APIs PARA O DESENVOLVIMENTO DE APLICAÇÕES DE ÁUDIO

Flávio Luiz Schiavoni, Antonio José Homsi Goulart, Marcelo Queiroz
Instituto de Matemática e Estatística, Universidade de São Paulo
{fls | ag | mqz}@ime.usp.br

Resumo: O áudio no computador passa por diversas camadas. O desenvolvimento ou a escolha de uma aplicação é fortemente afetada pela API de áudio que a aplicação utiliza. É diante deste cenário que este artigo apresentará uma discussão sobre algumas possibilidades de APIs para o desenvolvimento de aplicações de áudio.

Palavras-chave: APIs de áudio, plugins, ecossistema de áudio.

APIs for audio applications development

Abstract: Audio in computer systems flows through several layers. The API used by an application strongly influences the choice of and the development with this application. In face of that this paper presents a view on some possibilities of APIs for audio applications development.

Keywords: Audio APIs, plugins, audio ecosystem.

INTRODUÇÃO

O desenvolvimento de uma aplicação de áudio pode ter por objetivo criar ferramentas para fins musicais mas também pode possuir um aspecto didático/pedagógico (LAZZARINI, WALSH, 2007). Entre as várias possibilidades para a implementação da comunicação de entrada e saída de áudio no computador, comumente tal desenvolvimento é feito por meio de uma API de áudio. Uma API (*Application Programming Interface*) é uma série de funções que permitem ao programador acoplar novas funcionalidades a um programa já existente. A escolha de uma determinada API pode simplificar o desenvolvimento de aplicações mas também traz consigo limitações inerentes desta API. Além disto, o conhecimento prévio das vantagens e desvantagens de uma API pode influenciar também na escolha de uma aplicação a ser integrada ao *métier* tecnológico de um determinado processo musical.

Entre as várias características que podem ser analisadas em uma API para o desenvolvimento, este trabalho trata da portabilidade, linguagem de programação e licença para a distribuição. A licença da API é um critério mais burocrático que técnico mas que influencia diretamente nas condições de distribuição da ferramenta desenvolvida. Este artigo não irá tratar os requerimentos para a distribuição de uma aplicação que utiliza uma API com determinada licença mas irá apresentar esta característica de cada API analisada.

Outra característica que pode influenciar a escolha do desenvolvedor é a necessidade de utilizar comunicação MIDI pela mesma API de áudio. Apesar de não ser diretamente relacionada com aplicações de áudio, o protocolo MIDI é muitas vezes utilizado como protocolo de comunicação entre aplicações e controladores externos e por isto o mesmo pode ser útil para alguns tipos de aplicações.

Partindo do *hardware* de áudio, o mesmo pode trazer limitações para o desenvolvimento de aplicações de áudio, pois cada interface de som possui suas restrições de configuração, tais como a taxa de amostragem, tamanho da amostra de som e quantidade de canais de

entrada e saída. Estas configurações são limitações físicas da interface e uma aplicação de áudio pode necessitar obter tais configurações para o processamento do sinal de áudio. A aplicação obtém tais informações comunicando-se diretamente com a interface ou com o sistema operacional por meio do *driver* fornecido pelo fabricante. Para simplificar o acesso à interface, o sistema operacional fornece uma camada de abstração que é sua API de áudio. Por meio da API de áudio é possível implementar aplicações que se comunicam com o *hardware*. Porém, utilizar a API do sistema operacional significa limitar sua aplicação a esse sistema operacional e implica em criar dificuldades para a portabilidade da aplicação.

Para garantir a portabilidade entre aplicações e APIs de áudio do sistema operacional, surgiram APIs portáveis que permitem o desenvolvimento de aplicativos para vários sistemas operacionais. A portabilidade de um sistema é a capacidade de o mesmo trabalhar em várias arquiteturas de computadores e/ou vários sistemas operacionais. Além da portabilidade, é desejável que as aplicações de áudio se comuniquem com aplicações já existentes. Caso isto seja possível, podemos planejar o desenvolvimento de aplicações de áudio através do desenvolvimento de pequenos módulos que possam se acoplar criando outras aplicações.

Há vários exemplos de aplicações que trabalham de forma modular, como, por exemplo, os comandos de *shell* do Linux / Unix. Estes comandos são pequenas aplicações que possuem uma única funcionalidade mas que podem ser combinadas em *scripts shell* para a criação de aplicações poderosas. De maneira análoga ao *shell* do Linux, várias aplicações de áudio aceitam extensões modulares chamadas *plugins.* A definição de um *plugin* é feita por meio da API do *plugin*.

