FAUSTLIVE
UN COMPILATEUR À LA VOLÉE POUR FAUST
... ET BIEN PLUS ENCORE

Sarah Denoux
GRAME
sdenoux@grame.fr

Stéphane Letz
GRAME
letz@grame.fr

Yann Orlarey
GRAME
orlarey@grame.fr

RÉSUMÉ

FaustLive est une application qui, grâce à son compilateur Faust embarqué, se propose de réunir le confort d'un langage interprété avec l'efficacité d'un langage compilé. Basée sur *libfaust*, une librairie qui offre une chaine de compilation complète en mémoire, FaustLive ne requiert aucun outil externe (compilateur, éditeur de lien, ...) pour traduire du code FAUST en code machine exécutable. Par l'intermédiaire de cette technologie, FaustLive offre de multiples fonctionnalités. Par exemple, il est possible de glisser un nouveau fichier DSP sur une application FAUST en fonctionnement pour remplacer son comportement et ce, sans interruption du son. Il est aussi possible de transférer une application qui fonctionne en local, sur une autre machine, même si celle-ci utilise un système d'exploitation différent.

1. INTRODUCTION

FAUST [Functional Audio Stream] [5] est un langage fonctionnel, dédié spécifiquement à la synthèse et au traitement du signal en temps réel. Comparativement aux autres langages existants comme Max, PD, Supercollider, Csound, Chuck etc, l'originalité de FAUST est de créer des programmes non pas interprétés mais entièrement compilés [1] . Il offre ainsi une alternative au C/C++ pour l'implémentation d'algorithmes DSP travaillant efficacement à l'échantillon près.

Alors que les compilateurs ont l'avantage de l'efficacité, ils présentent certains désavantages par rapport à des interpréteurs. D'abord, les compilateurs traditionnels ont besoin d'une chaine complète d'outils (compilateurs, éditeurs de liens, librairies de développement, ...). Pour des "non-programmeurs", l'installation de ces outils peut être assez complexe. De plus, le cycle de développement, depuis l'édition du code source jusqu'à l'application exécutée, est bien plus long pour un compilateur que pour un interpréteur. Pour un artiste en situation de création, l'expérimentation rapide est indispensable. Un long cycle de compilation peut donc être un frein. D'autre part, le code binaire n'est pas compatible entre différentes plateformes et systèmes d'exploitation.

FaustLive essaie de réunir le confort d'un langage interpreté autonome avec l'efficacité d'un langage compilé. Grâce à *libfaust*, une librairie qui offre une chaîne de compilation complète en mémoire, FaustLive ne requiert aucun outil externe pour traduire du code FAUST en code machine, et l'exécuter. Grâce à son cycle de développement très court, FaustLive se comporte sur de nombreux points comme un interpréteur FAUST (se rapprochant des environnements modernes LISP compilés, ou de l'approche présentée par A. Graef avec Pure dans [1]).

Basé sur ce court cycle de développement, FaustLive présente des fonctionnalités avancées. Il est, par exemple, possible d'éditer le code source pendant qu'une application FAUST tourne. Le code est alors recompilé dynamiquement, sans interruption du son. Si cette application utilise JACK comme driver, toutes les connexions sont maintenues. FaustLive offre donc une certaine flexibilité pour le prototypage des applications FAUST. D'autre part, lorsque FaustLive fonctionne sur un réseau, il est possible de transférer les calculs audio d'une application sur une machine distante, même si celle-ci utilise un système d'exploitation différent.

Les applications FAUST peuvent aussi être contrôlées à distance, en utilisant les protocoles HTTP ou OSC. Enfin, FaustLive peut se connecter à FaustWeb, un service de compilation distant pour exporter une application en un binaire traditionnel pour l'un des systèmes d'exploitation et une des architectures supportées par la distribution FAUST.

2. LE COMPILATEUR FAUST

Le compilateur FAUST traduit un programme FAUST en son équivalent dans un langage impératif (C, C++, Java, etc), s'occupant de générer du code efficace. La distribution FAUST inclut de nombreux fichiers d'architecture, faisant la liaison entre le code généré et le monde extérieur (drivers audio, interfaces de contrôle, ...).

La version officielle du compilateur FAUST transforme

1 . Les langages interprétés ont souvent recours à une phase de traduction, parfois appelée «compilation», permettant de traduire le programme en une représentation interne, par exemple du bytecode, plus efficace pour l'interprétation. Dans le domaine audio, les langages interprétés ont également recours à une autre technique, le groupage des échantillons par vecteur, de façon à amortir d'avantage le coût de l'interprétation. Néanmoins ces techniques ne permettent pas d'atteindre l'efficacité d'un programme véritablement compilé en code machine, comme c'est le cas pour les programmes FAUST.

