AOC - TP

Pierre GAUDICHON et Aurélien ANNE

Automne 2017

Introduction

Ce projet met en place le *design pattern* ***Active Object*** dans la création d'une IHM qui affiche le résultat d'un générateur d'entiers. Il utilise JavaFX avec FXML pour créer la fenêtre.

On utilise le pattern **Active Object** deux fois : pour transmettre les requète d'updates depuis les afficheurs vers le générateur, et ensuite pour transmettre les valeurs générées aux afficheurs. On utilise aussi le pattern **Strategy** pour permettre de changer la stratégie de diffusion des updates à l'exécution. Enfin, on utilise le pattern **Observer** pour permettre aux différentes classes de connaître les changements de valeurs.

- Repo Github du projet : https://github.com/AurelienAnne/AOC_TP

Choix techniques

Pattern strategy

La GUI de notre programme permet de choisir les valeurs générées par le programme.

- *Compteur* génère des nombres a partir de 1 depuis le début de l'exécution du programme.
- *Timestamp* génère des timestamp[1] qui permettent de visualiser et mesurer les délais que le canal créé.

Cette GUI permet aussi de choisir la stratégie de diffusion des updates.

- *Atomique* : l'update[2] suivante est envoyé seulement si la précédente est reçue.
- *Séquentielle* : envoie toute les updates dans l'ordre, l'ordre d'arrivé n'est pas vérifié.

[1] **timestamp** : horodatage
[2] **update** : mise à jour

On voit ici que le **pattern strategy** a été utilisé deux fois dans notre application, pour le message envoyé et la stratégie de diffusion.

Ces choix sont pris en compte en temps réel par l'application. La sélection dans la GUI change la stratégie utilisée.

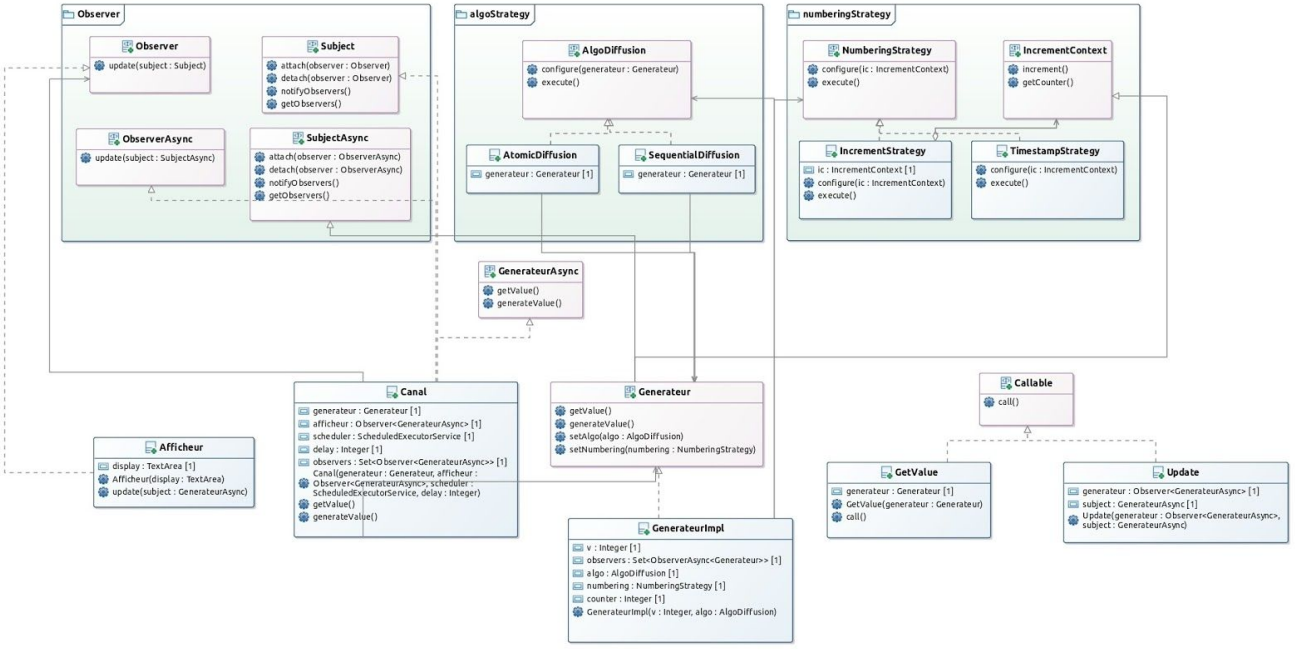
GUI

La GUI de notre programme est créé en utilisant le framework **JavaFX** et a l'aide d'un fichier **fxml**.