É possível ainda criar aplicações modulares para áudio utilizando um servidor de som. Alguns exemplos são o Jack e o Sound Flower, que permitem que aplicações sejam interconectadas na forma de um *patch bay*. Desenvolver uma aplicação que se conecte a um servidor de áudio depende também de utilizar a API do servidor para o desenvolvimento.

Este artigo apresenta algumas APIs como alternativas para a implementação de uma aplicação musical buscando principalmente a portabilidade da aplicação desenvolvida. Para obter uma visão geral destas APIs apresentamos uma arquitetura em camadas conforme apresentada pela Figura 1.

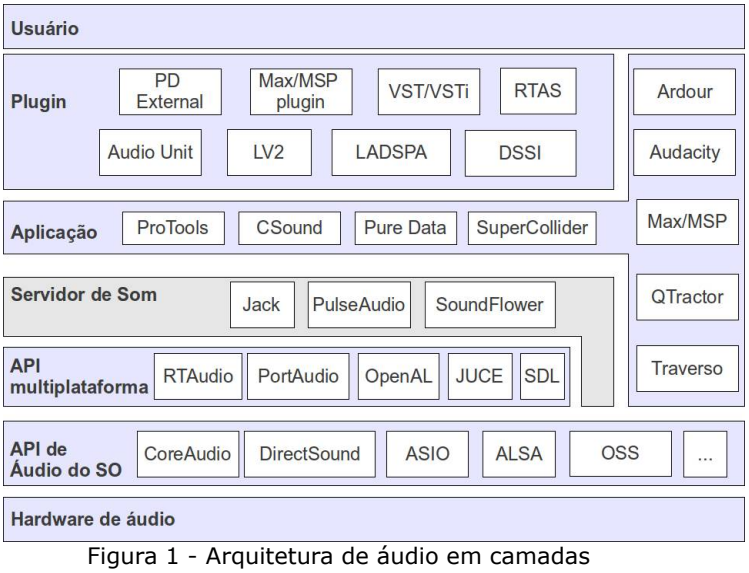

Figura 1 - Arquitetura de áudio em camadas

Este artigo irá detalhar algumas camadas apresentadas na Figura 1. As APIs de áudio do sistema operacional são apresentadas brevemente na segunda seção deste texto. Na terceira seção discutiremos as APIs multiplataforma, na terceira seção discutiremos os

servidores de som e na quarta seção trataremos dos *plugins* de aplicações. As camadas de aplicações e de *drivers* não serão comentadas por não se encontrarem no escopo deste trabalho.

API de áudio do sistema operacional

A API de áudio nativa de um sistema computacional depende de seu sistema operacional (SO). Apesar da possibilidade de desenvolvimento para tal camada, o acoplamento da aplicação desenvolvida e o SO é muito alto e a portabilidade do código para outros sistemas computacionais implicaria em reescrita do código da mesma utilizando APIs de outros SO. Além disto, um mesmo SO pode possuir diferentes implementações de sua camada de áudio, o que não garante que a aplicação possa ser executada em qualquer instalação deste SO.

Exemplos desta camada no Windows são o DirectSound, Windows Mutilmedia Extensions (MMEd), WinKS, WASAPI, Windows Multimedia Library e ASIO. Em Linux há o FFADO, ALSA e OSS além de variações de servidores de som como o ESD e o arts. No MacOS há versões com o Apple Sound Manager (SM), ASIO e Apple Core Audio. Além da existência de APIs distintas para se desenvolver aplicações sobre um dado sistema, deve-se considerar que em geral as APIs possuem latências diferentes para o mesmo sistema operacional (WANG, STABLES e REISS 2010; MACMILLAN, DROETTBOOM e FUJINAGA 2001). Para evitar um acoplamento tão alto entre aplicação e SO uma alternativa é subir para a camada das APIs multiplataformas.

APIs multiplataforma

Visando simplificar a portabilidade de aplicações, foram desenvolvidas bibliotecas que permitem a implementação de uma aplicação sobre várias APIs de áudio do sistema operacional. Exemplos destas APIs de áudio multiplataforma são JUCE, SDL, OpenAL, RTAudio e PortAudio.