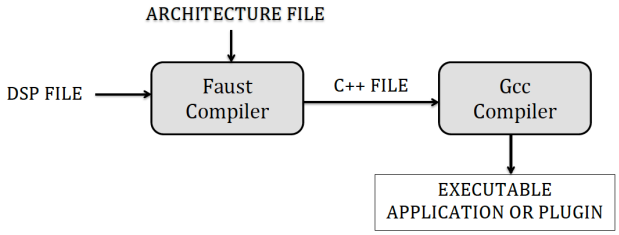

Figure 1. Étapes de la chaine de compilation FAUST

le code DSP en une classe C++, insérée ensuite dans une architecture. Le fichier C++ résultant est finalement compilé avec un compilateur standard pour produire une application ou un plugin (Figure 1).

L'application finale est alors structurée de la manière suivante : le DSP est représenté sous forme d'un module de calcul alors que l'architecture joue le rôle de lien vers le driver audio et vers l'interface graphique (Figure 2).

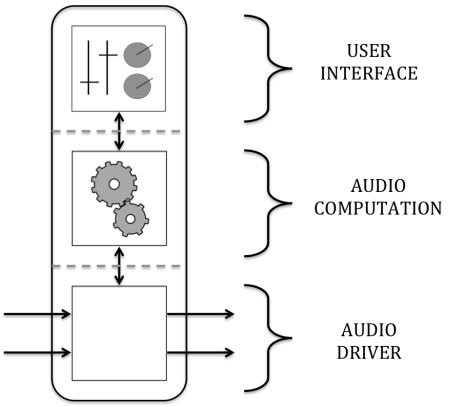

Figure 2. Structure d'une application FAUST

2.1. LLVM

LLVM (anciennement "Low Level Virtual Machine") est une infrastructure de compilateur. Elle est conçue pour compiler, éditer les liens et optimiser des programmes écrits dans un langage arbitraire. Le code exécutable est produit dynamiquement par un compilateur à la volée, à partir d'une représentation spécifique appelée LLVM IR (Représentation Intermédiaire). Clang, le compilateur natif LLVM pour le C/C++/Objective-C est un front-end pour le compilateur LLVM. Il peut, par exemple, convertir un fichier C en code LLVM IR (Figure 3).

Les langages spécialisés, comme FAUST, peuvent facilement produire du code LLVM IR. Un back-end spécifique FAUST vers LLVM IR a été ajouté au compilateur FAUST pour générer ce code, [4].

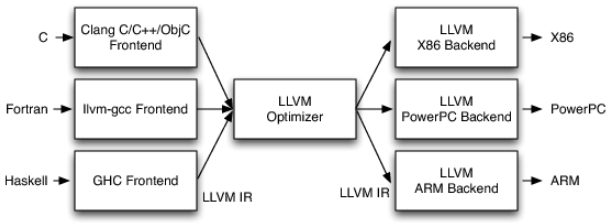

Figure 3. LLVM Compiler Structure

2.2. Chaîne de Compilation Dynamique

La chaîne de compilation complète démarre donc au code source DSP, compilé en LLVM IR en utilisant le back-end FAUST-LLVM, pour finalement produire du code exécutable grâce au compilateur LLVM-JIT. Toutes les étapes sont réalisées en mémoire. Des pointeurs sur les fonctions exécutables sont ensuite accessibles dans le module LLVM résultant, et leurs codes pourront être appelés avec les paramètres appropriés.

Dans la branche de développement *faust2*, le compilateur FAUST a été compilé sous forme d'une librairie embarquable, *libfaust*, publiée avec une API associée, [2]. Cette API imite le concept des langages orientés objet, comme le C++. L'étape de compilation, habituellement réalisée par GCC, est accessible au travers de la fonction createDSPFactory. À partir du code FAUST sous forme de string ou de fichier, la chaine de compilation embarquée (FAUST + LLVM-JIT) génère le "prototype" de la classe, appelée llvm-dsp-factory. Puis, la fonction *createDSPInstance*, correspondant à un "new nomClasse" du langage C++, permet d'instancier un llvm-dsp. L'instance peut ensuite être utilisée comme n'importe quel objet, fonctionner et être contrôlée à travers son/ses interfaces.

3. FAUSTLIVE - UTILISATION

FaustLive est un logiciel basé sur QT [2], qui permet de lancer des applications FAUST à partir de leur code source, sans avoir à les précompiler. La figure 4 présente donc la transformation des fichiers source FAUST baa.dsp et foo.dsp en applications JACK-QT, une fois déposés dans FaustLive.

