Grapheur

Généralisation et Rédaction de son document de conception

Kylian Gehier
3ème Année – Informatique & Réseaux
Ecole Nationale Supérieure d'Ingénieurs Sud Alsace
Mulhouse, France
kylian.gehier@gmail.com

Jean-Marc Perronne (Encadrant)
Enseignant Chercheur
Ecole Nationale Supérieure d'Ingénieurs Sud Alsace
Mulhouse, France
jean-marc.perronne@uha.fr

Résumé—Grapheur est une API Java permettant la construction et l'affichage de différents types de graphes (Charts en anglais) et d'y insérer des données prevenant de différentes sources (csv, fichier audios, ..)

Mots clés—chart, patron de conception, mvc, compossit, factory

I. Introduction

Ce projet a vu le jour en 2012. A l'époque, il avait été demandé à Jean-Marc Perronne de réaliser cette API pour une entreprise du nom d'Infral pour de la télémedecine.

Pour des raisons qui me sont inconnues, le projet a été abandonné et mis de côté jusqu'à cette année. Malgré son abandon, le projet avait néanmoins été bien avancé, il était capable de construire et d'afficher deux types de graphes :

- Les LinesChart : Graphes linéaires 2D permettnat l'affichage classique de points reliés par des courbes dans un repère orthonormé.

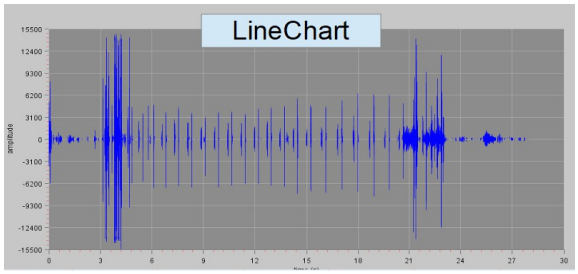


Fig 1 : Exemple de LineChart

- Les MapChart : Graphes linéaires 2D avec effet de profondeur 3D en jouant sur les couleurs dans une repaire orthonormé.

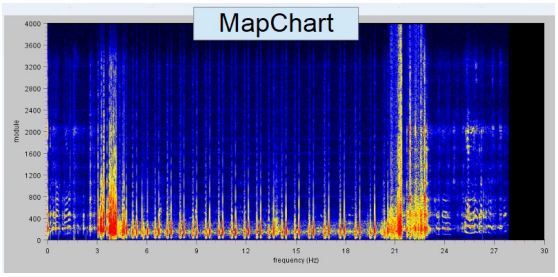


Fig 2 : Exemple de MapChart

Le grapheur permettait alors d'enregistrer des données depuis un micro en acquisition temps réelle ainsi que de simuler une acquisition temps réel depuis un fichier audio.

C'est donc dans le cadre d'un projet de 3ème année que ce Grapheur m'a été attribué avec différents objectifs .

II. Objectifs

A. *Dé-spécialisation du Grapheur*

Le projet Grapheur ayant été initialement créé pour un une entreprise, mon premier objectif consistait à nettoyer toute trace de cette entreprise.

Cela incluant :

- Une réorganisation de l'ensemble des packages du projet
- Une généralisation des fonctionnalités du Grapheur, notemment l'ajout de données entrantes statiques de fichiers CSV (Comma Separated Values)

B. *Réalisation du document de conception du Grapheur*

En parallèle de la spécialisation, il m'a été demandé de rédiger un document de conception du Grapheur destiné aux étudiants qui reprendront ce projets les années suivantes.

Ce document de conception devait comprendre :

- Une présentation de l'architecture du Grapheur, la plus simplifiée et claire possible
- Une présentation des fonctionnalités déjà présentes ainsi que les nouvelles ajoutées dans le cadre de ce projet 3A
- Une partie consacrée aux différents cas d'utilisations (diagrammes de cas d'utilisations , ..)
- Une partie d'ouverture devant servir aux étudiant qui reprendront le projet afin de les guider dans les débuts de leur projet 3A.

