FAUST2ANDROID : UNE ARCHITECTURE FAUST POUR ANDROID

Romain Michon

▶ **To cite this version:**

Romain Michon. FAUST2ANDROID : UNE ARCHITECTURE FAUST POUR ANDROID. Journées d'Informatique Musicale, May 2013, Paris, France. hal-03112218

HAL Id: hal-03112218
https://hal.science/hal-03112218

Submitted on 16 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

FAUST2ANDROID : UNE ARCHITECTURE FAUST POUR ANDROID

Romain Michon
CCRMA
Department of Music
Stanford University, CA 94305
USA
rmichon@ccrma.stanford.edu

RÉSUMÉ

faust2android est un programme qui permet de compiler un code FAUST en une application pour terminal Android. Les tâches de traitement du signal ainsi que l'accès aux ressources audios se font de manière native en *C++* grâce au *Native Development Toolkit (NDK)* d'Android. L'interface utilisateur et les autres éléments de l'application sont programmés en *JAVA*.

faust2android fait partie d'un projet plus large qui vise à implémenter un environnement d'utilisation d'objets FAUST sous Android : FAUSTDROID.

1. INTRODUCTION

Alors que les iPads et les iPhones ont été au cours des dernières années des plates-formes privilégiées pour le développement d'applications pour le traitement du signal en temps réel, les terminaux Androids ont été délaissés, principalement à cause de l'importante latence audio générée par leur système d'exploitation. Cependant, de récents développements [1] montrent que Google ainsi que d'autres entreprises [2] semblent s'intéresser à résoudre ce problème.

Une autre avancée dans le domaine de l'audio pour Android concerne la possibilité de pouvoir connecter une interface audio externe sur un terminal Android. Bien qu'il soit encore loin d'être simple d'accéder dans une application Android à une ressource audio USB externe, nous avons été en mesure d'utiliser une interface Behringer GUITAR LINK UCG102 [3] sur une tablette Google Nexus 7 sans augmenter la latence audio et en la réduisant même de quelques millisecondes.

Ces différentes observations montrent que l'utilisation fiable de terminaux Android pour le traitement du signal en temps réel devrait devenir possible dans un futur proche. Cela à grandement motivé les travaux présentés dans cet article.

faust2android [4] est un programme qui permet de générer une application Android à partir d'un code FAUST[3]. Il utilise un script qui place le code *C++* généré par le compilateur de FAUST dans une application Android « modèle » au contenu dynamique.

faust2android a été implémenté dans le cadre d'un projet plus large : FAUSTDROID. Cette application va permettre de mettre à disposition du musicien un environnement dans lequel il sera possible de télécharger de manière simple des objets FAUST contenus dans un catalogue en-ligne, de les connecter entre eux et de les utiliser en temps réel sur un terminal Android.

2. ANDROID ET TRAITEMENT DU SIGNAL EN TEMPS RÉEL

2.1. La question de la latence

Android a toujours été connu par la communauté de développeurs de logiciels comme une plateforme inutilisable pour l'audio en temps réel à cause de l'importance de la latence qui la caractérisait. Cependant, ce problème a été récemment traité par Google dans la dernière version de son système d'exploitation (Jelly Bean 4.2). Les témoignages de développeurs sur cette amélioration sont nombreux. Par exemple, Victor Lazzarini explique dans un billet de décembre 2012 posté sur son blog [5] qu'il a été en mesure d'obtenir des latences inférieures à 100ms pour de simples traitements de son et inférieures à 120ms pour des synthétiseurs utilisant une interface graphique (« Touch to Sounds Latency »).

Nous avons pu obtenir avec *faust2android* des résultats similaires pour le traitement du son et meilleurs pour la synthèse avec contrôle via une interface graphique (environ 30ms). Bien qu'encore loin d'être suffisante pour une utilisation dans un contexte musical, ces performances surpassent de loin celles des précédentes versions d'Android qui offraient des latences supérieures

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* est disponible dans le dossier */tools/* du répositoire de FAUST : http://sourceforge.net/projects/faudiostream/. Il peut également être utilisé directement dans le compilateur en ligne de FAUST : http://faust.grame.fr/index.php/online-examples.
5. http://audioprograming.wordpress.com/category/android/.