FaustLive exploite la compilation dynamique, associée à différents systèmes d'interfaçage et drivers audio pour moduler la structure des applications FAUST.
Pour donner une idée du potentiel de FaustLive, la section qui suit présente ses diverses fonctionnalités, montrant pour chacune d'entre elles les altérations correspondantes dans la structure des applications.

Le point de départ de FaustLive est le glisser/déposer. Un DSP FAUST peut être ouvert dans une nouvelle fenêtre,

2 . framework pour la conception d'interfaces graphiques

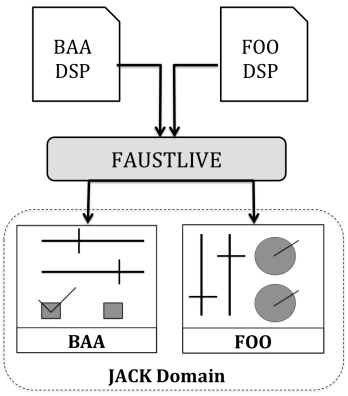

Figure 4. Principe de FaustLive

ou bien il peut être glissé sur une application en fonctionnement. Il apparaît alors un état intermédiaire pendant lequel les deux applications co-existent. L'application entrante copie les connexions audio établies, puis les deux comportements audio sont interpolés, via un crossfade (Figure 5). Pour finir, l'application entrante remplace durablement la précédente. Ce système permet de modifier indéfiniment l'application courante dans une fenêtre, en évitant les interruptions audio.

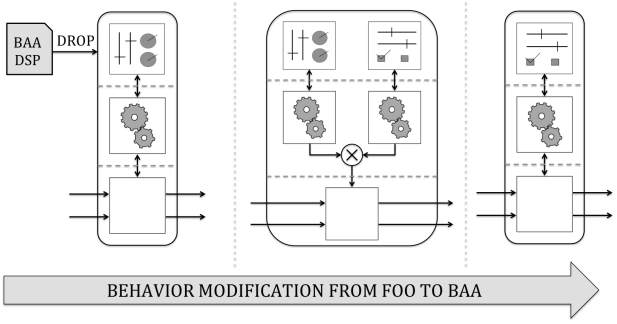

Figure 5. Interpolation de comportements

Ce mécanisme permet aussi l'édition du code source. Lorsqu'un utilisateur choisit d'éditer son code source, il est ouvert dans l'éditeur de texte par défaut. Puis, au moment où les modifications sont enregistrées, l'application est mise à jour, en utilisant l'interpolation de comportements. Cette fonctionnalité est centrale. Elle simplifie grandement le processus de prototypage : un utilisateur peut modifier son code à loisir et entendre/voir le résultat instantanément.

Originairement, JACK a été choisi comme driver audio, puisqu'il permet à l'utilisateur de connecter ses applications FAUST entre elles. Les drivers NetJack, CoreAudio et PortAudio ont ensuite été intégrés à FaustLive donnant à l'utilisateur la possibilité de changer dynamiquement de driver. L'application n'a pas besoin d'être quittée. La migration est réalisée pendant l'exécution de FaustLive et s'applique à toutes les applications FAUST en fonctionnement (Figure 6).

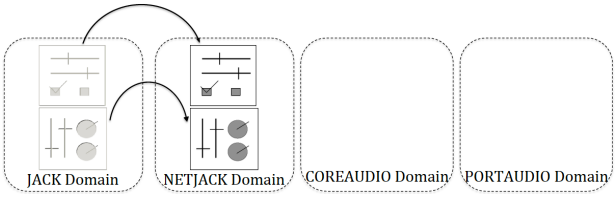

Figure 6. Migration des applications vers un nouveau driver audio

Afin d'offrir une application modulaire, FaustLive offre plusieurs choix d'interfaces de contrôle. Le protocole OSC [3] est intégré à FaustLive pour fournir plus d'interopérabilité. De nombreux environnements et matériels audio implémentent ce protocole, grâce auquel FaustLive pourra communiquer avec eux. Un smartphone peut alors ouvrir une interface OSC, contrôlant l'application à distance (Figure 7).

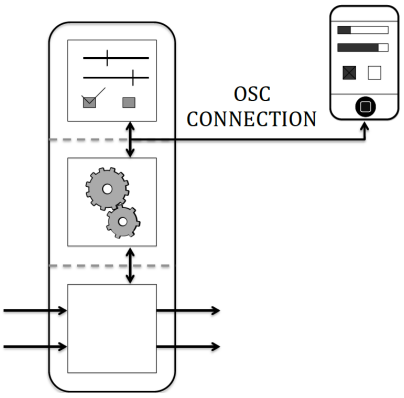

Figure 7. Interface OSC

De même, une interface HTML est accessible à partir d'un QrCode [4] . Une fois scanné, par exemple par une tablette, l'interface est ouverte dans un navigateur. Dans le cas d'OSC comme dans celui d'HTTP, l'interface est dupliquée et une synchronisation est établie entre l'interface locale et celle déportée.

