FAUST2ANDROID: A FAUST ARCHITECTURE FOR ANDROID

Romain Michon

CCRMA
Department of Music
Stanford Univeristy, CA 94305-8180
USA
rmichon@ccrma.stanford.edu

ABSTRACT

faust2android is a tool that turns a FAUST program into an Android application. Signal processing tasks as well as accessing the audio record and playback resources are done natively in C++ using the Android Native Development Toolkit (NDK). User interface and other components of the application are programmed in JAVA.

The implementation as well as issues related to real-time signal processing on Android platforms are discussed.

faust2android is part of a larger project whose goal is to build a full FAUST environment for Android: FAUSTDROID.

1. INTRODUCTION

While the iPhone and the iPad have been for the last few years privileged platforms to develop applications for real time signal processing, Android has been left behind, mainly because of the very bad latency perfomances it provided. However, recent developments[1] show that Google and others[2] seem to work at finding solutions to this problem.

Also, the possibility of connecting an external audio interface to an Android terminal seems to move forward. In fact, we were successfully able to use a Behringer GUITAR LINK UCG102[3] with a Google Nexus 7 without increasing the record and playback latency that was even reduced a little bit.

These different observations show that using Android terminals to do reliable real-time signal processing might be possible in a very near future. This was one of the main motivations for carrying out this work.

faust2android[4] is a tool that turns a FAUST[1] program into an Android application. It uses a script that embeds the C++ code generated by the FAUST compiler in a template Android app whose content is totally dynamic.

faust2andoid is part of a larger project (FAUSTDROID) that aims at providing an environment where musicians can easily access an online catalog of FAUST objects, download them, arrange them and use them on their Android terminal.

[1] http://developer.android.com/about/versions/jelly-bean.html.

[2] http://www.sonomawireworks.com/.

[3] http://www.behringer.com/EN/Products/UCG102.aspx.

[4] faust2android is available in the /tools/ directory of the FAUST repository: http://sourceforge.net/projects/faudiostream/.

2. ANDROID AND REAL TIME SIGNAL PROCESSING

2.1. The Latency Question

Android has always been infamous in the audio developer community for its very high latency for audio playback and recording. However, with the most recent release of this operating system (Jelly Bean 4.2), Google has made a step toward latency reduction. On his blog[5], Victor Lazzarini reports in a post from December 2012 that he was able to achieve a "round-trip latency" of 100ms and a "touch-to-sound latency" of 120ms. We obtained similar results with faust2android on a Nexus 7 (105ms for the "round-trip latency" and 130ms for the "touch-to-sound latency"). In both cases, these performances are greatly surpassing the one of the previous versions of Android where latency was generally larger than 300ms.

2.2. C or JAVA?

Android applications are mainly programmed in JAVA and the Android SDK provides an API for real-time audio recording and playback. Thus, signal processing classes can be directly implemented in JAVA which greatly simplifies the overall architecture of the app. Moreover, FAUST2 can now generate JAVA code instead of C++.

Several tests where various FAUST generated JAVA snippets code were "manually" embedded in an Android app were carried out on both a Samsung Galaxy S2 and a Google Nexus 7. While results varied greatly between the two devices (for example, we were not able to record and playback audio simultaneously in real-time on the Galaxy S2), they were very deceiving because of the instability of the process and the audio latency that was greater than 200ms.

On his blog, Victor Lazzarini describes in a post from March 2012[2] a technique to do Android audio streaming with OpenSL ES[6] and Android's Native Development Toolkit[7] (NDK). After several tests, this technique proved to be far more stable than the "full JAVA" one and was used to build faust2android.

[5] http://audioprograming.wordpress.com/category/android/.

[6] *Open Sound Library for Embedded Systems*: http://www.khronos.org/opensles/.

[7] http://http://developer.android.com/tools/sdk/ndk/.

2.3. Real-time audio with faust2android

As mentioned in 2.2, the Android NDK makes possible the use of functions written in `C` or `C++` in a `JAVA` app by wrapping them as a shared library using `SWIG`[8] that creates the elements to interface these two programming languages.

In an app generated by `faust2android`, the different tasks are shared between `C++` and `JAVA` as follow:

JAVA	C++
- Android application - dynamic user interface - send the values of the different DSP parameters at every audio frame	- DSP (process one audio frame) - information about the DSP parameters and the user interface - audio resources management

Accessing and managing audio resources is carried out using Victor Lazzarini's simple but very useful API[9] that makes available OpenSL ES on Android for real-time audio recording and playback. As a result, Android apps generated by `faust2android` operate audio streaming and signal processing natively which is far more efficient than if these tasks were done directly in `JAVA`.

