Rapport détaillé de conception

Synthesizer

Vincente NATTA
Dorian LEROY
Kheireddine BOURAHLI
Guillaume DARVES
Yves MOCQUARD
Caroline LAVAURE
Robin HACAULT
Anthony LHOMME

SOMMAIRE

Le projet Synthesizer

Le projet Synthetiser a pour objectif de concevoir et développer un logiciel simulant un synthétiseur analogique à synthèse soustractive. Ce logiciel est capable de synthétiser des flux audio grâce à des modules de traitement audio.

Principe général

Une bibliothèque de modules de traitement audio est mise à disposition de l'utilisateur.

Chaque module peut fournir des ports d'entrée, des ports de sortie, et des paramètres de réglage.

L'utilisateur peut assembler différents modules grâce à des connexions entre les ports d'entrées et les ports de sortie afin de créer son propre synthétiseur.

Le câble de connexion transmet le signal de la sortie sur laquelle il est branché du composant 1 à l'entrée sur laquelle il branché du composant 2.

Le logiciel Synthesizer

Fonctionnalités proposées

Les principales fonctionnalités de l'application sont :

- Ajout / déplacement / suppression de modules de traitement audio
- Connexion des modules à l'aide de câbles
- Choix de la couleur des câbles
- Sauvegarde et chargement de montages

Technologies employées

Synthesizer est développé en Java et repose sur la bibliothèque Jsyn permettant le traitement de signaux audio.

L'interface graphique est produite en JavaFX 8. Chaque interface des modules est dans un fichier fxml. Enfin, le style est extériorisé dans des fichiers CSS, permettant un changement d'apparence à la demande.

Composants disponibles

Synthesizer est composé de plusieurs composants logiciels interconnectés, c'est-à-dire :

- Du plan de travail : composant principal contenant les autres composants
- D'une bibliothèque de composants de traitement audio. Ce sont des modules qui reçoivent ou transmettent un signal audio grâce à leur port d'entrée et de sortie. Chaque module possède au moins un potentiomètre permettant de faire des réglages.
- De câbles permettant la connexion entre les différents composants.

Les modules disponibles sont :

- **OUT** : module transmettant le son à la carte son de l'ordinateur
- **IN** : module d'entrée de ligne (par défaut : le microphone)
- **NOISE** : module produisant un bruit blanc
- **KEY** : module d'entrée clavier
- **VCO** (Voltage Controlled Oscillator) : module générant des signaux de différentes formes. Les formes disponibles sont triangle, carré et en dent de scie
- **VCA** (Voltage Control Amplifier) : module permettant de moduler l'amplitude du signal d'entrée
- **VCF-LP** : module transformant un signal en atténuant ou amplifiant certaines fréquences. Ce module et de type "passe-bas", c'est-à-dire qu'il laisse passer les fréquences basses mais atténue les fréquences hautes (24 dB/octave)
- **VCF-HP** : module transformant un signal en atténuant ou amplifiant certaines fréquences. Ce module et de type "passe-haut", c'est-à-dire qu'il laisse passer les fréquences hautes mais atténue les fréquences basses (12 dB/octave)
- **REP :** module répliquant le signal d'entrée sur les ports de sortie
- **MIXER 2** : module permettant d'additionner deux signaux
- **MIXER 4** : module permettant d'additionner quatre signaux
- **EG** (Enveloppe Generator) : générateur d'enveloppe
- **SEQ** : module permettant de générer une suite de valeur de voltage
- **SCOPE :** module oscilloscope permettant de visualiser le signal
- **ECHO** : module permettant de générer un écho au signal d'entrée
- **RECORDER** : module d'enregistrement du son dans un fichier
- **PLAYER** : module de lecture du son depuis un fichier
- **SAVE** : module de sauvegarde et chargement d'une configuration

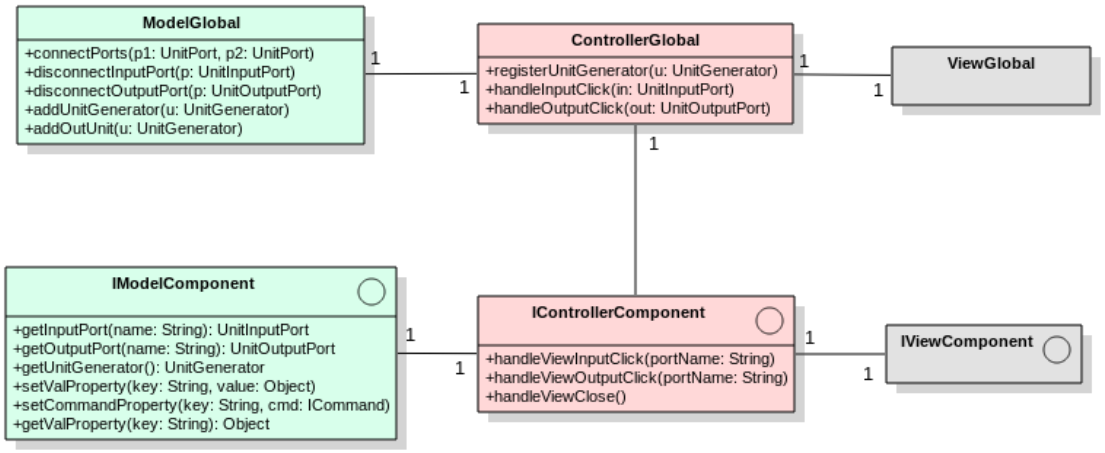
Architecture de conception

Pattern MVC

Lors de la conception de notre application, nous avons opté pour une architecture où chaque composant implémente son propre pattern MVC et où la connexion des composants entre-eux est gérée par le contrôleur d'un pattern MVC global.