L'interface HTML présente un intérêt additionnel : elle est paramétrée pour accepter le glisser/déposer. L'utilisateur contrôlant l'interface à distance peut donc modifier le comportement de l'application en glissant son DSP sur l'interface. Le code source est envoyé à l'application locale, où elle remplace l'application en fonctionnement par le processus d'interpolation. Enfin, l'interface distante est remplacée (Figure 8).

3 . Open Sound Control

4 . *QR code* (abbréviation de "Quick Response Code") est un type de code barre à deux dimensions

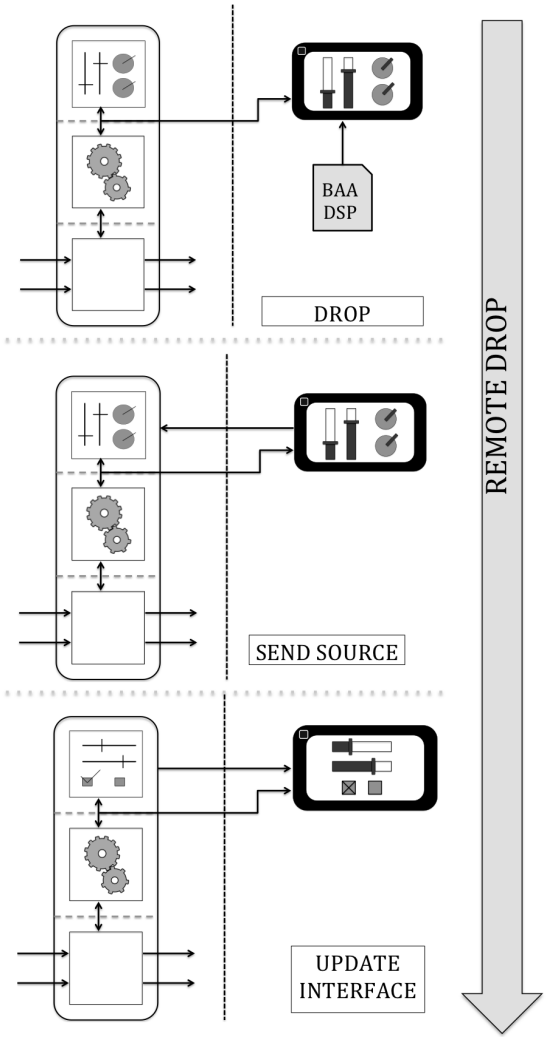

Figure 8. Glisser/Déposer sur une interface distante

Dans le cas où de nombreuses applications coûteuses en calculs sont ouvertes, la charge CPU peut saturer le processeur. Le déplacement du calcul sur une machine distante peut donc permettre d'alléger cette charge (Figure 9).

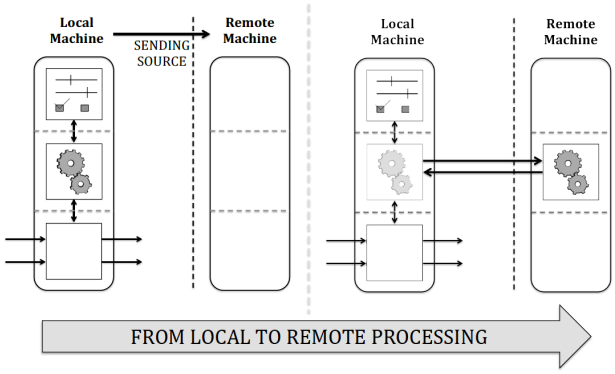

Figure 9. Calcul distribué

Un utilisateur peut vouloir utiliser son application FAUST dans un environnement différent (Max/MSP, SuperCollider, ...). Dans ce but, un lien au service web de compilation, FaustWeb, est intégré dans FaustLive. L'utilisateur doit seulement choisir la plateforme et l'environnement qu'il désire cibler. En retour, il reçoit l'application binaire ou le plugin demandé.

Lorsque FaustLive est quitté, la dernière configuration est sauvegardée et sera restaurée à la prochaine exécution. D'autre part, l'utilisateur peut choisir d'enregistrer l'état de l'application à n'importe quel moment. Il pourra ensuite recharger son "snapshot" en l'important dans l'état courant ou en le rappelant (Figure 10).

