SIMULATION OF GUITAR SOUND EFFECTS ON MOBILE DEVICE WITH ANDROID

Tomáš Mészáros
Bachelor Degree Programme (4), FIT BUT
E-mail: xmesza03@stud.fit.vutbr.cz

Supervised by: Vítězslav Beran
E-mail: beranv@fit.vutbr.cz

Abstract: This paper deals with real-time audio signal processing on mobile devices with the Android operating system. It also introduces the purpose and benefits of such application, and the main reasons why similar software is not available for this platform.

Keywords: Android, audio, sound effects, guitar effects, simulation, real-time, sound card, ALSA, sound processing, USB audio

1 INTRODUCTION

The idea of this project is a sound processing application for the Android platform to enhance the sound of musical instruments. Similar applications are available for iPhone and iPad, but not for Android due to its hardware inconsistency and lack of API support. This work focuses on the implementation aspects of such application, especially on the Android OS. The result is going to be a framework to create applications for real-time sound processing targeted to Android, but generally usable on many other systems as well. The outlined solution deals with USB-audio driver integration, low latency audio implementation, sound filter (plug-in) hosting, and many other platform specific barriers.

2 HARDWARE COMPONENTS

The audio source is a USB sound interface to which the instrument is connected. The sound is digitalized and processed by the CPU of the mobile device. The result is sent back to the earphones jack or to the output jack of the USB-audio interface. The layout is illustrated on figure 1.

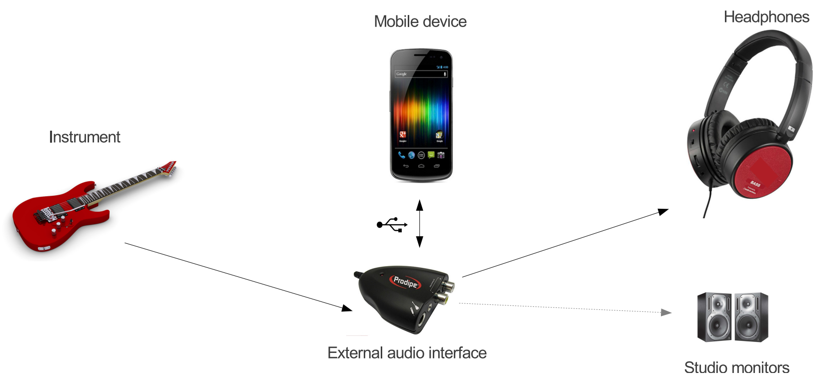

Figure 1: Hardware components and their connections

3 LOW LATENCY BARRIERS

Android is built on a Java VM implementation called Dalvik. All the applications run as a separate process inside an instance of the VM. This is the main barrier of real-time processing as the overhead of Dalvik prevents the platform to achieve unrecognisable latencies. The official Android SDK does not have direct support for low latency audio, not even with its native C/C++ level [3]. Fortunately, it is in progress and could be released in future versions. Nevertheless, a library is needed to control and utilize USB sound devices and a USB-audio driver that makes this possible. Most Android devices do not include such driver. The Linux kernel, which is the base of Android, uses the ALSA[1] audio architecture [2]. Its code base includes a generic USB-audio driver which can be compiled and loaded into the kernel as a module. ALSA is known to have sufficient latencies for real-time processing. Buffering can be minimized with adjusting specific hardware parameters in the driver.

The internal audio chip of the mobile device can be used as an output interface. It depends on the hardware parameters, mainly focused on the input and output latency, and whether the driver is compatible with ALSA.

4 BYPASSING THE ANDROID AUDIO STACK

ALSA has a default library called `alsa-lib` which is not compatible with Android due to specific IPC[2] implementation aspects. There are a few similar alternatives of this API. Some of them can be compiled with the native C/C++ tool-chain (NDK[3]). The Android source tree contains `tinyalsa`. This is a tiny version of `alsa-lib` not source-level compatible with the official alternative. Testing results revealed its weaknesses in conjunction with the requirements of this application. `Salsa-lib`, a lightweight version of `alsa-lib`, is more feature rich and more stable. Beside that, it has a source-level compatibility with the original API.