III. Architecture

A. *Mes débuts avec le projet Grapheur*

Lorsque le projet m'a été remis par M.Perronne, il comportait plus de 68 classes et n'était pas documenté. Il m'a donc fallut, dans un premier temps, me familiariser avec ce Grapheur en étudiant les cas de tests déjà présents. S'en est alors suivie, une analyse rigoureuse et longue de l'architecture du projet.

Il est important de comprendre que ce Grapheur est basé sur un modèle d'axe, permettant de construire plusieurs types de graphes dans un même projet.

B. *Arhcitecture MVC (Modèle Vue Contrôleur)*

Le Grapheur repose sur une architecture MVC-Swing à laquelle vient s'ajouter un Renderer à qui la Vue délègue la tâche de dessiner le modèle.

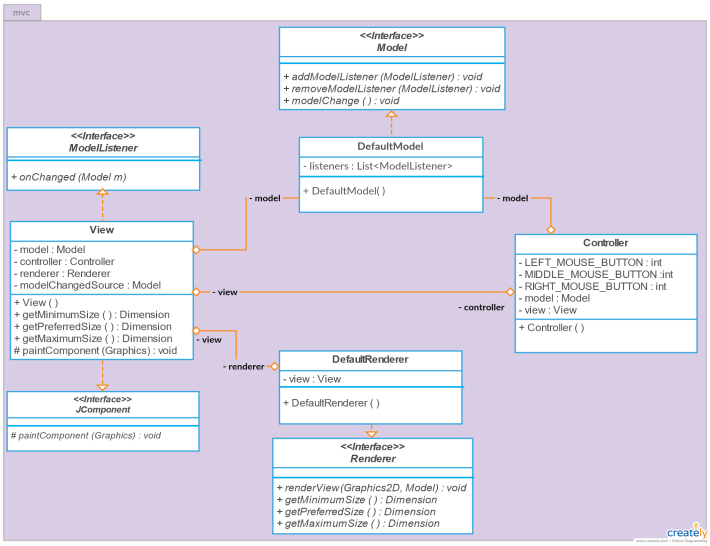


Fig 3 : Diagramme de classe - MVC

Voici un diagramme de classe pris de mon document de conception illustrant le MVC du Grapheur. Ce MVC est donc composé des entités suivantes:

- Modèle : contient les données du Chart (Graphe) sous une forme compréhensible pas la vue et notifie cette dernière lorsqu'il change.
- Controller : réceptionne les différentes actions effectuées depuis la Vue notemment grâce aux différents “Listener” (Ecouteurs) qu'il implémente (Cf . diagramme ci dessous) et modifie le Modèle en conséquence.

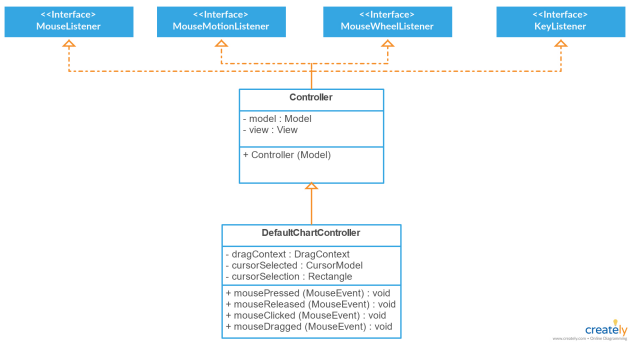
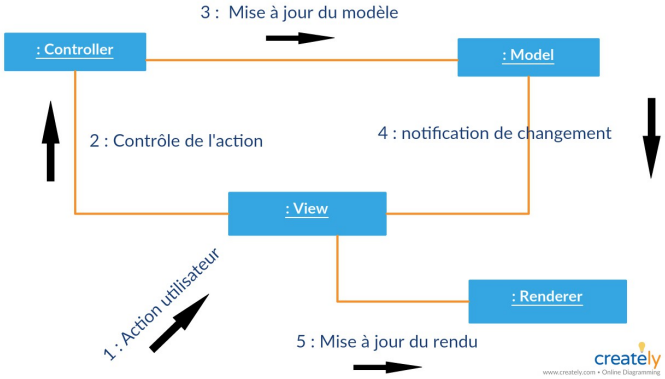


Fig 4 : Diagramme de classe - Contrôleur

- Renderer : chargé par la Vue de dessiner le Modèle
- Vue : c'est l'IHM (Interface Humain Machine) du Chart. Le Modèle y est dessiné par le Renderer

Voici un diagramme résumant ce qui a été expliqué précedemment.