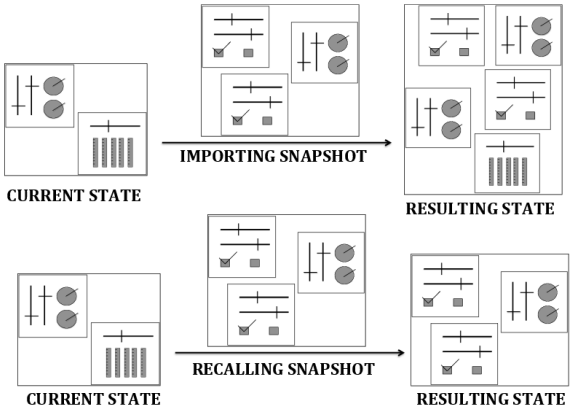

Figure 10. Reloading Snapshot

4. FAUSTLIVE - ASPECTS TECHNIQUES

4.1. Fonctionnalités basiques

Le but principal de FaustLive est de créer un environnement dynamique pour le prototypage d'applications FAUST en embarquant *libfaust*. Les fenêtres QT, une fois paramètrées pour accepter le glisser/déposer, permettent à l'utilisateur de glisser son code FAUST sous forme de fichier, de string ou d'url. Le code est immédiatement donné au compilateur embarqué par le biais de la fonction *createDSPFactory* de l'API libfaust (cf 2.2). L'avantage de la chaîne de compilation résultante (Figure 11) est d'accélérer le processus de compilation, en retournant quasiment instantanément les pointeurs sur les fonctions exécutables. Le nouveau processeur audio remplace alors l'application courante par le biais d'un crossfade, évitant une coupure brutale du son.

Un avantage important de FaustLive est la coexistence de plusieurs applications FAUST, en opposition avec l'architecture QT-JACK de la distribution FAUST "statique" où chaque programme doit être compilé séparément pour produire une application propre. L'environnement résultant est présenté Figure 12.

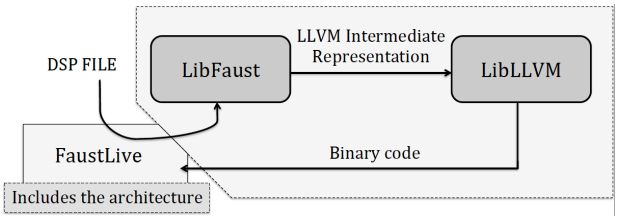

Figure 11. Compilation chain in FaustLive

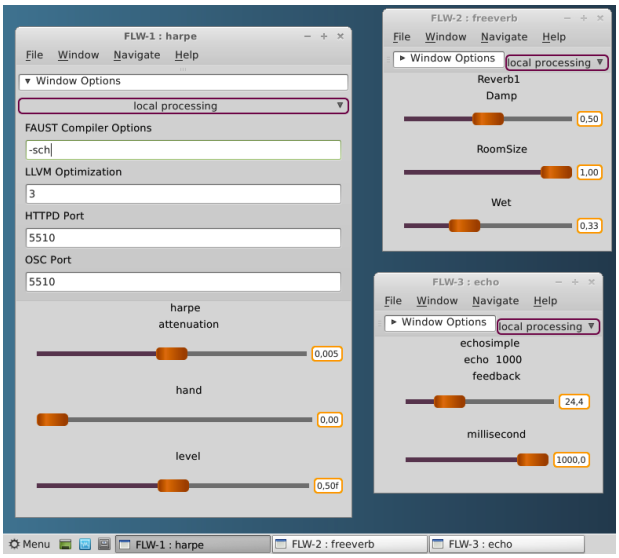

Figure 12. FaustLive's environment

4.2. Audio Drivers

FaustLive a intégré JACK, CoreAudio, NetJack et PortAudio. Selon la plateforme, les librairies sont ou non compilées dans FaustLive [5] . Grâce à cette diversité de drivers, il est possible de basculer de l'un à l'autre ou de modifier les paramètres (taille de buffer, fréquence d'échantillonnage, ...) durant l'exécution de FaustLive. Tous les clients audio sont d'abord arrêtés, puis les applications sont transférées dans le nouveau domaine pour être redémarrées.

4.2.1. JACK

JACK est un système qui gère en temps réel de l'audio/midi basse-latences. Il fonctionne sous GNU/Linux, Solaris, FreeBSD, OSX et Windows. Il peut connecter plusieurs applications à un dispositif audio. De plus, il permet de partager des flux audio entre applications.

Une contrainte intéressante amenée par l'utilisation de JACK a donc été la question des connexions audio. Lorsque des connexions ont été établies, l'objectif est de les conserver, quels que soient les changements d'application FAUST dans une fenêtre. Lorsqu'une nouvelle application entre dans la fenêtre, le graphe des connexions JACK est sauvé sous forme de fichier. Ce dernier est parsé pour remplacer le nom du client sortant par le nom du nouveau processeur audio. Le nouveau client JACK peut alors être connecté à la manière de son prédecesseur. Si la nouvelle application a plus de ports que la précédente, l'utilisateur devra faire les connexions lui-même.

