Rapport COA

Observer

2016-2017

Amel MESSADENE - Amina BOUSSALIA

Encadrant : Noël PLOUZEAU

Sommaire

Introduction

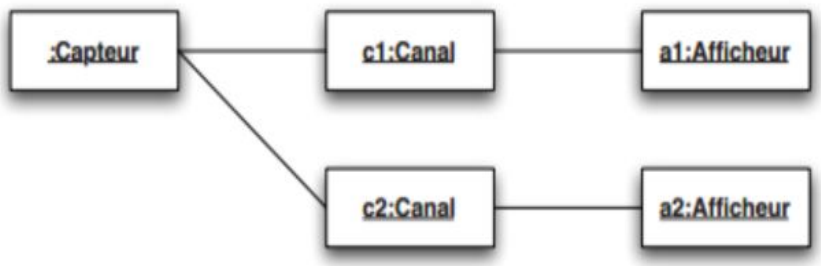
Le projet Observer a pour objectif de réaliser un service de diffusion de données de capteur. Ce service permet de diffuser un flot de valeurs vers des objets abonnés et exécutés dans des threads différents de la source.

Notre solution s'appuie sur le patron de conception Active Object ainsi que les autres patrons de conception vus en cours d'AOC.

Nous avons étudiés pendant les TD de COA trois versions de ce service mais seule la dernière version (V3) sera développée pour ce projet.

Descriptif de notre service de diffusion

Au final, ce service permettra de diffuser un flot de valeurs vers des objets abonnés exécutés dans des threads différents de la source du service. L'objectif du TP étant la mise en œuvre parallèle d'Observer, les données diffusées seront une séquence croissante d'entiers (c'est à dire un simple compteur). Le compteur sera incrémenté à intervalle fixe. La transmission de l'information vers les abonnés au service emploiera un canal avec un délai de transmission aléatoire.



L'architecture comprendra donc :
● une source active (capteur), dont la valeur évolue de façon périodique .
● un ensemble de canaux de transmission avec des délais variables .
● un ensemble d'afficheurs réalisés en utilisant JavaFX .
● un ensemble de politiques de diffusion, comprenant .

○ la diffusion atomique consiste à envoyer la donnée du capteur à tous ses observateurs en s'assurant que cette dernière est affichée sur tous les afficheurs avant de se modifier. Ainsi, ces derniers ne doivent passer aucune valeur. Ce qui nécessitait de bloquer le capteur tant que tous les afficheurs n'ont pas reçu la valeur.

○ la diffusion séquentielle pareil que l'atomique tous les observateurs reçoivent la même valeur, mais la séquence peut être différente de la séquence réelle du capteur. toutes les valeurs du capteur ne sont pas obligatoirement affichables.Cela se traduit par le fait que le capteur continue de fonctionner durant la transmission des valeurs.

○ la gestion par époque comme vu en cours, ce qui consiste à ajouter un numéro de version à la valeur transmise.

Choix Technique

Notre application a été développé en Java avec une interface homme machine écrite en JavaFX.

Pour réaliser ce projet, nous avons utilisé des patron de conception vu en cours tels que: Active Object, Observer et Strategy.

Observer afin de mettre à jour les afficheurs lors d'un changement de valeur du capteur.