à 300ms.

2.2. C ou JAVA ?

La plupart des applications Android est programmée en *JAVA* et le *SDK* de Android contient une API pour l'audio en temps-réel. Ainsi, une classe pour le traitement du signal peut être implémentée directement en *JAVA* ce qui permet de simplifier grandement l'architecture générale de l'application. De plus, FAUST2 permet maintenant de générer du code *JAVA* au lieu de *C++* ce qui est un argument supplémentaire pour le choix de *JAVA*.

Plusieurs tests lors desquels des codes *JAVA* générés par FAUST ont été manuellement copiés dans une application Android puis exécutés sur un téléphone Samsung Galaxy S2 et une tablette Google Nexus 7 ont été menés. Tandis que les résultats de ces expériences ont grandement variés d'un terminal à un autre (nous n'avons par exemple pas été en mesure d'accéder aux ressources pour l'acquisition audio sur le Galaxy S2), ils ont été de manière générale assez décevant, principalement à cause de l'instabilité des applications et de l'importance de la latence.

A l'inverse, Victor Lazzarini a décrit sur son blog dans un billet de mars 2012 [6] une technique permettant d'accéder aux ressources audios temps-réel d'un terminal Android de manière native en utilisant *OpenSL ES* [7] et le *NDK* [8]. Après plusieurs tests, cette technique s'est avérée être beaucoup plus efficace et stable que celle n'utilisant que du code *JAVA* et a été utilisée pour implémenter *faust2android*.

2.3. Audio en temps-réel avec faust2android

Comme cela a été mentionné dans la partie 2.2, le *NDK* de Android rend possible l'utilisation de classes *C++* dans une application *JAVA* en les compilant sous la forme d'une *shared library*. Cette tâche est grandement simplifiée par *SWIG* [9] qui crée les fichiers d'interface nécessaires entre les deux langages de manière presque automatisée.

Dans une application générée par *faust2android*, les différentes tâches sont réparties de la manière suivante entre *JAVA* et *C++* :

JAVA	C++
- application Android - interface utilisateur dynamique - envois des valeurs des différents paramètres DSP à chaque Frame	- traitement du signal (calcul d'une Frame audio) - informations sur les paramètres DSP et sur l'interface utilisateur - gestion des ressources audios

6. http://audioprograming.wordpress.com.

7. *Open Sound Library for Embedded Systems* : http://www.khronos.org/opensles/.

8. *Native Developement Toolkit* : http://http://developer.android.com/tools/sdk/ndk/.

9. *Simplified Wrapper and Interface Generator* : http://www.swig.org/.

La gestion des ressources audios en *C++* est menée à bien à l'aide de l'API [10] mise en place par Victor Lazzarini qui fournit des fonctions de haut niveau à *OpenSL ES* sur Android.

Pour résumer, une application générée par *faust2android* effectue les tâches de traitement du signal nativement ce qui est beaucoup plus efficace et stable que si cela était fait directement en *JAVA*.

3. IMPLÉMENTATION DE FAUST2ANDROID

3.1. Générer le code

A la différence d'autres architectures FAUST, *faust2android* ne peut pas générer un simple fichier *C++* contenant l'ensemble des éléments nécessaires à la création d'une application complète. En effet, comme cela a été expliqué précédemment, une application *faust2android* contient des fichiers *JAVA*, *C++* et *XML*.

faust2android utilise un simple script *Bash* pour mener à bien les différentes actions permettant de créer l'application. Dans un premier temps, ce dernier appelle le compilateur de FAUST qui produit du code *C++*. Ce code est alors copié dans un fichier d'architecture qui permet de l'interfacer avec une application Android « modèle » dont le contenu totalement dynamique s'adapte en fonction des informations contenues dans le code généré par FAUST.

faust2android peut également créer de manière optionnelle un projet éclipse si l'utilisateur souhaite modifier manuellement le contenu de l'application.