Part of this work is a layer which solves dependencies and can utilize the official NDK API as soon as the latency issues will be solved. The new layer is called `llaudio` (shown on figure 2), and it represents a wrapper for any specific audio library for accessing sound devices.

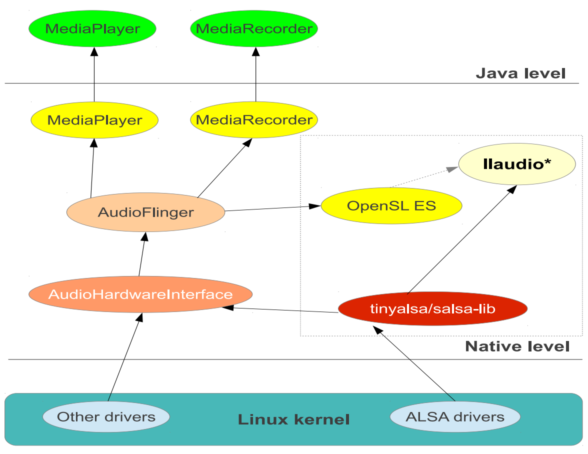

Figure 2: Android audio layers and libraries. [1]

[1]Advanced Linux Sound Architecture - divided to a kernel (drivers, part of Android) and a user-space (`alsa-lib`) layer

[2]Inter-process Communication

[3]Native Development Kit - optional part of the Android SDK for writing native C/C++ code

5 DESCRIPTION OF SOFTWARE COMPONENTS

The figure 3 shows the three main parts of the project. The idea is to use proprietary audio filters written with a specific framework like LADSPA[4] or VST[5]. An open source plug-in can be compiled and loaded to the application extending the number of available filters. The design allows a client-server relationship of the front-end and the plug-in hosting back-end communicating through an IPC mechanism. Building the back-end as a static or shared library is also possible. The client-server version was necessary to access root privileges in the Android environment. This architecture is more problematic taking in account the communication protocol and the processing overhead of message sending and receiving between the UI and the audio service.

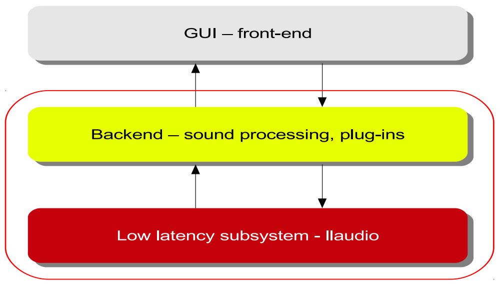
GUI – front-end
Backend – sound processing, plug-ins
Low latency subsystem - llaudio

Figure 3: Design stack - layers which are the subjects of this work

This design is very extendible in terms of front-end support and audio subsystems. It can be configured to work on various platforms (embedded or desktop) by implementing specific interfaces. A MIDI front-end, which is not included, would enable to receive MIDI messages from a foot controller and use the application while playing in a live situation.

6 CONCLUSION

The result of this work includes an extendible audio processing back-end which can be built as a stand-alone service or as a static/shared library for all platforms with C++ and multi-threading support. This architecture is completely independent from the overlying user interface. It solves the problem of low latency sound processing, audio plug-in hosting, and USB-audio support on Android. At the end it discusses further possibilities and potentials of the current design. The reference device used for testing was a Galaxy Nexus smart-phone and a Digitech RP250 audio interface.

7 ACKNOWLEDGEMENT

This work was partially supported by the BUT FIT grant FIT-S-11-2 and the research plan MSM0021630528.

REFERENCES

[1] Google. Audio subsystem [online]. `http://www.netmite.com/android/mydroid/development/pdk/docs/audio_sub_system.html`, 2008-06-09 [cit. 2013-03-20].

[2] Mattias Nagorni. Alsa programming howto [online]. `http://users.suse.com/~mana/alsa090_howto.html`, 2010-02-24 [cit. 2013-03-20].

[3] Sylvain Ratabouil. *Android NDK Beginner's Guide*. Packt Publishing, January 2012. ISBN-978-1-84969-152-9.

[4]Linux Audio Developer's Simple Plugin API
[5]Steinberg's Virtual Studio Technology framework