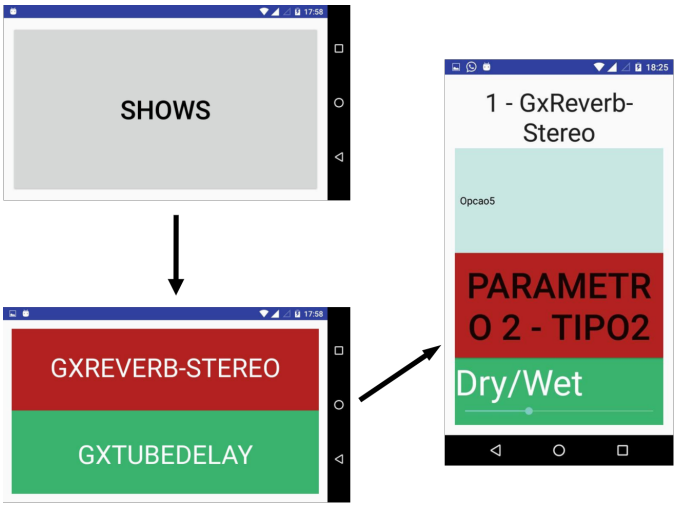

Fig. 5. Ilustração das três telas disponíveis no aplicativo, seguindo a ordem do fluxo: Nome do *pedalboard* corrente, seus efeitos (suas instâncias de *plugins* de áudio) e lista de parâmetros de um efeito.

B. Componentes de Abertura

Em sistemas distribuídos, um sistema é considerado aberto quando oferece serviços de acordo com regras padronizadas que descrevem a sintaxe e semântica desse serviços [30]. No contexto musical, há uma preferência da utilização do protocolo MIDI [31] para a comunicação e controle entre equipamentos distintos. Além de mensagens padronizadas, essa especificação ainda permite a definição de comandos próprios por meio de mensagens SysEx (*System Exclusive Messages*), suportando as nuâncias de produtos específicos.

Um ponto de falha da utilização do protocolo MIDI como único ponto de abertura é a falta de definição das mensagens SysEx. Uma empresa pode fornecer mensagens de sistema exclusivas para controles particulares de seu equipamento, mas não tornar os documentos de especificação abertos impossibilitará a comunicação com produtos de outras empresas. Outro ponto de falha consiste na implementação incompleta dos tipos de mensagem. Há casos em que equipamentos não respondem, por exemplo, à mensagens *Program (Patch) Change*, associadas à alteração da configuração corrente.

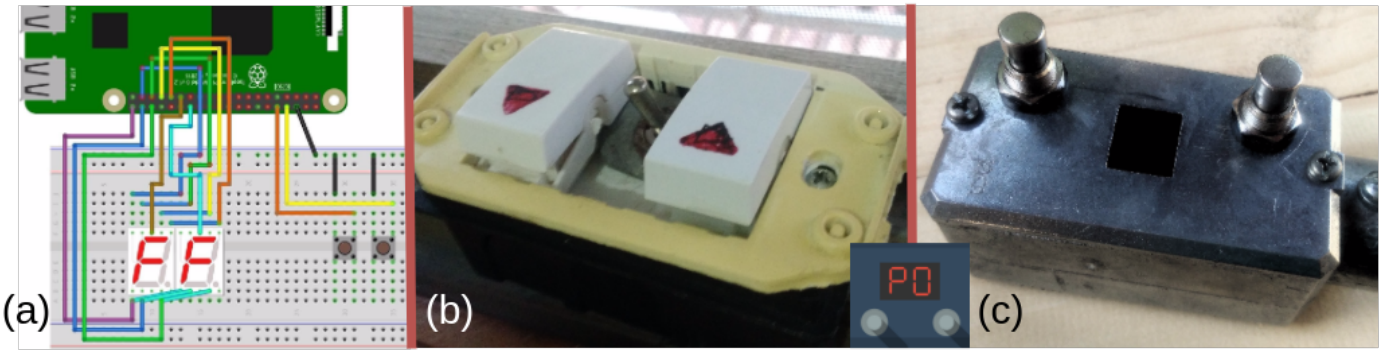

Fig. 6. (a) conexão entre os componentes e um Raspberry Pi (modelo de pinos $B+$). Respectivamente (b) e (c), protótipos dos botões controladores do *pedalboard* atual (configuração corrente) de Raspberry – P0, construídos com pulsadores de campainha e *footswitches*.