Enfin, le cross-compilateur d'Android est appelé par le script pour générer le fichier binaire de l'application. Une option permet de charger ce dernier sur une tablette ou un téléphone connecté en USB à la machine qui effectue ces opérations. Un résumé du fonctionnement de *faust2android* peut être vu dans la figure 1.

3.2. Interface utilisateur

Bien que la diversité des *widgets* fournis par défaut avec le *SDK* de Android soit assez limitée, ils sont actuellement utilisés pour construire les différents contrôleurs de paramètres d'une application générée par *faust2android*. En effet, alors qu'une architecture FAUST standard autorise la création de potentiomètres horizontaux et verticaux, de potards, de cadrans digitaux, de boutons et de groupes verticaux et horizontaux, à la date de la rédaction de cette article, *faust2android* ne permet la mise en place que de potentiomètres horizontaux, de cadrans digitaux, de boutons et de groupes horizontaux et verticaux.

Par conséquent, si un potentiomètre vertical ou un potard est déclaré dans un code FAUST, il est automatiquement converti en potentiomètre horizontal.

10. https://bitbucket.org/victorlazzarini/android-audiotest.

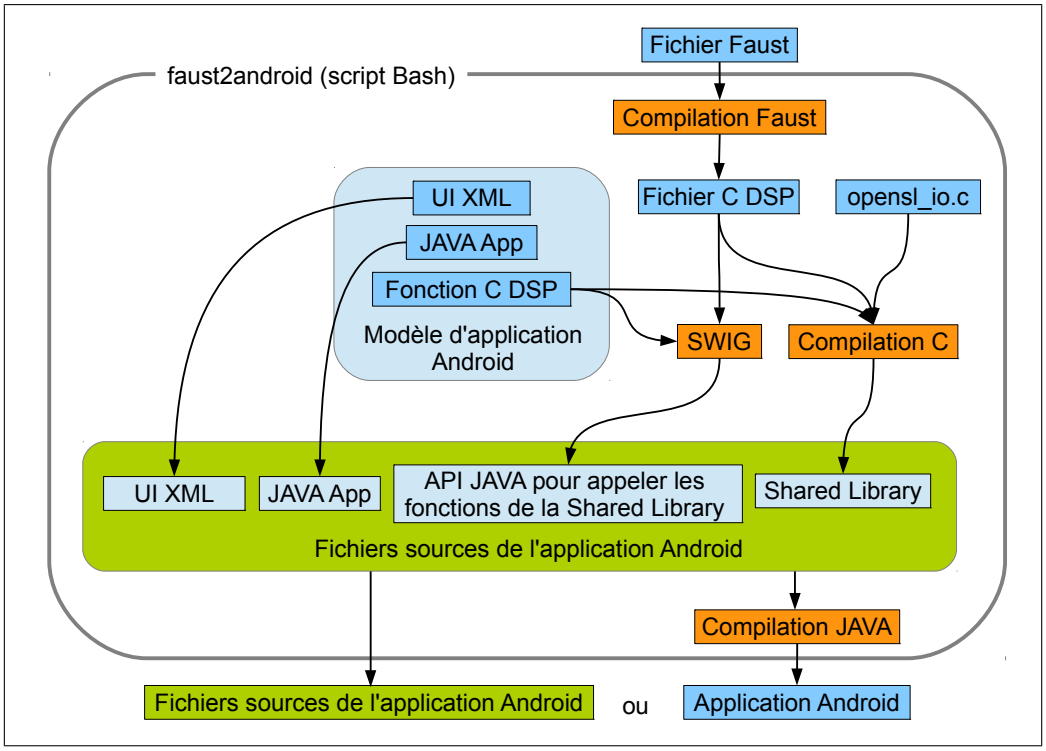

Figure 1. Résumé du fonctionnement de *faust2android*.