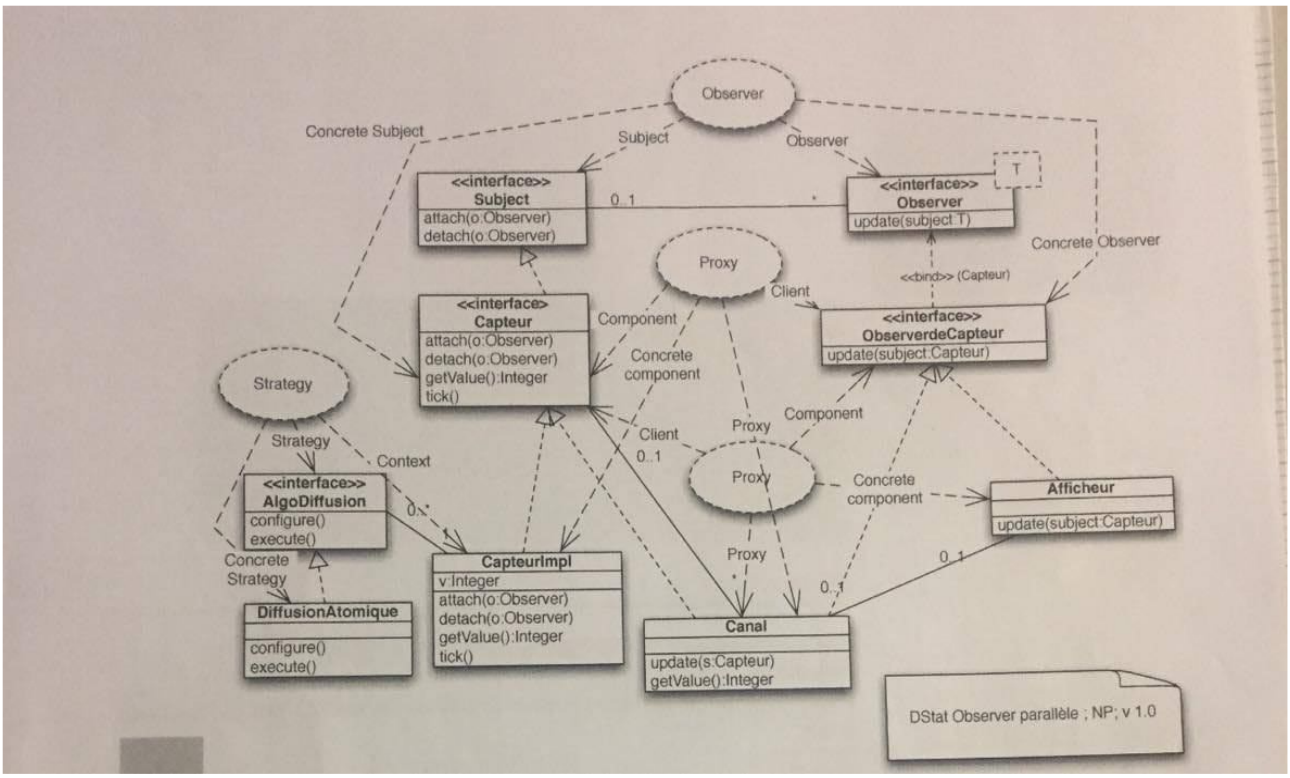
Strategy car nous devions définir trois stratégies de diffusion pour les valeurs du capteur.

Ainsi que des classes Oracle telles que :

- ScheduledExecutorService
- Future
- ExecutionException
- Platform
- Timer et TimerTask
- Callable