Pedal Pi atualmente não possui suporte ao protocolo MIDI. Os esforços tomados por seus desenvolvedores concentraram-se na implementação em *Python 3* da API (*Application Programming Interface*) e em sua documentação. A partir da API, é possível adicionar pontos de abertura. Foram desenvolvidos dois *componentes de abertura* e clientes que consomem os serviços oferecidos:

1) Componente Android Controller: Android Controller é um componente de abertura desenvolvido para possibilitar a comunicação entre um dispositivo *Android* e o Raspberry Pi via TCP sobre o USB 2.0 por intermédio da ferramenta para depuração ADB (*Android Debug Bridge*).

Para esta interface, foi desenvolvido o aplicativo *Display Controlador*. Esse consumidor do serviço fornecido por Android Controller busca seguir uma sequência lógica e intuitiva entre as telas para uma melhor desenvoltura do instrumentista em performances ao vivo (figura 5): Uma tela inicial informa o *pedalboard* atual; seus efeitos estão acessíveis a um clique; e, para entrar no modo de configuração do efeito é necessário pressioná-lo durante alguns segundos (*long click*).

2) Componente WebService: WebService fornece um serviço REST (*Representational State Transfer*) e WebSocket sobre a API de Pedal Pi. O controle a partir de WebService é mais extenso comparado ao Android Controller: por meio dele é possível o gerenciamento de bancos, *pedalboards*, suas instâncias e parâmetros de *plugins* de áudio dos *pedalboards*. Ainda, através do protocolo ZeroConf (*Zero Configuration Networking*) é possível a descoberta automática de equipamentos Pedal Pi que utilizem este componente, facilitando a conexão com clientes. A especificação da API desenvolvida pode ser consultada através da descrição do repositório no Github [32].

Foi desenvolvido o cliente Apk para consumir os serviços oferecidos pelo WebService. Internacionalizado para os idiomas inglês, espanhol, português e alemão, promove uma interface amigável para a configuração do Pedal Pi nos momentos de ensaio e estudos diários. Baseado em tecnologias web, pode ser utilizado como um aplicativo *Android* como ainda em computadores pessoais com navegadores modernos. A figura 7 apresenta algumas das telas do *Apk*.

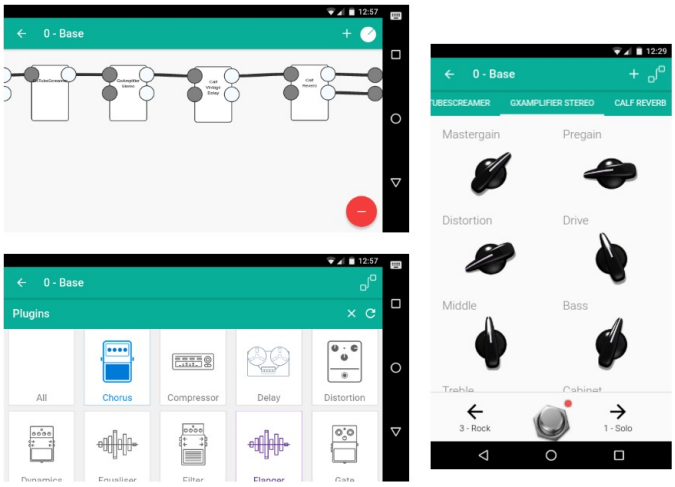

Fig. 7. Telas do cliente Apk para o componente WebService: Conexões das instâncias dos *plugins* de áudio, categorias de *plugins* de áudio e parâmetros de uma instância para *plugins* de áudio.

V. Resultados

Os trabalhos com tecnologias abertas para DSP no eixo musical encontram-se avançados. Nos projetos descritos na seção II são utilizados *hardware* de $i)$ computadores pessoais [5], $ii)$ de sistemas embarcados [7] [21] ou ainda $iii)$ desenvolvidas especificamente para o projeto [6] [17]. A utilização de *softwares* abertos permitiu o avanço do estado da arte e o desenvolvimento de tecnologias abertas faz com que o sistema seja retroalimentado. Como exemplo, tecnologias desenvolvidas por MOD são utilizados em Zynthian, em POP e no Pedal Pi, que, por sua vez, desenvolvem tecnologias e contribuem para MOD.