O **JUCE** (Jules' Utility Class Extensions) (RAW, 2008) é uma API particularmente eficiente para a construção de GUIs altamente customizadas, e para trabalhos que envolvam processamento de gráficos ou de áudio. Esta biblioteca pode ser utilizada também para o desenvolvimento de plugins que serão vistos neste trabalho. Os sistemas compatíveis com o JUCE são o Mac OS X, iOS, Windows, Linux e Android.

O **SDL** (Simple DirectMedia Layer) (SDL, 2012) é uma API projetada para oferecer acesso em baixo nível a áudio, teclado, mouse, hardware 3D por OpenGL e vídeo. É utilizado por softwares de reprodução de MPEG, emuladores e vários vídeo-jogos populares. O SDL é compatível com Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, e QNX. O código possui ainda suporte para Dreamcast e Atari. Funciona com C e C++, além de possuir vinculações para as linguagens Ada, C#, D, Eiffel, Erlang, Euphoria, Go, Guile, Haskell, Java, Lisp, Lua, ML, Objective C, Pascal, Perl, PHP, Pike, Pliant, Python, Ruby, Smalltalk e Tcl.

O **openAL** (CREATIVE, 2012) foi desenvolvido visando a migração para Linux de jogos criados originalmente para Windows. O desenvolvimento foi realizado pela empresa Loki Software e atualmente o projeto pertence à empresa Creative Technology. O openAL é compatível com as plataformas Mac OS X, iOS, Linux, Solaris, IRIX, Windows, Xbox, Xbox360 e MorphOS. Além disso, existe a possibilidade de diferentes fabricantes de *hardware* de som poderem incluir suas próprias extensões para o openAL em seus equipamentos. Dependendo do sistema sobre o qual roda, o openAl possui diferentes licenças.

O **RTAudio** (SCAVONE, 2002; SCAVONE e COOK, 2005; SCAVONE, 2012) tem como objetivo proporcionar entrada e saída de áudio e MIDI em tempo real. A ferramenta incorpora o conceito de fluxos (*streams*), que representam saídas ou entradas de áudio, e funciona em Windows, Linux e Mac OS X. Gary Scavone, o desenvolvedor, mantém extensa documentação no site (SCAVONE, 2012).

O **PortAudio** (BENCINA e BURK, 2001) (BENCINA, 2003) (PORTAUDIO, 2012) é a API de áudio proposta por Ross Bencina para o PortMedia. Este projeto possui como objetivo permitir o acesso multiplataforma a conteúdos de mídia e apresenta o PortMidi como alternativa para Midi. A API do PortAudio permite a escrita de programas para Windows, Mac OS X e Linux (OSS/ALSA). Alguns exemplos de programas escritos sobre PortAudio são Pure Data, CSound e Audacity.

Tabela 1: Comparação entre APIS de áudio

API	MIDI	Linguagem	Licença
JUCE	Sim	C++	GPL
SDL	Não	várias	LGPLv2
RTAudio	Sim	C/C++	MIT /GPL3
PortAudio	Sim	C/C++	MIT
OpenAL	Não	C++	LGPL/outras

As APIs OpenAL e SDL possuem uma abordagem voltada para jogos e por isto possuem amplo suporte para espacialização. Por outro lado, o fato de não possuírem acesso a dispositivos MIDI pode limitar a utilização das mesmas em aplicações musicais que necessitem deste suporte. Já o JUCE, PortAudio e RtAudio possuem uma abordagem voltada para aplicações em contexto musical.

Servidores de Som

Servidores de som adicionam uma camada a mais no sistema de comunicação entre a aplicação de áudio e o hardware. Isto elimina a necessidade de comunicação da aplicação com o *driver* ou a API do sistema operacional, pois parte-se da premissa que o servidor de som fará tal comunicação. Entre os servidores de som, destacam-se o Jack, PulseAudio, Sound Flower, Rewire e DirectConnect. Aqui abordaremos o Jack, PulseAudio e SoundFlower, pois não está disponível publicamente documentação suficiente sobre o Rewire (software proprietário da Propellerheads) e o DirectConnect (software proprietário da Digidesign).