5 . JACK, CoreAudio et NetJack sont utilisés sur OSX, JACK et NetJack sur Linux, PortAudio, JACK et NetJack sur Windows.

4.2.2. NetJack

NetJack est un système de transport audio en temps réel à travers un réseau IP. Il est intégré à JACK. NetJack synchronise ses clients sur une unique carte son pour qu'il n'y ait pas de rééchantillonnage ou de click. Le "master" impose sa fréquence d'échantillonnage, ainsi que sa taille de buffer, en liaison avec le dispositif audio qu'il utilise. Grâce à NetJack, plusieurs utilisateurs peuvent envoyer leurs sorties audio sur une unique carte son. Une utilisation typique de ce système est une classe où seule la machine du professeur est connectée au système audio et les élèves envoient leurs sorties audio à travers NetJack.

4.2.3. CoreAudio et PortAudio

Sous OSX, l'application FaustLive en version JACK n'est pas autonome, elle nécessite que l'utilisateur installe JACK par ailleurs. Pour faciliter l'installation de FaustLive aux utilisateurs débutants, une version *CoreAudio* [6] , ainsi qu'une version *PortAudio* [7] ont été créées. Incluses par défaut dans les systèmes ou facilement distribuées sous la forme de DLL, ces architectures n'ajoutent pas de contrainte d'installation supplémentaire pour l'utilisateur.

4.3. Interfaces de contrôle

Pour pouvoir contrôler l'interface utilisateur à distance, un port UDP doit être ouvert pour le protocole OSC et un port TCP pour la connexion HTTP. Les deux protocoles utilisent par défaut le port 5510 et sont configurables dans la barre d'options de la fenêtre. Lorsque le port est utilisé, le système utilise automatiquement le prochain port disponible.

4.3.1. Interface HTML

Lorsqu'une interface HTML est construite, un serveur HTTP est démarré et s'occupe de délivrer la page HTML (Figure 13). Ce serveur est géré par la librairie *libmicrohttpd*.

Pour simplifier l'accès à cette interface, un QrCode est construit à partir de l'adresse HTTP, grâce à la librairie *libqrencode*. La plupart des smartphones et des équipements portables sont équipés de décodeurs de QrCode. En scannant le code, un navigateur se connecte à la page de l'interface.

6 . *CoreAudio* est l'infrastructure audio de iOS et OS X. Elle offre un framework pour manipuler de l'audio dans les applications.

7 . *PortAudio* est une librairie audio écrite en C/C++. Elle est libre et cross-platformes. Son intention est de promouvoir le portage d'application audio entre différentes plateformes.

4.3.2. *Glisser/déposer à distance*

Comme le reste de la distribution FAUST, l'interface HTML a un comportement "statique". Pour dynamiser son comportement à l'image des interfaces locales de FaustLive, une zone de glisser/déposer a été ajoutée à l'interface HTML. Ce service HTTP est indépendant et spécifique à FaustLive. Le serveur, démarré par FaustLive, est capable de créer une page HTML qui encapsule les interfaces de contrôle. Les interactions de cette page sont gérées avec Javascript. Le service résultant (interface de contrôle + glisser/déposer de DSP) a l'adresse suivante : http ://adresseIP :portServiceGlisserDéposer/ portInterfaceControle (Figure 13).

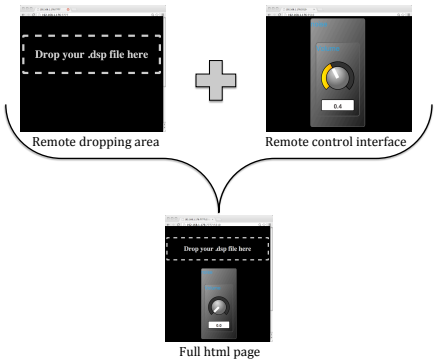

Figure 13. Interface HTML avec le service de glisser/déposer

Le port du service de glisser/déposer est configuré dans les préférences générales de FaustLive et il est commun à toutes les applications FAUST. Les interfaces de contrôle ont, elles, chacune un port différent, modifiable dans les options de la fenêtre.

La réaction à un glisser/déposer sur l'interface distante suit le même modèle que sur l'interface locale. Le code DSP est d'abord envoyé à FaustLive par une requête HTTP, POST. Le code est alors compilé et l'application est remplacée après le mécanisme de crossfade. Enfin, l'interface distante est remplacée.