Alguns dos projetos oferecem meios para expansão do controle por meio de componentes externos que seguem um protocolo ou API especificada. Quando não, confiam na possibilidade de consultar e alterar o próprio código fonte. Ao seguir algumas das premissas de *software* livre, os projetos permitem sua duplicação (*fork*) e seu aperfeiçoamento [33]. Entretanto, o processo torna-se custoso quando no desenvolvimento do código original não foram aplicadas boas práticas

para mantê-lo limpo [28] e a ausência de documentação atualizada restringe o conhecimento da aplicação aos desenvolvedores originais [34].

Dentre as soluções apresentadas na revisão bibliográfica, Pedal Pi destaca-se. Através da análise da arquitetura foram percebidos cuidados no código atentando a sua legibilidade, a sua qualidade e sua documentação. Como decorrência, a criação de componentes para a extensão do Pedal Pi foi facilitada considerando os projetos Zynhian e POP, onde a ausência de documentação resultava em esforço para analisar e modificar o código da aplicação dificultava a criação de novas formas de controle em seus ambientes de desenvolvimento.

Como resultados, foram desenvolvidas IHMs para o componente Raspberry – P0, respeitando as premissas DIY e custo acessível. Ainda, componentes de abertura foram construídos permitindo a comunicação com outras aplicações para o controle do Pedal Pi. Entretanto, o trabalho desenvolvido não abordou a criação do protocolo com maior adoção para controle de equipamentos de áudio (MIDI). Isso foi motivado em respeito às limitações de gasto definidas no projeto base: a montagem do Pedal Pi no desenvolvimento do projeto continuou a utilizar uma interface de áudio sem suporte à MIDI. Contudo, a própria comunidade do projeto mostrou-se interessada em implementar [35]. Desta forma, a divulgação do desenvolvimento e dos resultados de componentes desenvolvidos é um importante meio e oferecer o suporte à comunidade, e, consequentemente, aos usuários de equipamentos de processamento de áudio.

VI. Considerações finais

A documentação apresentada do componente Raspberry – P0 permitiu a reprodução rápida da interface física de duas formas distintas, demonstrando ser adaptável às necessidades do usuário e estando alinhado à filosofia DIY do projeto. A liberdade oferecida pela API do Pedal Pi facilitou o processo de desenvolvimento de componentes de abertura para integração com outros equipamentos, em contraste com os outros equipamentos existentes no mercado. Ainda, o processo de desenvolvimento de componentes que utilizem outros equipamentos físicos, como *displays* LCD e sensores, torna-se simplificado pela quantidade de material de apoio já produzido pela comunidade do Raspberry Pi e pela Raspberry Foundation.

Além da documentação do código fornecida como material de apoio para a manutenção da biblioteca, estão sendo desenvolvidos um site e um *blog* como material didático, fornecendo instruções de como montar o equipamento e desenvolver novos componentes. Para aqueles que estão impossibilitados a montarem o próprio equipamento — seja por falta de acesso a equipamentos ou de habilidades com solda, cogita-se a venda de kits pré-prontos como alternativa.

Como sugestões para trabalhos futuros, são propostos o desenvolvimento de IHM adaptadas para pessoas com deficiência motora, o suporte ao protocolo MIDI e a criação de módulos para sistemas microcontrolados como Android e NodeMCU.

Agradecimientos

Os autores gostariam de agradecer à comunidade *open-source*, em especial a MOD Devices pelas bibliotecas desenvolvidas, como também à Raspberry Foundation, cujo trabalho de tornar sistemas embarcados financeiramente acessível em países emergentes resultou na criação de equipamentos com baixo custo e alta performance. O agradecimento estendem-se ao IFCE — Instituto Federal de Educação, Ciência e Tecnologia — pelo suporte educacional e à disponibilidade de espaço e de equipamentos e à Universidade Federal do Ceará (UFC) e aos membros Secretaria de Tecnologia da Informação (STI) pelo suporte, discussões e sugestões para a realização do trabalho.

Referências

[1] F. L. Schiavone, A. J. H. Goulart, and M. Queiroz, "Apis para o desenvolvimento de aplicações de áudio," *Seminário Música Ciência Tecnologia*, vol. 1, no. 4, 2012.

[2] D. M. Huber, *The MIDI Manual: A Practical Guide to MIDI in the Project Studio (Audio Engineering Society Presents)*, 3rd ed. Focal Press, 2012.

[3] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," *IEEE Transactions on Software Engineering*, vol. 30, no. 4, pp. 246–256, 2004.

[4] A. Hars and S. Ou, "Working for free? motivations for participating in open-source projects," *International Journal of Electronic Commerce*, vol. 6, no. 3, pp. 25–39, 2002.