La figure 2 présente un exemple d'interface utilisateur générée à partir du code suivant qui implémente un simple synthétiseur FM :

```
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);
```

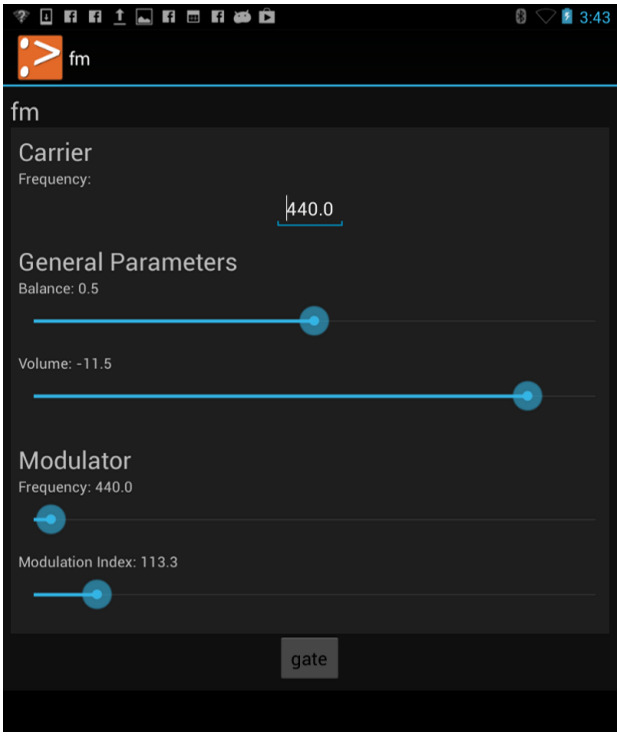

Figure 2. Exemple d'interface graphique d'une application Android générée par *faust2android*.

4. PROJETS FUTURS

faust2android a été développé dans le cadre d'un projet plus large visant à combler le vide entre les musiciens et la communauté FAUST de développeur *OpenSource* qui travaille entre autre à l'implémentation d'algorithmes de traitement du signal.

Ce système sera basé sur une application Android appelée FAUSTDROID où un utilisateur pourra facilement accéder un catalogue en-ligne à partir duquel il sera possible de télécharger des objets FAUST et de les utiliser

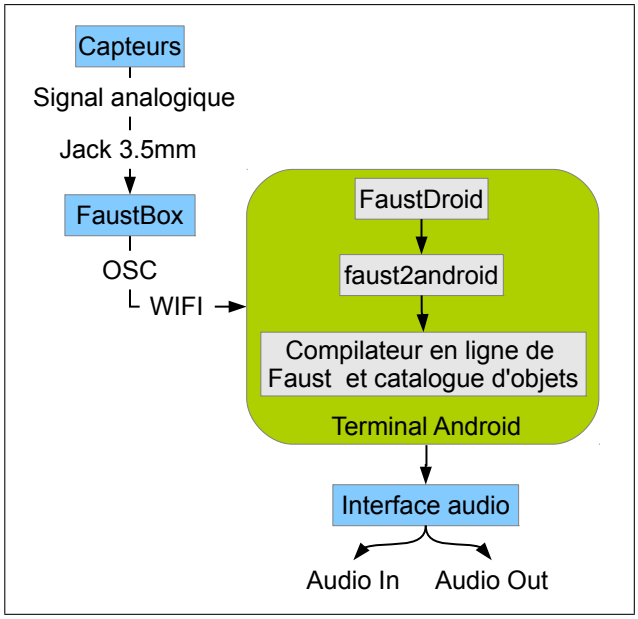

Figure 3. Résumé du projet FAUSTDROID.

en les connectant entre-eux et en les contrôlant via une interface graphique aussi simple que possible.

Ce projet est résumé par le diagramme présenté dans la figure 3.

4.1. FaustDroid

De récents développements menés à GRAME [11] autour du compilateur en ligne de FAUST permettent l'utilisation d'une API RESTFull pour compiler du code FAUST à distance pour différentes architectures et systèmes d'exploitation [12]. Le compilateur en ligne de FAUST met également déjà à disposition un catalogue d'objets pouvant être édité par n'importe quel internaute pour ajouter ou modifier des éléments.