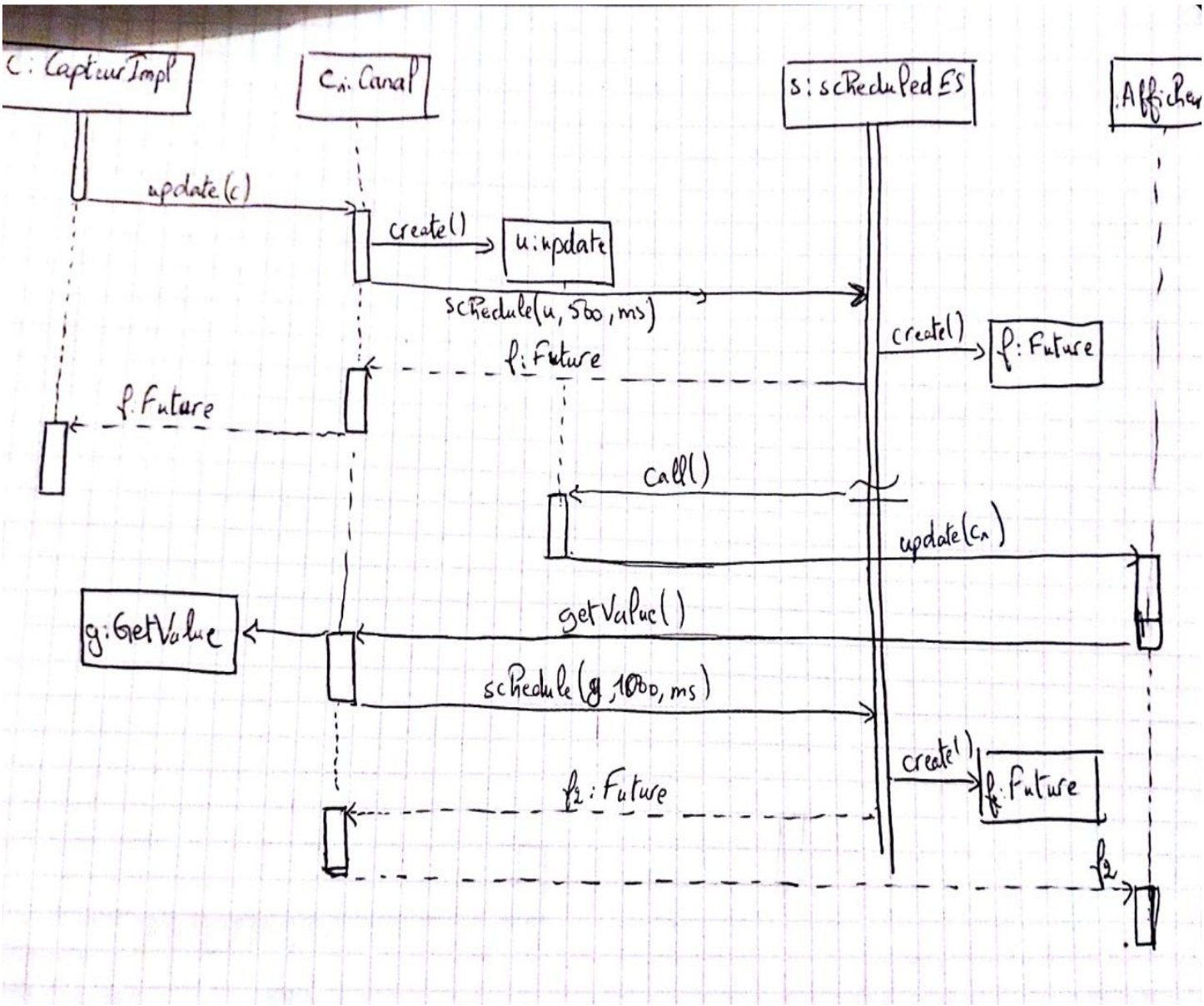
Architecture de conception

Pour la mise en oeuvre de notre projet on a suivi le diagramme de classe donné en cours, et qui est mentionné ci dessous:



voici également un des diagrammes de séquence donné en cours et sur lequel on s'est basé lors du développement de notre projet.

NB. Vous trouverez dans notre dêpot Git le diagrammes de classe globale, à ce lien: https://github.com/amelMess/TP_COA/tree/master/src/resources



Problèmes rencontrés

Dans un premier temps, la difficulté consistait en mise en oeuvre du patron de conception Active Object, il nous a fallu nous documenter et comprendre son fonctionnement à travers des exemples ainsi que des diagrammes de séquences et ceci pour une bonne utilisation du patron.

Egalement, il nous a fallu assimiler les différences entre les algorithmes de diffusion pour ensuite pouvoir les formaliser en langage Java.
Au bout de plusieurs passage d'une diffusion à une autre le programme plante sans erreurs.
La diffusion par époque n'a pas été faite.

Conclusion

Ce projet nous a permis l’utilisation de nouveaux concepts, tel que le patron de conception Active Object ainsi des classes Oracles ScheduledExecutorService, Future et Callable, ce qui a enrichi notre formation et nous a permis également d'appréhender le parallélisme en Java différemment.