Fig 5: Diagramme de communication -



Fonctionnement du MVC

C. *Modèle du MVC*

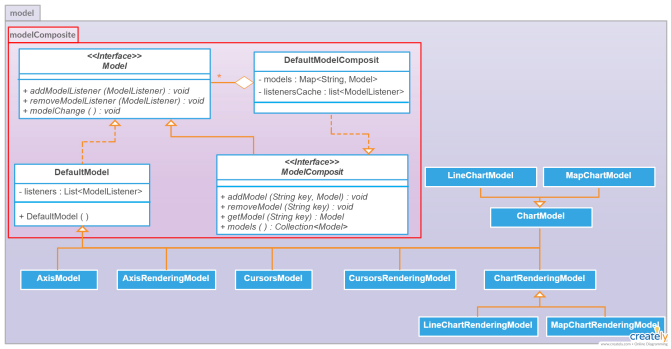


Illustration 6 : Diagramme de classe - Modèle

Comme le montre le diagramme de classe ci-dessus, le Modèle du MVC est décliné en 3 sous-modèles.

- ChartModel : Modèle comprenant les informations relatives au Chart (visibilité de la grille de fond, couleur de fond, données du Chart, …). Ce modèle se décline en LineChartModel et MapChartModel en fonction du Chart que l'on souhaite afficher.
- AxisModel : Modèle d'axe (graduations, nom de l'axe, ..)
- CursorsModel : Modèle comprenant une liste de CursorModel (modèle de curseur) ajoutés par un utilisateur depuis la Vue du Chart.

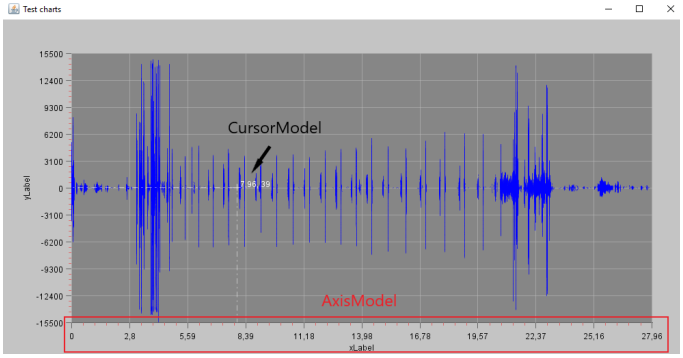


Fig 7 : Shéma illustrant AxisModel et CursorModel

Le précedent diagramme de classe nous montre également que ces sous-modèles sont eux même scindés en un couples Model/RenderingModel :

- Model : Information sans rapport avec le rendu du modèle (Ex : AxisModel → présence de graduation, de sous-graduation, nom de l'axe)
- RenderingModel : Information relative au rendu (Ex : AxisModel → couleur des graduations)

Le tout est englobé dans un "Pattern Composit" pour simplifier l'instanciation des modèles dans la Vue.

D. Vue du MVC

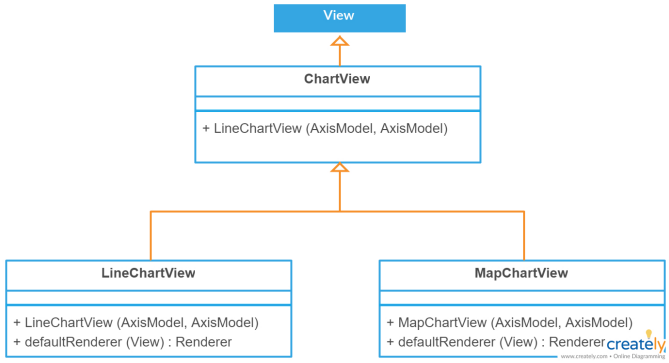


Fig 8 : Diagramme de classe - Vue du MVC

Ce diagramme nous montre que la Vue est déclinée en LineChartView pour les LineChart et MapChartView pour les MapChart.

