Compte-rendu à t=2.0y

Jean-Michaël Celerier

November 13, 2016

Contents

Chapter 1

Introduction

Ce document survole les travaux qui ont été réalisés jusqu'à présent lors de la thèse, étudie les pistes qui sont ouvertes et les possibilités pour la dernière année.

1.1 Mise en relation avec le sujet

Calques audio interactifs : théorie, mise en oeuvre et usages.

1.2 Articulation et analyse générale

Chapter 2

Réalisations

2.1 Étude de cas

2.1.1 Sonopluie

2.1.2 Tableaux sonores

Modèle pyramidal : - ambiance globale - zones - éléments individuels

Bosch

Vernet

2.1.3 Jardin connecté

2.2 Développements théoriques, publications

2.2.1 États de l'art

2.2.2 Modèle théorique

TENOR2015: OSSIA

IUI2015 (refusé)

JNMR: Vérification

JIM2016: Interface

ICMC2016: Programmation structurée

Démo : inspiré Klavierstucke XI.

i-score qui communique par WebSockets avec un serveur Node.JS

Des gens se connectent par téléphone au serveur et peuvent influer sur l'exécution du scénario : prochaine partie à jouer, et vitesse moyenne de lecture.

2.2.3 Espace

Compte-rendu espace

JIM2016: Démo

JIM2016: Espace

-> Conclusion : CAS peu adéquat, dur d'avoir de bonnes performances à un tick rate quelconque. Alternatives : se restreindre aux cas linéaires ? GPU ? Mais latence.

2.2.4 Audio

SMC2016: i-score et LibAudioStream

Démo : un peu toutes les fonctionnalités

2.2.5 Répartition

Rapport de stage

2.3 Conférences, présentations, workshops

2.3.1 Cycles SCRIME 2015

2.3.2 Forum IRCAM 2015

2.3.3 FOSDEM 2016

2.3.4 Cycles SCRIME 2016

2.3.5 GDR ESARS

2.3.6 DESINC2016

2.3.7 Workshop improvisation

2.4 Développements logiciels

2.4.1 Génie logiciel et généralités

Performances

Question des performances ? Comment mettre en valeur ? Un accent très fort est mis dessus.

Tests

Idem pour tests. Couverture de code : 70 % pour libossia, 50 % pour i-score, 0 % pour extensions i-score

2.4.2 i-score

Architecture

Problèmes actuels

Portabilité

Après avoir enlevé Jamoma, exécution sur Android et iOS.

2.4.3 Extensions à i-score

Édition répartie

Audio

Automation 3D

PureData

Espace

Image

Vidéo

Controle à distance

Analyse statique

Segments

Extension "Preset"

2.4.4 libossia

Architecture

Problèmes actuels

- temps de compilation

Portages

C

Csharp et Unity

Qt

Java

Javascript

2.4.5 OSCQuery

2.4.6 coppa

2.4.7 Études et développements mineurs

External RealSense

Outils pour graphe de calcul

DisPATCH

RaftLib

Contribution à d'autres projets open-source

LibAudioStream

FAUST

Jamoma

Contributions mineures

- Placeholder/Nodeeditor
- verdigris
- fmt
- Qt-color-widgets
- jni.hpp
- quazip
- QRecentFilesMenu
- ModernMIDI
- libsamplerate
- ofxMSAPhysics
- Cotire

2.5 Projets liés

2.5.1 Audio

Stage Magali Chauvat

Objectifs - Il faut encore faire la brique d'intégration dans i-score

2.5.2 Robots

Stage Nicolas 2015

Stage Kinda Al Chahid 2015

Stage Paul Breton 2016

Stage Maëva 2016

Projet TM - Robot 2015 - 2016

Objectifs

Groupe TM

Groupe Robots

Projet TM - Robot 2016 - 2017

Objectifs

Groupe TM

Groupe Robots

PFA2016 - 2017

Objectifs

2.6 Cours et TDs donnés

2.6.1 TIM

2.6.2 TAP

Chapter 3

Objectifs à venir

3.1 Système réparti

3.1.1 Exécution répartie

3.1.2 Répartition des protocoles

3.2 Audio

3.2.1 Article dans CMJ ?

Pour que ce soit convainquant : offrir en plus la possibilité de réutiliser les flux passés. Et bien tout modéliser.

- Utiliser la modélisation de la LibAudioStream (Flux) ? - Utiliser la modélisation de OSSIA (Petri) ?

3.2.2 Signatures temporelles

3.2.3 Support audio étendu

- Gestion des pistes (entrées / sorties virtuelles comme dans un vrai séquenceur) - Intégration travail TrackListModel de Magali - Spatialisation / autre chose que stéréo -> pistes de sortie 1, 2, 3, N canaux

3.2.4 Question des queues de reverb

Possibilité : - Au début, lire tous les streams à l'infini dès qu'ils commencent. - Question des boucles ? que doit-il se passer ? Dans un DAW normal on boucle la lecture de contenu (fichiers audio, notes midi), mais les effets sont hors de la boucle.

Ici les deux choses sont possibles, c'est à l'auteur de choisir (en mettant les effets dans la boucle ou hors de la boucle).

- Compensation de délai : certains effets (compresseur, limiteur, certaines réverbs) impliquent un délai dans le traitement du signal. La plpuart des DAW compensent ce délai au prix d'une latence d'entrée plus élevée : les autres pistes sont décalées de manière à tenir compte du délai des plug-ins. Ableton Live offre le choix : compenser le délai ou avoir une latence d'entrée plus faible. Si le

délai n'est pas prix en compte, deux pistes qui étaient synchronisées à la base ne le seront plus après application de l'effet, ce qui peut poser problème pour des sections rythmiques par exemple.