O **Jack** (JACK, 2012) é um servidor de áudio e MIDI de baixa latência que funciona em tempo real. Seu nome é um acrônimo recursivo (JACK = Jack Audio Connection Kit). O Jack permite que aplicações conectem a sua saída de áudio na entrada de outra aplicação, criando assim o compartilhamento de dados de áudio entre aplicações. O Jack é multiplataforma e funciona em GNU/Linux, Solaris, FreeBSD, OS X e Windows. Existe também o Jack2 (LETZ, ARNAUDOV e MORET, 2009), que apesar do nome não é uma versão mais nova do Jack, mas simplesmente uma versão deste servidor escrita em C++. O Jack foi implementado de forma a permitir sua execução sobre APIs de áudio como ALSA, PortAudio, CoreAudio, FFADO e OSS. Ele ainda permite utilizar o netjack como *driver*, o que permite a interconexão de instâncias de Jack em máquinas distintas por meio do protocolo UDP, bem como a utilização de um *driver dummy*, para o processamento de sinais sem comunicação de E/S com hardware. Vários softwares foram escritos para funcionar sobre o Jack, como, por exemplo,

Ardour, Rosegarden, CSound, Pure Data, SuperCollider e Audacity (veja a lista completa em http://jackaudio.org/applications).

O **PulseAudio** (FREEDESKTOP.ORG, 2012) foi desenvolvido como um servidor de som para Linux, mas também foi migrado para Solaris, FreeBSD, NetBSD, MacOS X, Windows 2000 e Windows XP. O PulseAudio possui nativamente conexão UDP para troca de fluxos em rede e permite controle de volume por aplicação. Nativo em várias versões de Linux (Fedora, Ubuntu, Mandriva, Linux Mint, openSUSE), o PulseAudio funciona sobre o ALSA, OSS e ESD. Apesar de sua documentação descrevê-lo como um sistema de tempo real, o mesmo não oferece conexões para troca de fluxos entre aplicações de áudio distintas, e por essa razão não costuma ser muito utilizado para aplicações profissionais de áudio.

O **SoundFlower** (CYCLING74, 2012) foi desenvolvido na empresa Cycling 74, pelo mesmo time que fez o MAX/MSP. Este servidor de som permite que aplicações troquem fluxos de áudio ou MIDI, porém está disponível apenas para Mac OS X.

Tabela 2: Comparação entre servidores de som

API	MIDI	Linguagem	Multiplataforma	Licença
Jack	Sim	C / C++	Sim	GNU GPL / LGPL
PulseAudio	Não	C	Sim	GNU LGPL-2.1
Sound Flower	Sim	C++	Não	GNU GPL v2

O PulseAudio propõe uma solução elegante de servidor de som multiplataforma porém o fato de não suportar a conexão de *streams* entre aplicações impede a sua utilização como servidor de som para o desenvolvimento de aplicações modulares. O SoundFlower possui uma boa solução para isto mas seu uso é restrito ao Mac OS X. Por esta razão o Jack é atualmente o servidor de som mais utilizado para aplicações musicais multiplataforma.

Plugins

Muitas ferramentas são feitas de forma extensível por meio de *plugins*. *Plugins* de áudio podem ser integrados diretamente a ferramentas de alto nível como Audacity, Ardour, ProTools, Cubase, Rosegarden, Traverso, entre outros.

Um software que permite sua extensão por *plugins* é chamado de *host* do *plugin*. Um *host* pode aceitar uma ou mais arquiteturas de *plugins*. Entre as arquiteturas de *plugins* disponíveis atualmente podemos citar LADSPA, DSSI, LV2, Audio Unit, RTAS, VST / VSTi e DirectX.

O **LADSPA** (Linux Audio Developer's Simple Plugin API) (FURSE, 2000) é a primeira plataforma de plugins para o Linux. Várias aplicações de áudio para Linux funcionam como *host* para o LADSPA, sendo o *host* responsável pela interface gráfica do *plugin*. O programador LADSPA utiliza-se de dicas (*hints*) para que o *host* apresente corretamente os controles do *plugin*. Apesar de ter sido escrita para o Linux, vários *plugins* LADSPA foram migrados para Windows e Mac OS pela equipe de desenvolvimento do Audacity. O LADSPA não aceita MIDI e sugere que seja utilizado o DSSI (Disposable Soft Synth Interface) (DSSI, 2012) para tal propósito.

O **LV2** (HARRIS, 2008) é a Versão 2 do LADSPA. Esta arquitetura de *plugins* surgiu para complementar o LADSPA adicionando a esta plataforma novas funcionalidades. Estre estas funcionalidades está a possibilidade de o desenvolvedor do *plugin* escrever sua interface

gráfica em Qt ou GTK, além da capacidade de processar MIDI.