4.4. Calcul distribué

Pour élagir ses bénéfices, FaustLive permet de faire du calcul à distance. La compilation ainsi que le calcul audio sont redirigés sur la machine distante. La charge du processeur local peut ainsi être réduite.

Sur la machine distante, une application démarre un serveur HTTP, offrant le service de calcul distribué. Ce serveur attend des requêtes de compilation/calcul.

Du côté du client (FaustLive), une API "proxy" rend transparent la création d'un remote-dsp plutôt qu'un llvm-dsp local (c.f 2.2). Cette API, *libfaustremote*, prend soin d'établir la connexion avec le serveur.

La première étape (la compilation) est exécutée par la fonction *createRemoteDSPFactory*. Le code est envoyé au serveur, qui le compile et crée la "réelle" llvm-dsp-factory. La remote-dsp-factory est, pour l'utilisateur, une image de la "véritable" factory. Avant d'envoyer le code FAUST, une étape de compilation FAUST-FAUST résout les dépendances du code à ses librairies. De cette manière, le code envoyé est indépendant et pourra être correctement compilé du côté serveur. La remote-dsp-factory peut ensuite être instanciée pour créer des remote-dsp instances, qui peuvent être lancées dans l'architecture audio et visuelle choisie, ici FaustLive (Figure 14).

Pour être capable de créer l'interface localement, le serveur renvoie une interface encodée en JSON [8] . De cette manière, la fonction *buildUserInterface* peut être recréée pour donner l'impression qu'un remote-dsp fonctionne de la même manière qu'un dsp local.

De plus, les calculs audio sont redirigés à travers une connexion NetJack. Les données audio sont envoyées à la machine distante qui traite les données avant de renvoyer ses résultats. Outre le flux audio standard, un port midi est utilisé pour transférer les valeurs des contrôleurs. L'intérêt de cette solution est de transmettre les buffers audio et les contrôleurs, synchronisés dans la même connexion. D'autre part, les échantillons audio peuvent être encodés en employant les différents types de données audio : float, integer et audio compressé [9] .

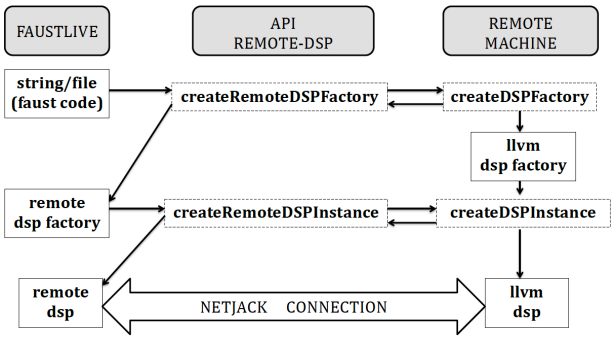

Figure 14. Compilation et Calcul distribués

libfaustremote utilise la librairie *libcurl* pour envoyer des requêtes HTTP au serveur de calcul, lui même manié avec *libmicrohttpd*.

Sur chaque fenêtre de FaustLive, l'accès au service de calcul à distance est simple. Le protocole ZeroConf est utilisé pour scanner les machines distantes présentant ce service. Une liste est construite dynamiquement à partir des machines disponibles. En naviguant dans cette liste, l'utilisateur peut ensuite faire basculer son application FAUST d'une machine à l'autre très facilement.

8 . *JavaScript Object Notation* est un format de données textuelles, dérivé de la notation des objets du langage JavaScript.

9 . à l'aide du codec OPUS : http ://www.opus-codec.org

4.5. FaustWeb

Pour simplifier l'accès à la compilation FAUST, un service de compilation web a été conçu. Il reçoit du code FAUST et renvoie une application ou un plugin pour l'architecture et la plateforme choisie. Dès lors, l'installation de la distribution complète FAUST n'est plus nécessaire. N'importe qui peut décrire un programme FAUST, l'envoyer au serveur puis utiliser le plugin qu'il aura reçu.

Ce service est accessible depuis un navigateur mais nécessite plusieurs requêtes. À travers FaustLive, l'export des applications est facilité. Un menu est construit dynamiquement avec les plateformes et les architectures disponibles, encodées sous forme de JSON . Tandis que l'utilisateur fait le choix d'exporter son application, le code FAUST correspondant est envoyé au serveur. Ce dernier vérifie la syntaxe FAUST. Une clé SHA1 unique correspondant à ce DSP est générée pour faire ensuite les requêtes de différents binaires. La seconde étape est la compilation, utilisant la chaîne de compilation "statique" réalisée coté serveur, pour renvoyer enfin l'application choisie par l'utilisateur (Figure 15). Pour réaliser les requêtes au service FaustWeb, FaustLive utilise la librairie NetWork du framework QT.