Certains types d'effets préviennent à l'avance du décalage imposé, tandis que d'autres non. Par exemple: - Les effets VST supportent un délai statique (fixé au chargement du plug-in) - Les effets Reaper, Pro Tools supportent un délai dynamique. - Ce doit être fait manuellement par l'auteur pour FaUST.

VST / VSTi

LV2

Format de plug-ins qui permet l'analyse en temps réel de données.

3.3 Embedding de i-score

3.3.1 DLL dans d'autres moteurs d'exécution

Max

VST

Unity

3.3.2 Scénarios compilés

3.3.3 Web

3.3.4 IncludeOS pour devices?

3.4 Unification temps - espace

- Lister les cas possibles : comprendre ce que "appliquer les structures de i-score dans l'espace" veut dire. Ex : - si on considère qu'une contrainte temporelle est toujours sur une seule dimension, de temps ? - si on considère qu'une contrainte temporelle est toujours sur une seule dimension, d'espace ? - si on considère qu'une contrainte temporelle est sur plusieurs dimensions d'espace ? - si on considère qu'une contrainte temporelle est sur toutes les dimensions à la fois (on fait "progresser" l'espace temps et on arrive dans des espaces différents selon les outcome du scénario i-score). - si on considère non pas l'avancement dans un scénario comme quelque chose de global, mais quelque chose de lié à un acteur. => on a dès lors besoin de multiples curseurs de temps.

Alternative "intégration dans unity" : on applique à un objet existant dans Unity, un scénario i-score. Ce scénario devrait pouvoir interagir avec le reste des paramètres ? On voit juste un aspect d'un scénario global ?

3.5 Modèle par graphe de noeuds pour calcul par tranches

- On veut à chaque instant (tick) avoir un graphe de calcul différent, en fonction des processus qui sont exécutés. -> faire petit schéma.

Il faut : * Changer le moteur d'exécution afin qu'il fonctionne réellement sous la forme d'un graphe -> Changer les processsus / le type "ossia::state" pour que les noeuds effectuent des calculs (lazyness). -> Changer les processus pour qu'ils spécifient les adresses qu'ils utilisent en entrée - sortie. Par exemple pour le processus Javascript cela permettrait d'activer le listening sur les adresses qu'il utilise en entrée, plutôt que de faire un "pull" dessus qui peut être plus long.

3.6 Édition à l'exécution

Actuellement, les structures de données utilisée à l'exécution sont recréées à chaque fois que le bouton "exécuter" est pressé. C'est du au fait que les structures de données de l'API OSSIA ne sont pas thread-safe : il est actuellement dangereux de changer par exemple la durée d'une contrainte pendant qu'elle s'exécute car ces deux choses se font dans des threads distincts.

La première possibilité est de rajouter des mutex aux endroits impactés par l'édition (c'est à dire à peu près partout).

La seconde (ayant ma préférence) est d'utiliser une file de commandes pour l'édition : des commandes d'édition sont envoyées et mise dans une file sans verouillage (lock-free queue).

Le moteur d'exécution peut appliquer les commandes d'édition dans le cadre de sa boucle d'exécution, une fois que le tick d'exécution courant est terminé. Cela permet de garder les performances du code sans mutex, tout en prévenant les crashs.

Par la suite, il est nécessaire de changer la manière dont l'API est appelée depuis i-score pour qu'elle garde les structures en mémoire plutôt que de les recréer à chaque exécution, et qu'elle réagisse aux changements dans l'interface graphique.

Une perspective intéressante serait d'offrir par la suite une API d'édition par le réseau (OSC) ou via Javascript, pour pouvoir écrire des partitions autogénératives.

3.7 Polyphonie

- Faire une liste des évènements communs à tous les types de polyphonie

3.7.1 Polyphonie dans les scénarios

3.7.2 Polyphonie dans l'espace

- interaction entre deux agents / utilisateurs

3.7.3 Polyphonie dans le son

- plusieurs sons - plusieurs auditeurs

3.7.4 Spatialisation du son

- plusieurs enceintes En général deux méthodes : - Spatialisation par piste - Spatialisation par objet - Intégration temps / espace

3.8 Scripting de l'interface en Javascript

3.9 Interaction

- Extraction des informations. Informations de quoi ? Les dataspace le font déjà dans un sens.

3.10 Langage

Modéliser les développements réalisés via un langage.

Approche escomptée : langage réactif ou les variables sont des processus.

Comment le langage peut-il rendre compte de l'aspect "plug-in" du logiciel ?

Très dur de faire un langage extensible au niveau de la syntaxe / sémantique ...

Approche DSL dans un langage existant ?

Le langage doit-il produire un scénario i-score ou bien directement s'exécuter ?

Compiler en code C++ qui correspond ?

3.11 Problématiques d'UI / de développement

- Afficher un processus sous plusieurs aspects (e.g. pour une courbe 3D, les automations x, y, z ainsi que l'allure générale que ça donne). - Alias dans arbre (par exemple trois noeuds addrR, addrG, addrB => une seule addresse addr) - Problématique de l'édition réciproque.

3.11.1 Dataspace

- Gérer les dataspace qui wrap (0 -> 360)

3.11.2 Plug-in marketplace

3.11.3 Bibliothèque

3.12 Lien entre différents aspects

Comment spatialiser un son efficacement en combinant les processus espace, etc.

Comment tout exécuter dans navigateur : il faut du webaudio pour la libaudiostream.

3.12.1 Objets sonores interactifs

-Objets sonores interactifs "self-contained" -> "Chien qui aboie" et qui se souvient qu'on l'a embêté.

Chapter 4

Conclusion

4.1 Emploi du temps

4.2 Discussion