O **AudioUnit** (AU) (APPLE, 2012) é uma arquitetura de *plugins* proprietária para Mac OS X desenvolvida pela própria equipe da Apple sobre a API do Core Audio. Os *plugins* AU possuem interface gráfica própria e são suportados por vários *hosts* neste sistema operacional.

O **RTAS** (Real Time AudioSuite) (AVID, 2012) é uma arquitetura proprietária de *plugins* desenvolvida pela Avid Technology (Digidesign). Esta empresa, fabricante das interfaces Digi e M-Audio, disponibilizou esta arquitetura de *plugins* para garantir uma maior eficiência de suas interfaces de áudio. Há alguns tipos diferentes de *plugins* RTAS para diferentes versões de interfaces, e os mesmos estão disponíveis para MacOS e Windows em máquinas que possuam estas interfaces de áudio.

Feitos pela empresa Steinberg, os *plugins* **VST** (Virtual Studio Technology) (STEINBERG, 2012) podem possuir interfaces próprias para o usuário e possuem versões para Mac OS X, Windows e migrações para Linux. O desenvolvimento de um *plugin* VST depende da utilização do SDK da Steinberg.

A Microsoft também possui uma API proprietária para plugins chamada **DirectX** (MICROSOFT, 2012). O desenvolvimento de um *plugin* DirectX depende da utilização do SDK fornecido pela Microsoft.

A tecnologia de *plugins* está ainda muito atrelada a um sistema operacional. Além disto, a necessidade de desenvolver um *plugin* para um *host* específico pode acabar por limitar a escolha de uma API. Infelizmente não encontramos nenhum iniciativa para o desenvolvimento de uma API comum a todos os *hosts* e sistemas operacionais.

Tabela 3: Comparação entre plugins de áudio

API	MIDI	Linguagem	Licença	Multiplataforma
LADSPA	Não	C	LGPL	Sim
LV2	Sim	C	LGPL	Sim
AU	Sim	C++	Proprietária	Não
RTAS	Sim	C++	Proprietária	Sim
VST/VSTi	Sim	C++	Proprietária	Sim
DirectX	Sim	C++	Proprietária	Não

Conclusão

Este artigo apresentou uma visão de arquitetura de áudio para computadores visando a possibilidade de escolha de diferentes APIs para a implementação de aplicações de áudio. As APIs permitem um paradigma de desenvolvimento em que o desenvolvedor pode se concentrar apenas nas tarefas de processamento de sinais ao invés de se preocupar com implementações de mais baixo nível (WALSH, 2011).

Partindo da abordagem arquitetural, várias APIs foram apresentadas para o desenvolvimento de aplicações. Esta apresentação pode sugerir o quanto a escolha da API irá influenciar a interação entre a aplicação desenvolvida e um ecossistema de aplicativos já existente. As tabelas comparativas foram feitas com o objetivo de simplificar a visão sobre as

características das APIs. Pontos importantes podem ser considerados como, por exemplo, a portabilidade, a licença do software e as linguagens de programação pelas quais estas APIs se tornam disponíveis ao programador.

Evidentemente este trabalho não esgota todos os cenários de APIs de desenvolvimento; há outros cenários importantes, embora ainda mais específicos, como a extensão de aplicativos já existentes, como, por exemplo, o Pure Data e Max/MSP, por meio de seus *externals* escritos em C, ou ainda através da construção de *opcodes* para Csound.

Foi notado o uso predominante das linguagens C e C++ para a utilização das APIs apresentadas. O domínio de tais linguagens é um importante passo em direção ao desenvolvimento de aplicações profissionais para processamento de áudio.

Indo além do escopo deste trabalho, pode-se observar que o funcionamento interno das APIs aqui apresentadas é bastante similar entre elas; por exemplo, todas possuem funções de inicialização, uma função que é chamada a cada ciclo de processamento em blocos, e funções de finalização. Será um trabalho futuro comparar estas APIs a partir de seu código-fonte, para apresentar comparações quanto à dificuldade de implementação de uma mesma tarefa de processamento. ou quanto à qualidade da documentação disponível para o programador.

Agradecimento

Esta pesquisa é realizada com o apoio do CNPq, CAPES e FAPESP.

Referências

APPLE. **Audio Unit Programming Guide: Introduction.** 2012. Disponível em: https://developer.apple.com/library/mac/#documentation/MusicAudio/Conceptual/AudioUnitProgrammingGuide/Introduction/Introduction.html. Acessado em 25/04/2012.