[5] D. Merrill, B. Vigoda, and D. Bouchard, "Audiopint: A robust open-source hardware platform for musical invention," in *Pd Convention 2007*, Montréal, Québec, Canada, 2007.

[6] G. Ceccolini and L. Germani, "Mod – an lv2 host and processor at your feet," in *Linux Audio Conference 2013 Proceedings*, I. m zmölnig e Peter Plessas All, Ed., Institute of Electronic Music and Acoustics, Graz, Austria, May 2013, pp. 159–161.

[7] J. F. Moyano, "Zynthian: The open synth platform," 2016, [accessed August-2016]. [Online]. Available: http://zynthian.org/

[8] P. M. M. d. Silva, L. d. S. Costa, and S. C. S. Juca, "Pedal pi – multi-processador de plugins de áudio diy," in *III Escola regional de informática do Piauí*, vol. 1, Jun 2017, pp. 23–28.

[9] A. Antoniou, *Digital Signal Processing: Signals, Systems, and Filters*. McGraw-Hill Education, 2005.

[10] V. Tarokh, N. Seshadri, and A. Calderbank, "Space-time codes for high data rate wireless communication: performance criterion and code construction," *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 744–765, mar 1998. [Online]. Available: https://doi.org/10.1109/18.661517

[11] G. A. Blake, E. C. Sutton, C. R. Masson, and T. G. Phillips, "Molecular abundances in OMC-1 - The chemical composition of interstellar molecular clouds and the influence of massive star formation," , vol. 315, pp. 621–645, Apr. 1987.

[12] J. J. Goldberger and J. Ng, Eds., *Practical Signal and Image Processing in Clinical Cardiology*. Springer London, 2010. [Online]. Available: https://doi.org/10.1007/978-1-84882-515-4

[13] J. Pakarinen and D. T. Yeh, "A review of digital techniques for modeling vacuum-tube guitar amplifiers," *Computer Music Journal*, vol. 33, no. 2, pp. 85–100, 2009. [Online]. Available: http://dx.doi.org/10.1162/comj.2009.33.2.85

[14] M. Karjalainen, H. Penttinen, and V. Valimaki, "Acoustic sound from the electric guitar using dsp techniques," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 2. IEEE, 2000, pp. II773–II776.

[15] E. Berdahl and J. O. Smith, "Some physical audio effects," in *Proceedings of the 9th International Conference on Digital Audio Effects, pages*, 2006, pp. 165–168.

[16] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA, USA: California Technical Publishing, 1997.

[17] M. O. Devices, "Mod duo: The limitless multi-effects pedal." 2015, [accessed August-2017]. [Online]. Available: https://www.kickstarter.com/projects/modduo/mod-duo-the-limitless-multi-effects-pedal

[18] disadmin, "Success cases - building zynthian," 2016, [accessed August-2017]. [Online]. Available: https://discourse.zynthian.org/t/success-cases/41/1

[19] J. F. Moyano, "Building a zynthian box using an official kit," Jan 2017, [accessed September-2017]. [Online]. Available: http://wiki.zynthian.org/index.php/Building_a_Zynthian_Box_using_an_official_Kit
[20] ——, "Zynthian midi-ub controller," May 2017, [accessed September-2017]. [Online]. Available: https://discourse.zynthian.org/t/zynthian-midi-ub-controller/709/1
[21] I. M. Manuel Guerrero Escobar, "Pop: The programmable open pedal," March 2016, [accessed August-2017]. [Online]. Available: https://popularelectronics.technicacuriosa.com/2017/03/07/pop-the-programmable-open-pedal/
[22] D. E. Robillard, "Why lv2?" 2012, [accessed August-2017]. [Online]. Available: http://lv2plug.in/pages/why-lv2.html
[23] A. Gräf, "Creating lv2 plugins with faust," in *Proc. 11th Int. Linux Audio Conf.(LAC-13), Graz, http://lac. linuxaudio. org*, 2013.
[24] D. Robillard, "Lv2 atoms: A data model for real-time audio plugins," in *Linux Audio Conference*, 2014.
[25] M. Devices, "mod-host — lv2 host for jack controllable via socket or command line," 2017, [accessed August-2017]. [Online]. Available: https://github.com/moddevices/mod-host
[26] egonkling, 2015, [accessed September-2017]. [Online]. Available: https://pixabay.com/pt/m\%C3\%BAsica-effektger\%C3\%A4t-bass-extenuar-1004104/
[27] H. R. J. R. Gamma, E. and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1994, iSBN:0-201-63361-2.
[28] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.
[29] R. Niekamp, "Software component architecture," 2005, institut für Wissenschaftliches Rechnen, Germany, Lecture Note. [Online]. Available: https://www.wire.tu-bs.de/forschung/projekte/ctl/files/ctl_course.pdf
[30] A. S. Tanenbaum and M. v. Steen, *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.
[31] M. M. Association *et al.*, "The complete midi 1.0 detailed specification," *Los Angeles, CA, The MIDI Manufacturers Association*, 1996.
[32] P. M. M. Silva, "Raspberry latency configuration," 2016, [accessed September-2016]. [Online]. Available: https://discourse.zynthian.org/t/raspberry-latency-configuration/133
[33] "What is free software?" April 2017, [accessed August-2017]. [Online]. Available: https://www.gnu.org/philosophy/free-sw.en.html
[34] M. J. C. Sousa and H. M. Moreira, "A survey on the software maintenance process," in *Software Maintenance, 1998. Proceedings., International Conference on*. IEEE, 1998, pp. 265–274.
[35] Holger, 2017, [accessed September-2017]. [Online]. Available: https://discourse.zynthian.org/t/mod-on-pisound/1103