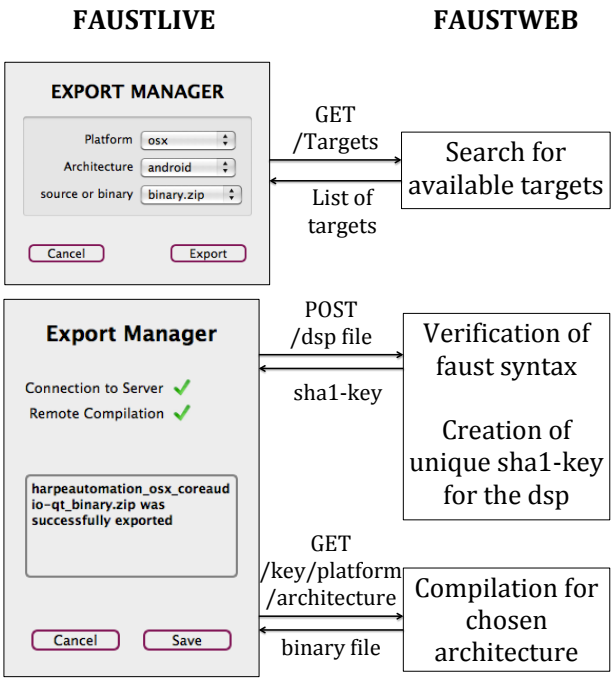

Figure 15. Les étapes de la chaîne de compilation à travers FaustLive

4.6. Concept de session

Un concept de session est conservé pour préserver l'état de l'application (valeur des paramètres, position sur l'écran, connexions audio, options de compilation, ...). Quand l'application est quittée, la session est enregistrée par défaut pour être restaurée à la prochaine exécution. L'utilisateur peut aussi sauvegarder l'état de l'application complet dans un "snapshot", enregistré dans un dossier, xxx.tar.

Toutes les ressources locales nécessaires (fichiers DSP, ...) sont copiées dans le dossier, xxx.tar, rendant indépendants les "snapshots" de FaustLive. Les chemins d'accès vers les ressources originelles sont utilisés prioritairement. Cependant, si un fichier source est supprimé ou si un "snapshot" est déplacé sur une autre machine, les copies doivent être utilisées.

Pour réduire le temps de compilation d'un DSP, la sortie du compilateur FAUST (le code LLVM IR optimisé) est enregistrée. Lorsqu'une application est rechargée, la phase de compilation FAUST du code DSP au code LLVM IR est alors économisée, ainsi que la phase d'optimisation du code LLVM IR. Pour des programmes couteux, le gain est notable. On peut passer de quelques secondes de compilation à une exécution instantanée.

4.7. Limites

Certains aspects techniques limitent l'ouverture d'un nombre indéfini de fenêtres FaustLive.

D'abord, pour pouvoir synchroniser la recompilation avec l'enregistrement du fichier source, un moniteur est placé sur le fichier, ce qui consomme des ressources systèmes. Selon la plateforme utilisée, cette limite est plus ou moins grande. De plus, il existe une limitation du nombre de liaisons OSC parallèles. Enfin, le nombre de clients JACK est limité.

Pour toutes ces raisons, un maximum de soixante applications FAUST peuvent être ouvertes parallèlement.

5. CONCLUSION

FaustLive réunit le confort d'un langage interprété autonome et l'efficacité d'un langage compilé. FaustLive offre actuellement le cycle de développement le plus rapide pour des applications FAUST, fournissant un outil modulaire pour le prototypage d'applications FAUST. Elle intègre aussi des fonctionnalités avancées de calcul distribué et de contrôle à distance pour des applications audio temps-réel. De plus, FaustLive met à disposition, à travers sa fonction d'export, un front-end confortable pour FaustWeb, le service de compilation web.
Ce projet est open-source et disponible sur SourceForge [3]. Il fonctionne sur Linux, OS X et Windows.

6. REMERCIEMENTS

Ces développements ont été menés dans le cadre des projets INEDIT [ANR-12-CORD-0009] et FEVER [ANR-13-BS02-0008], soutenus par l'Agence Nationale pour la Recherche.

7. REFERENCES

[1] A. Graef. Functional signal processing with pure and faust using the llvm toolkit. 2011.

[2] faust2 repository. http ://source-forge.net/p/faudiostream /code/ci/faust2/tree/.

[3] faustlive repository. http ://source-forge.net/p/faudiostream /faustlive/ci/master/tree.

[4] S. Letz, Y. Orlarey, and D Fober. Dynamic compilation of parallel audio applications. 2013.

[5] Y. Orlarey, S. Letz, and D. Fober. Faust : an efficient functional approach to dsp programming. 2009.