AVID. **Avid — Audio Plug-In Developer Program.** 2012. Disponível em: <http://www.avid.com/us/partners/audio-plugin-dev-program>. Acessado em 25/04/2012.

BENCINA, R. **PortAudio and Media Synchronisation- It's All in the Timing.** 2003. Disponível em: http://www.rossbencina.com/static/writings/portaudio_sync_acmc2003.pdf Acessado em 08/06/2012

BENCINA, R.; BURK, P. **Portaudio - an open source cross platform audio api.** Ann Arbor, MI: Scholarly Publishing Office, University of Michigan Library, 2001. Disponível em: http://www.rossbencina.com/static/writings/portaudio_icmc2001.pdf Acessado em 08/06/2012.

CREATIVE. **Home - OpenAL.** 2012. Disponível em: http://connect.creativelabs.com/openal/default.aspx. Acessado em 25/04/2012.

CYCLING74. **Download Soundflower Cycling 74.** 2012. Disponível em: http://cycling74.com/soundflower-landing-page/. Acessado em 25/04/2012.

DSSI. ***DSSI.*** 2012. Disponível em: <http://dssi.sourceforge.net/>. Acessado em 25/04/2012.

FREEDESKTOP.ORG. **freedesktop.org - Software/PulseAudio.** 2012. Disponível em: http://www.freedesktop.org/wiki/Software/PulseAudio. Acessado em 25/04/2012.

FURSE, R. **Linux Audio Developer's Simple Plugin API (LADSPA).** 2000. Disponível em: http://www.ladspa.org/. Acessado em 25/04/2012.

HARRIS, D.R.S. **LV2 Track** 2008. Disponível em: http://lv2plug.in/trac/.Acessado em 25/04/2012.

JACK. **JACK: Connecting a world of audio.** 2012. Disponível em: http://jackaudio.org/. Acessado em 25/04/2012.

LAZZARINI, V., WALSH, R. **Developing LADSPA plugins with Csound.** In: LAC (Ed.). *Proceedings of Linux Audio Conference.* 2007, p.60-63.

MACMILLAN, K.; DROETTBOOM, M.; FUJINAGA, I. **Audio latency measurements of desktop operating systems.** In Education, (sn, 2011), p.259-262

MICROSOFT. **Download: DirectX End-User Runtime - Microsoft Download Center - Download Details.** 2012. Acessado em 25/04/2012. Disponível em: http://www.microsoft.com/download/en/details.aspx? Id=35.

PORTAUDIO. **PortAudio - an Open-Source Cross-Platform Audio API.** 2012. Disponível em: http://www.portaudio.com/. Acessado em 25/04/2012.

RAW, M. S. L. **Raw Material Software.** 2008. Disponível em: http://www.rawmaterialsoftware.com/juce.php. Acessado em 25/04/2012.

SCAVONE, G. P. **Rtaudio: A cross-platform c++ class for realtime audio input/output.** In: *in Proceedings of the 2002 International Computer Music (ICMC'02)*. , 2002. p. 196–199.

SCAVONE, G.P. **The RtAudio Home page.** 2012. Disponível em: http://www.music.mcgill.ca/~gary/rtaudio/. Acessado em 25/04/2012.

SCAVONE, G. P.; COOK, P. R. **Rtmidi, rtaudio, and a synthesis toolkit (stk) update.** In: *In Proceedings of the International Computer Music Conference (ICMC'05)*. 2005. p.1-4.

SDL. **Simple DirectMedia Layer.** 2012. Disponível em: http://www.libsdl.org. Acessado em 25/04/2012.

S.LETZ; N.ARNAUDOV; R.MORET. **What's new in JACK2?** In: LAC (Ed.). *Proceedings of Linux Audio Conference*. 2009. p.1-9.

STEINBERG. **Home : Welcome to Steinberg — http://www.steinberg.net/.** 2012. Disponível em: http://www.steinberg.net/en/home.html. Acessado em 25/04/2012.

WALSH, R. **Audio Plugin development with Cabbage** In: LAC (Ed.). *Proceedings of Linux Audio Conference*. 2011. pp.47-53.

WANG, Y.; STABLES, R.; REISS, J. **Audio latency measurement for desktop operating systems with onboard soundcards.** In: Audio Engineering Society Convention 128. 2010. Disponível em: http://www.aes.org/e-lib/browse.cfm? elib=15378. Acessado em 25/04/2012.