3. FAUST2ANDROID IMPLEMENTATION

3.1. Generating the Code

Unlike other FAUST architectures, `faust2android` can't generate a single file containing all the elements needed by the C compiler to create an object. Indeed, as mentioned before, the generated apps are based on `JAVA`, `C++` and `XML` files which doesn't make the task easier.

`faust2android` uses a simple `bash` script to carry out the different tasks that will turn a FAUST program into an Android application. It first calls the FAUST compiler that generates `C++` code. This code is then embedded into an architecture file that interfaces it with a template Android app whose content is dynamically changed according to the user interface specifications contained in the `C++` code produced by FAUST.

Finally, the Android cross compiler is called by the script to generate the binary file of the app. A simple option allows to load the app on the default Android device connected to the computer that execute the script. Another option creates an eclipse project in the current directory if the user wishes to "manually" modify the content of the app.

An overview of `faust2android` is given in Figure 1.

3.2. User Interface and Parameters Control

Although the diversity of the standard user interface widgets provided with the Android SDK is rather limited, it is currently used to build the different parameter controllers of an app generated by `faust2android`. Indeed, while standard FAUST architectures allow the creation of vertical and horizontal sliders, knobs, digital entries, buttons, check boxes and vertical and horizontal groups, only horizontal sliders, digital entries, buttons, check boxes and vertical and horizontal groups can be currently used with `faust2android`. Therefore, if a vertical slider or a knob is declared in the FAUST code, it will be automatically converted to a horizontal slider.

[8]Simplified Wrapper and Interface Generator: `http://www.swig.org/`.

[9]`https://bitbucket.org/victorlazzarini/android-audiotest`.

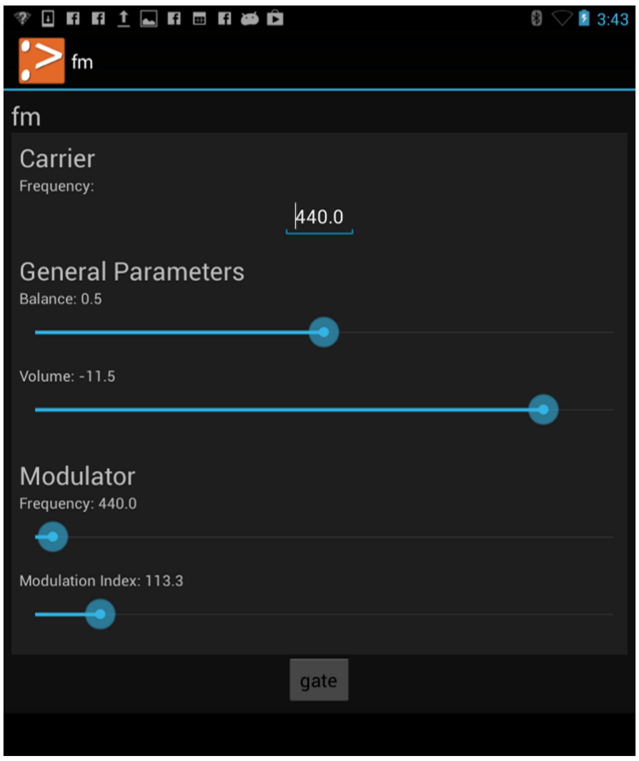

Figure 2: *Example of a user interface generated by* `faust2android` *running on a Google Nexus 7.*

Figure 2 shows an example of a user interface of an app generated by `faust2android` with the following code that implements a simple FM synthesizer:

```
import("music.lib");
import("filter.lib");

freqMod = hslider("v:Modulator/Frequency",
    440, 20, 15000, 1);
modIndex = hslider("v:Modulator/Modulation
    Index", 0, 0, 1000, 0.1);
freq = nentry("v:Carrier/Frequency", 440,
    20, 8000, 1);
vol  = hslider("v:General Parameters/Volume"
    , 0, -96, 0, 0.1) : db2linear;
bal  = hslider("v:General Parameters/Balance
    ",0.5,0,1,0.1);
gate = button("gate");

process = osc(freqMod)*modIndex+freq : osc
    * gate * vol <: *(bal),*(1-bal);
```

Finally, every parameter of a `faust2android` app can be controlled by one of the axes of the built-in accelerometer of the Android device. To do so, the user just has to touch the name of one of the parameters which will open the accelerometer