FAUSTDROID offrira la possibilité d'accéder à ce catalogue de la manière la plus simple possible, de compiler certains de ces objets avec *faust2android* et d'intégrer l'application résultante dans un espace de travail où plusieurs objets pourront être assemblés et connectés.

4.2. La FaustBox

Un autre élément important du projet FAUSTDROID est la FAUSTBOX. Cet objet sera basé sur un Arduino Uno et un Arduino Wifi Shield [13] (cf. figure 4) et sera alimenté par une batterie. Elle permettra l'utilisation de capteurs pour contrôler les différents paramètres d'un objet FAUST fonctionnant dans FAUSTDROID.

La FAUSTBOX enverra des messages OSC au terminal Android via le WIFI pour communiquer avec FAUSTDROID.

11. http://grame.fr.
12. git://faudiostream.git.sourceforge.net/gitroot/faudiostream/FaustWeb.
13. www.arduino.cc.

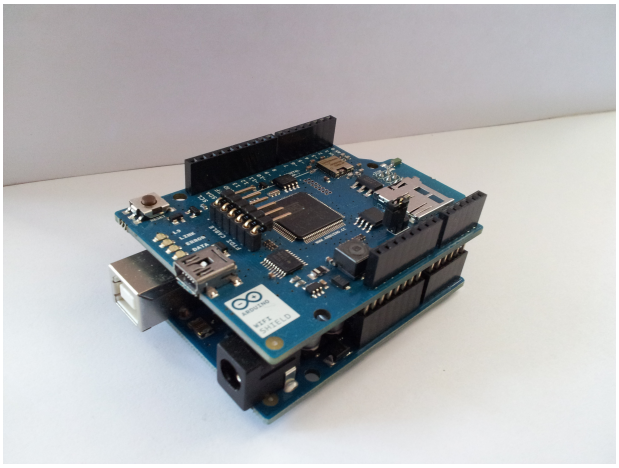

Figure 4. Arduino Uno with a Wifi Shield.

Enfin, chaque capteur sera alimenté en électricité et enverra son signal analogique via une prise Jack stéréo 3.5mm qu'il sera possible de connecter à la FAUSTBOX. Dans la mesure où les signaux des capteurs auront tous la même échelle, il sera possible de les interchanger très facilement.

5. CONCLUSION

Tandis que des dizaines d'applications pour le traitement du signal en temps différé sont actuellement disponibles sur le *Google Play Store* [14], seules quelques unes d'entre elles offrent des possibilités en temps réel. Il est aisé d'en déduire que l'origine de ce manque est due à la mauvaise réputation de Android concernant la latence audio.

Cependant, comme cela a été expliqué dans cet article, il semblerait que Google déploie beaucoup d'effort pour trouver une solution à ce problème comme le prouve la plus récente version d'Android.

Plus le nombre de personnes montrant de l'intérêt pour ce sujet sera important, plus Google s'attachera à fournir des solutions efficaces pour réduire la latence de son système d'exploitation afin de le rendre utilisable pour le traitement du signal en temps réel à des fins musicales.

faust2android est juste une pierre de plus à l'édifice et nous ne pouvons qu'espérer que d'autres travaux dans un futur proche contribueront à convaincre Google de réduire encore plus la latence audio d'Android.

6. REFERENCES

[1] Michon, R. ; Orlarey, Y., « The Faust online compiler : a web-based IDE for the Faust programming language », *Proceedings of the Linux Audio Conference (LAC)*, CCRMA – Université Stanford, USA, 2012.

14. https://play.google.com/store.

[2] Michon, R. ; Smith, J. « Faust-STK : a set of linear and nonlinear physical models for the Faust programming language », *Proceedings of the Conference on Digital Audio Effects (DAFx-11)*, IRCAM, Paris, France, 2011.

[3] Orlarey, Y. ; Fober, D. ; Letz, S. « An algebra for block diagram languages », *Proceedings of the International Computer Music Conference (ICMA)*, Gothenburg, Suède, 2002.