Paulo Mateus Moura da Silva. Brasileiro, nascido em Maracanaú – CE, em 1994. Possui o Técnico em Informática pelo Instituto Federal de Educação, Ciência e Tecnologia do Ceará, campus Fortaleza (IFCE Fortaleza). Graduado em Bacharelado em Ciência da Computação pelo Instituto Federal de Educação, Ciência e Tecnologia do Ceará, campus Maracanaú (IFCE Maracanaú) e mestrando no Programa de Pós-Graduação em Ciência da Computação do campus Fortaleza desta mesma instituição. É técnico de Tecnologia da Informação da Universidade Federal do Ceará e bolsista voluntário na área de Telemática do IFCE – Campus Maracanaú. Os temas de pesquisa são Sistemas Embarcados e Aprendizagem de Máquina.

Leonardo da Silva Costa. Brasileiro, nascido em Fortaleza – CE, em 1995. Possui o Técnico em Finanças pela Escola Estadual de Educação Profissional Gov. Luiz de Gonzaga Fonseca Mota. Graduado em Bacharelado em Ciência da Computação pelo Instituto Federal de Educação, Ciência e Tecnologia do Ceará, campus Maracanaú (IFCE Maracanaú) e Mestrando no Programa de Pós-Graduação em Ciência da Computação do campus Fortaleza desta mesma instituição. É bolsista atuante no Laboratório de Eletrônica e Sistemas Embarcados (LAESE) do mesmo campus. Os temas de pesquisa são Sistemas Embarcados e Inteligência Artificial.

Renata Imaculada Soares Pereira. Nascida em Fortaleza – CE, Brasil, em 1990. Atualmente é aluna de pós-doutorado do programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Paraíba (UFPB) com bolsa PNPD-CAPES. Concluiu mestrado (2014) e doutorado (2018) em Engenharia Elétrica pela Universidade Federal do Ceará (UFC) com pesquisa de doutorado sanduíche na Universidade Técnica de Colônia, Alemanha. Concluiu Tecnólogo em Manutenção Industrial (2012) e curso técnico em Automação Industrial (2009) no Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE - Campus Maracanaú). Áreas de atuação: Sistemas embarcados, Internet das coisas, Aquisição de dados, Monitoramento online e Energias renováveis.

Sandro César Silveira Jucá. Brasileiro, nascido em Fortaleza – CE, em 1975. Graduou-se em Tecnologia Mecatrônica pelo Centro Federal de Educação Tecnológica do Ceará (CEFET–CE). Cursou Licenciatura em Física e especialização em Automação Industrial pela Universidade Estadual do Ceará (UECE). É mestre e doutor em Engenharia Elétrica pela Universidade Federal do Ceará (UFC). Realizou pesquisa com doutorado-sanduíche do DAAD na Universidade de Paderborn (Uni Paderborn), de 2011 a 2012. É professor e pesquisador da área de Telemática do IFCE – Campus Maracanaú. É coautor do livro "Desenvolvimento de Sistemas de Aquisição de Dados Sem Fio". Os temas de pesquisa são Fontes Renováveis de Energia e Sistemas Embarcados.