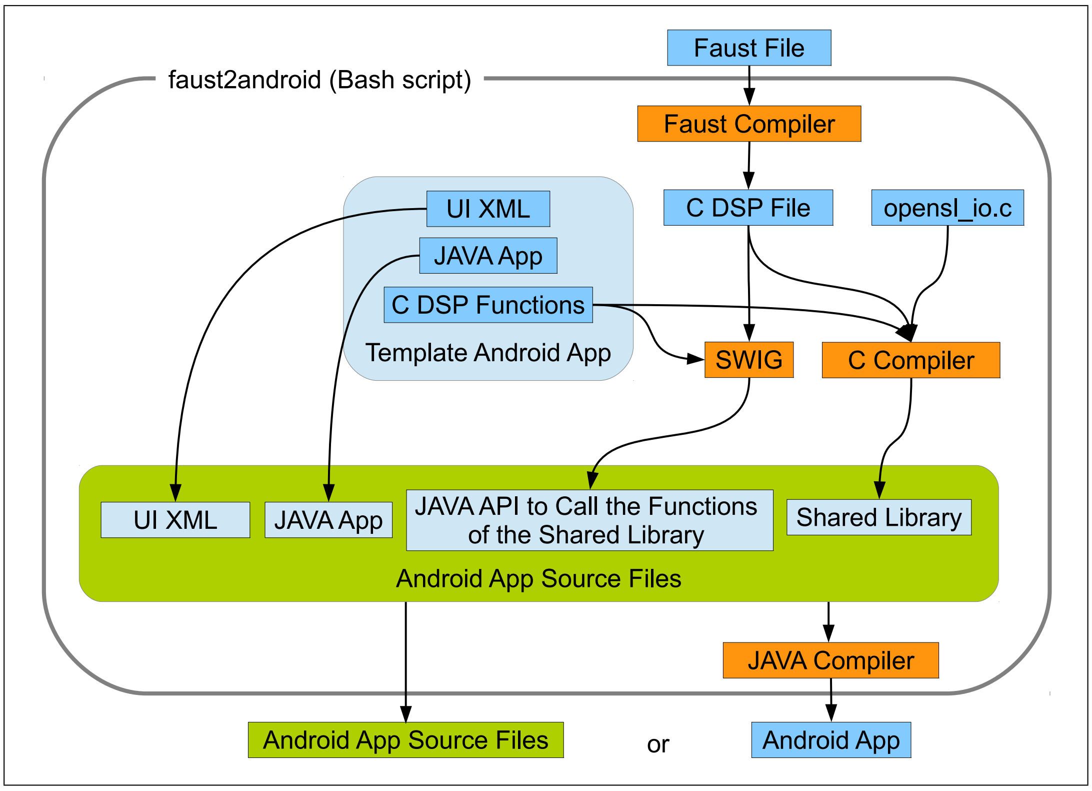

Figure 1: `faust2android` *overview.*

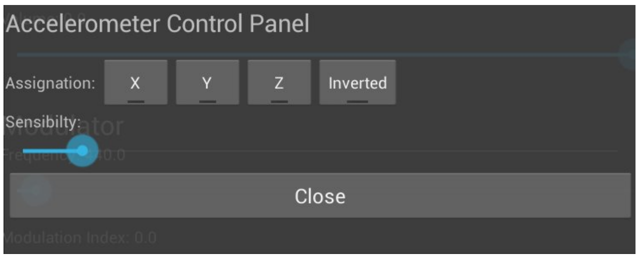

Figure 3: *Accelerometer assignement control pannel.*

assignment control panel that can be viewed in Figure 3.

4. LOOKING FORWARD

`faust2android` was the first step made in the framework of a larger project that aims at filling the gap between musicians and the open source FAUST community. This system will be based on an android application: FAUSTDROID where users will easily access an online catalog of FAUST objects, download them on their Android terminal and finally connect them together and use them through a user interface as simplified as possible. This project is summarized in Figure 5.

4.1. FAUSTDROID

Recent developments made at Grame around the FAUST Online Compiler [3] make the use of a RESTFull API to remotely compile FAUST code possible for different architectures and operating systems[10]. Also, the FAUST Online Compiler provides a catalog of FAUST objects that can be edited by anyone to add new items.

FAUSTDROID will provide users with an environment that will easily access the catalog of objects of the FAUST Online Compiler, compile them using `faust2android` and use the resulting signal processors in a workspace where they could be arranged and plugged together.

4.2. The FAUSTBOX

Another important feature of the FAUSTDROID project is the FAUSTBOX. This device will be based on an Arduino Uno and an Arduino Wifi Shield[11] (see Figure 4) and will be battery-powered. It will enable the use of sensors to control the different parameters of the object generated by FAUST running in FAUSTDROID.