Le fichier **fxml** a été créé grace au logiciel Gluon qui permet facilement de générer des GUI à l'aide d'un système de *drag and drop*[3]. Ce logiciel permet aussi d'associer facilement des *controllers* aux actions accessible depuis l'interface.

Diagramme de classe M3

Nous avons essayé de séparer les classes en fonction de leur utilité, c'est pour cela que nous avons un paquetage pour les éléments du patron de conception Observer, et deux paquetages pour gérer les patrons de conception Stratégie concernant l'algorithme de diffusion et le numbering utilisés durant l'exécution de l'application.



[3] **drag and drop** : glisser-déposer

Nous avons mis en place deux fois le patron de conception **Active Object**, car nous avions besoin d'appels asynchrones dans les deux sens. La première application du patron concerne la fonction *Update*, tandis que la deuxième concerne la fonction *GetValue*.

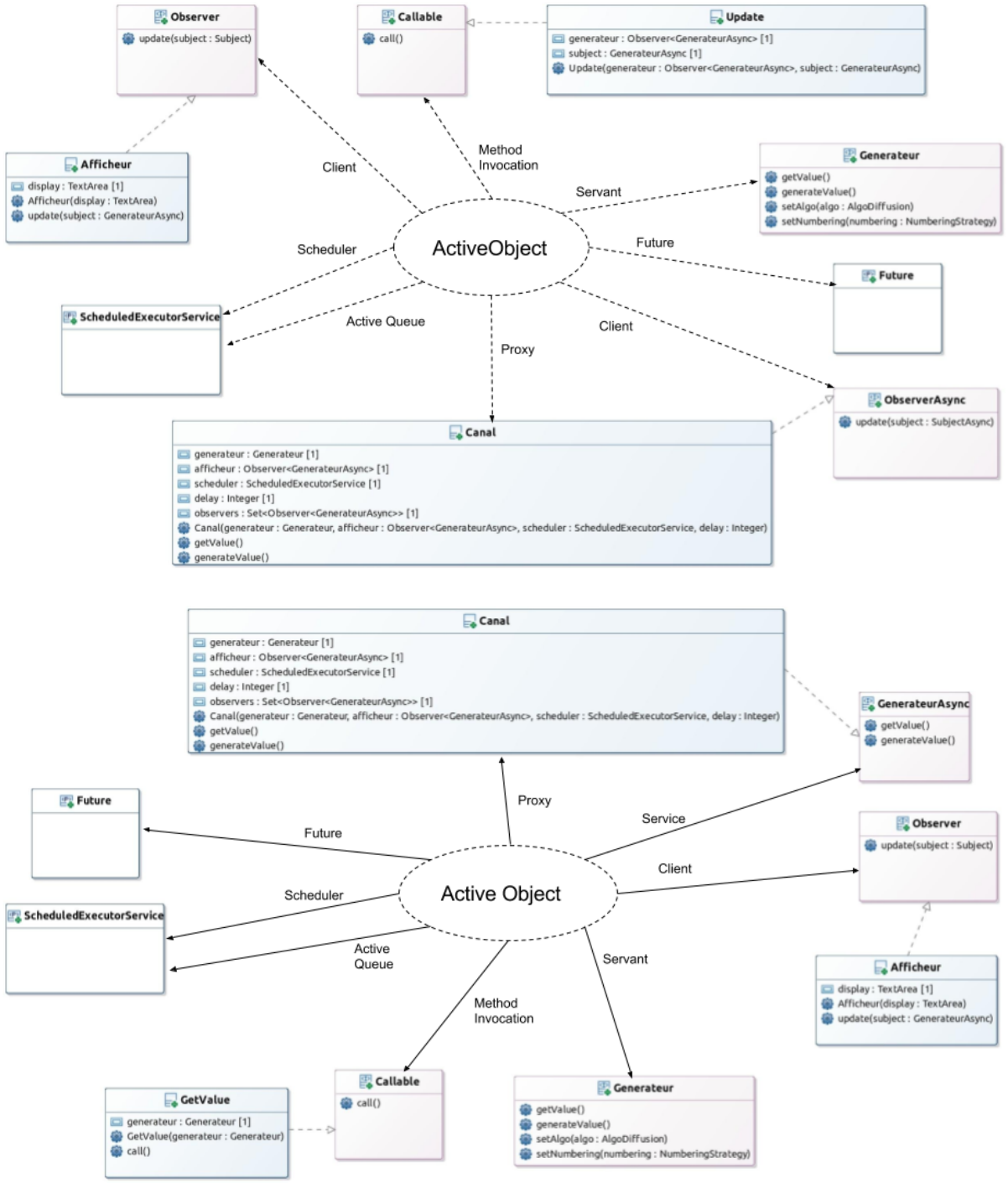
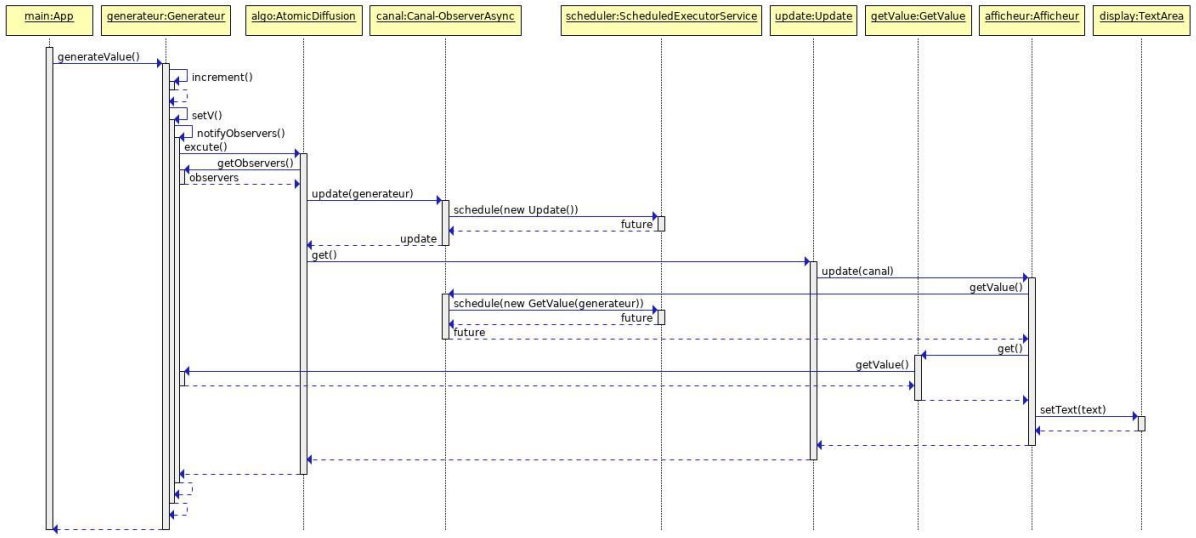


Diagramme de séquence M3



Ce diagramme met en évidence plusieurs points d'implémentation supplémentaires.

- On peut voir l'utilisation du **scheduler**[4] d'**Oracle** (*ScheduledExecutorService*).
- Pour représenter les résultats d'appels, on utilise des *Futures*.

[4] **scheduler** : ordonnanceur

Impressions d’écran

Voici le rendu final de notre application. Deux choix sont offerts à l’utilisateur en haut de la fenêtre concernant les stratégies de numération et de génération des valeurs. Ensuite, deux boutons permettent de démarrer et d’arrêter la génération et l’affichage des nombres. Enfin, quatre afficheurs affichent les valeurs générées selon l’algorithme et la numération choisis.