Le diagramme de classes

Il représente l'architecture globale de l'application.



Pattern MVC global

Le pattern MVC global regroupe les classes ViewGlobal, ControllerGlobal et ModelGlobal. La classe ViewGlobal est la vue principale de l'application et communique avec le contrôleur (ControllerGlobal), chargé d'informer le modèle (ModelGlobal) des connexions à effectuer ou à supprimer entre les différents composants, en réponse aux actions de l'utilisateur.

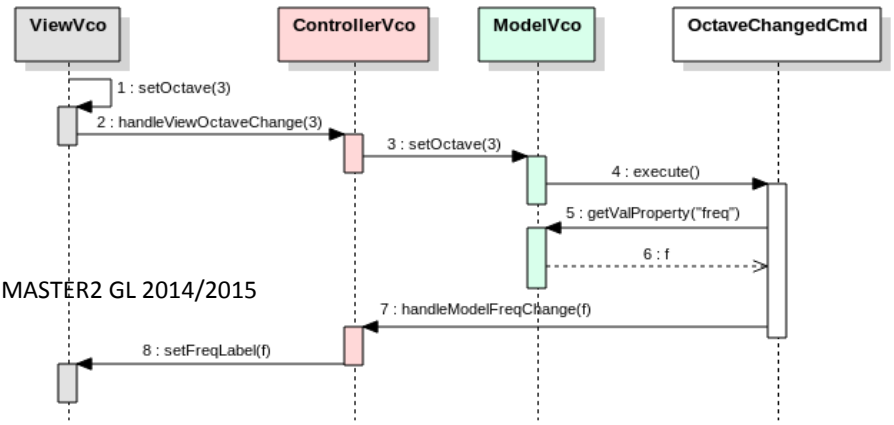
Lorsque l'utilisateur fait part à l'application de son désir d'ajouter un composant, la vue globale en notifie le contrôleur global. Ce contrôleur en informe le modèle global, qui se charge de l'ajout du modèle du composant à sa liste de modèles de composants. Lorsque l'ajout est confirmé par le modèle, celui-ci notifie le contrôleur, qui demande à la vue d'afficher le composant.

La fonction principale de la classe ModelGlobal est de stocker les modèles des composants actuellement instanciés (qui implémentent l'interface IModelComponent). C'est la classe ControllerGlobal qui le notifie des composants à ajouter ou à retirer, ainsi que des connexions à effectuer et à supprimer.

Pattern MVC composant

Comme précisé plus haut, chaque composant implémente son propre pattern MVC. Le modèle d'un composant encapsule le module JSyn qui lui est associé. Il sert également à stocker les paramètres de configuration spécifiés par l'utilisateur en interagissant avec la vue du composant.

Chaque modèle d'un module étend de plus la classe abstraite ModelComponent, qui implémente des méthodes d'enregistrement de Properties observables par le contrôleur selon le patron de conception Observer/Command. Ces commandes, instanciées par le contrôleur et stockées dans le modèle, sont utiles pour notifier à la vue des changements effectués depuis le modèle. Les Properties nous permettent d'obtenir un code générique pour tous les modules.



Le diagramme ci-contre représente l'observation de la fréquence du VCO par la vue, après un changement de l'octave venant de l'utilisateur. On

suppose que le contrôleur a auparavant créé une commande et l'a associée dans le modèle à la Property "octave".

Afin de faciliter le plus possible le développement de nouveaux modules, vous avons choisi de factoriser le plus de méthodes possibles dans les classes abstraites ViewComponent, ControllerComponent et ModelComponent. Ces classes masquent des détails d'implémentation que les développeurs de modules n'ont pas besoin de garder en tête, comme entre-autres la réaction à l'observation des Properties.

Bibliothèque JSyn

Nous avons choisi d'utiliser la librairie gratuite pour des projets open source JSyn, afin de nous faciliter le développement côté métier.

Cette librairie possède toutes les fonctions de bases pour développer un synthétiseur audio numérique.

Nous avons pu fournir une bonne cohabitation entre JSyn et notre architecture en mettant en oeuvre chaque composant de manière à ce qu'il puisse remonter au modèle global le composant JSyn qui lui est associé, afin de permettre le câblage au sein du modèle global.

Bilan

Cette architecture nous permet de reprendre les avantages du patron de conception MVC. Nous pensons avoir rempli l'objectif de séparer clairement les préoccupations et rendre le développement indépendant. De plus, la séparation nette entre vue globale et composants renforce la possibilité de développer des modules en parallèle et de les tester sans avoir besoin d'une vue globale déjà mise en oeuvre. Enfin, tout cela nous a permis un partage efficace des tâches entre les différents membres du groupe de projet.