The FAUSTBOX will send OSC messages through WIFI to communicate with FAUSTDROID.

Finally, every sensor will be powered and output its signal through stereo 1/8 inches jacks that will be easily connectable to the FAUSTBOX. These signals will be in the same range in order to easily permute the sensors.

[10]`git://faudiostream.git.sourceforge.net/gitroot/faudiostream/FaustWeb`.
[11]`www.arduino.cc`.

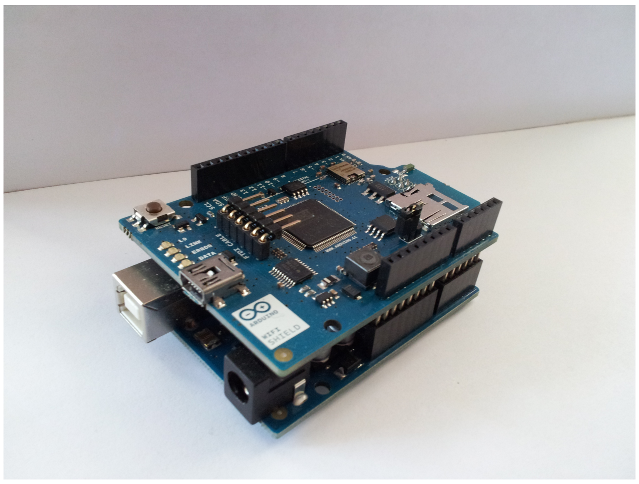

Figure 4: *Arduino Uno with a Wifi Shield.*

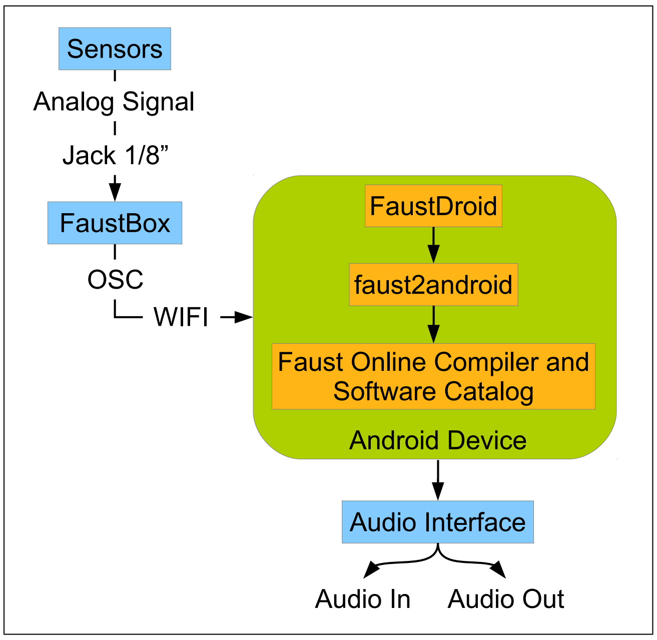

Figure 5: *Summary of the* FAUSTDROID *project.*

5. CONCLUSION

While dozens of Android applications for asynchronous signal processing are available on the Google Play Store[12], only a few of them allow real-time signal processing. It is easy to guess that the reason of this lack is related to the audio playback latency problems of Android.

However, it seems that Google is willing to solve this problem and made a step toward lower latency in the most recent Android release.

We can guess that the more people will show interest in this topic, the more Google will try to increase the performances of their operating system for real-time audio recording and playback.

`faust2android` is just one more brick in the wall and we can hope that more works involving real-time signal processing on Android will be carried out in the future.

6. REFERENCES

[1] Yann Orlarey, Dominique Fober, and Stephane Letz, "An algebra for block diagram languages," in *Proceedings of the International Computer Music Conference (ICMA)*, Gothenburg, Sweden, 2002, pp. 542–547.

[2] Victor Lazzarini, "Android audio streaming with opensl es and the ndk," in *The Audio Programming Blog*, March 2012, http://audioprograming.wordpress.com/2012/03/03/android-audio-streaming-with-opensl-es-and-the-ndk/.

[3] Romain Michon and Yann Orlarey, "The faust online compiler: a web-based ide for the faust programming language," in *Proceedings of the Linux Audio Conference (LAC)*, Stanford University, USA, 2012, pp. 111–116.

[4] Romain Michon and Julius Smith, "Faust-stk: a set of linear and nonlinear physical models for the faust programming language," in *Proceedings of the Conference on Digital Audio Effects (DAFx-11)*, IRCAM, Paris, France, 2011, pp. 199–204.

[12]https://play.google.com/store.