Lorsqu'un utilisateur souhaite afficher un Chart, il lui suffit d'instancier une de ces vues et de la paramétrer. Cette instanciation engendrera les instanciations des déclinaisons du MVC necessaires à son fonctionnement comme le montre le diagramme ci dessous .

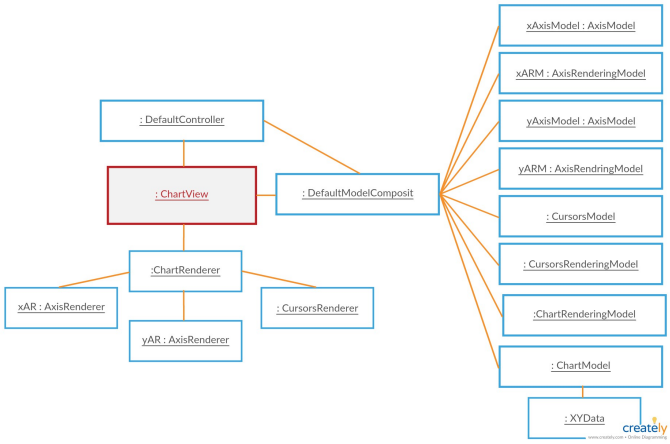


Fig 9 : Digramme d'objet - Composition d'une ChartView

IV. Factory

L'architecture principale du projet vous ayant été présentée, je vais maintenant aborder les améliorations que j'ai apportées au grapheur dont l'ajout d'une Factory.

A. Pourquoi une Factory?

Au vus de la taille des tests pour afficher un Chart (120~150 lignes de code effectives), il m'a semblé evident de centraliser le tout à l'aide d'une Factory dont l'objectif serait de créer des ChartView. L'implémentation de cette Factory à permis de diminuer la taille des tests de 120~150 à une vingtaine de lignes de codes effectives.

B. Comment?

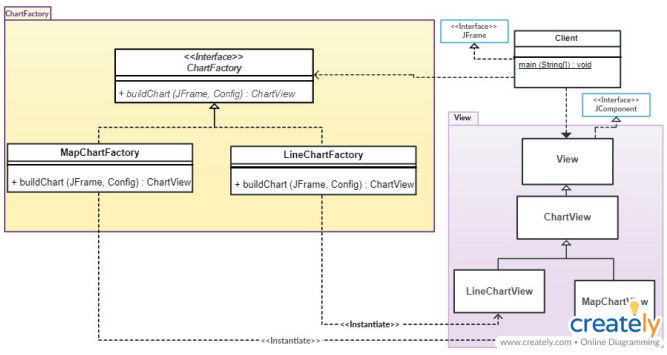


Fig 10 : Diagramme de classe - Factory

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors).*

J'ai donc commencé par créer un système de fichier de configuration comprenant le paramétrage des Chart qui était la partie la plus dense en code des tests ainsi qu'une classe “Config” permettant la lecture de ces fichiers à l'aide d'une méthode de “Parsing”.

Dans un deuxième temps, j'ai réalisé ma Factory comme le montre le précedent diagramme.

Les ChartView héritant de la classe “JComponent”, il m'a suffit de prendre en paramètre de la méthode “buildChart”, la “JFrame” dans laquelle l'utilisateur souhaitait construire son Chart ainsi qu'une instance de la classe “Config” comprenant le fichier de configuration nécessaire à l'affichage du graphe.

V. Données du Chart

A. *Modèle de données du Chart*

Les Chart possèdent leur propre modèle de données établi par M.Perronne.

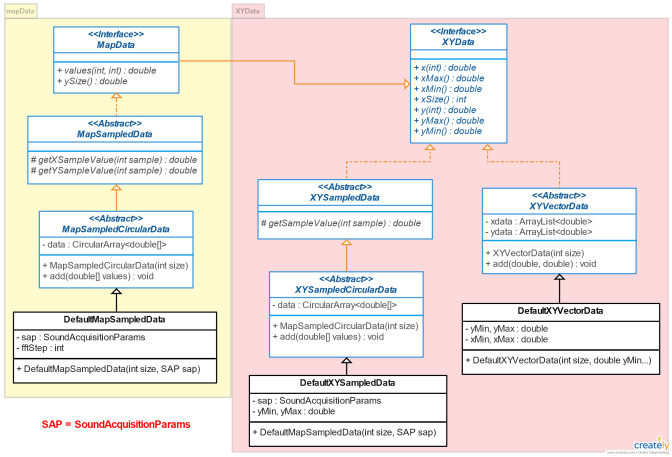


Fig 11 : Diagramme de classe - Modèle de données des Chart

On peut distinguer deux familles de données sur le diagramme ci-dessus : les XYData sont utilisées pour les LineChart tandis que les MapData le sont pour les MapChart.

A l'origine, seules les XYSampledData et les MapSampledData (ainsi que leur classes héritières) ont été conçues par M.Perronne car elles étaient utilisées pour des données entrantes échantillonnées (Acquisition temps réelle ou simulée).

De ce fait il m'a fallut ajouté un modèle de données XYVectorData afin de pouvoir implémenter des modèles de données entrantes autres qu'échantillonnées comme des données prevenant de CSV.

B. *Modèles de données entrantes : Entry*

Afin de ne pas confondre les deux modèles de données, les données du Chart seront maintenant appelées “Data” tandis que les données entrantes (telles que les fichiers Audio, les fichiers CSV, …) seront appelées Entry.

Afin de centraliser les données entrantes, il m'a fallut en créer une abstraction rendant possible l'insertion de ces données dans les différents types de Chart.

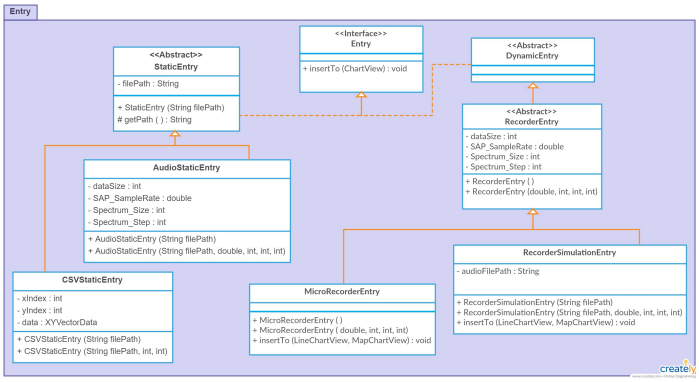


Fig 12 : Diagramme de classe - Entry

Comme le montre le diagramme de classe ci-dessus, les Entry sont déclinées en deux abstractions :

- StaticEntry : entrées statiques comprenant la lecture de fichiers Audios et de fichier CSV.
- DynamicEntry : entrées comprenant l'affichage dynamique de la lecture d'un fichier audio (simulation d'acquisition) ou de l'enregistrement direct d'un micro (acquisition temps réelle).

VI. Use cases – Cas d'utilisations

Un package spécialement dédié aux use cases à été créer dans le projet contenant 11 cas d'utilisations. Voici un listing de quelques uns des use cases :

- Affichage statique de données provenant d'un fichier CSV sur un LineChart
- Affichage dynamique simultané, sur un LineChart et un MapChart, de données acquisent depuis un Micro.
- Affichage dynamique simultané, sur un LineChart et un MapChart, de la simulation d'une acquisition de données provenant d'un fichier Audio.

VII. Conclusion & Ouverture

Mon travail sur ce projet aura donc principalement consister à comprendre , expliquer et initier une amélioration du Grapheur de M.Perronne.

La réalisation du “Document de Conception”, l'ajout de la Factory ainsi que de l'arborescence Entry permettra donc une prise en main beaucoup plus rapide pour les étudiants qui reprendront le projet qu'elle ne l'aura été pour moi.

Diverses propositions d'amélioration du projet sont proposées à la fin du “Document de Conception” afin d'aider ces prochains étudiants. Notemment l'ajout d'un nouveau type de Chart : les BarChart (ou Graphe en barre) ainsi que l'ajout d'une palette de paramétrage pour chaque Chart visant à remplacer les fichiers de configurations.