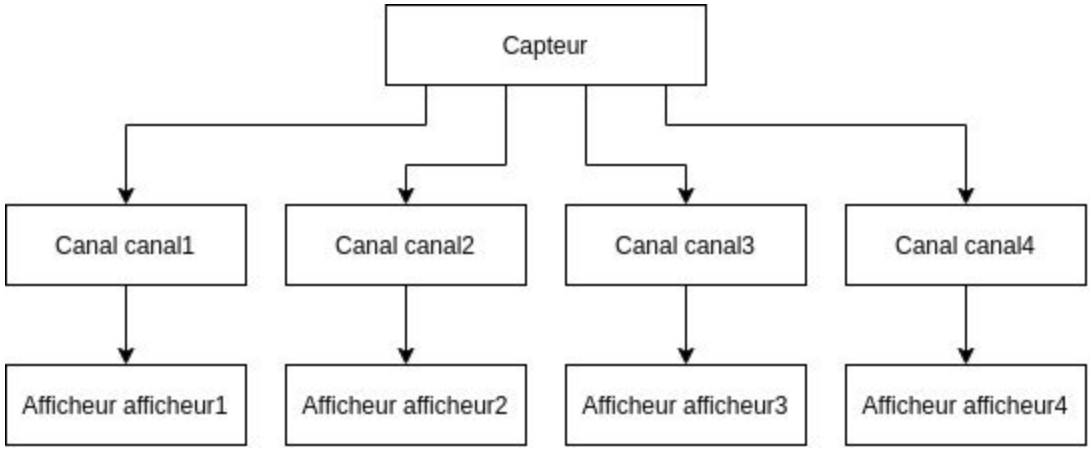
Architectures à Objets Canonique

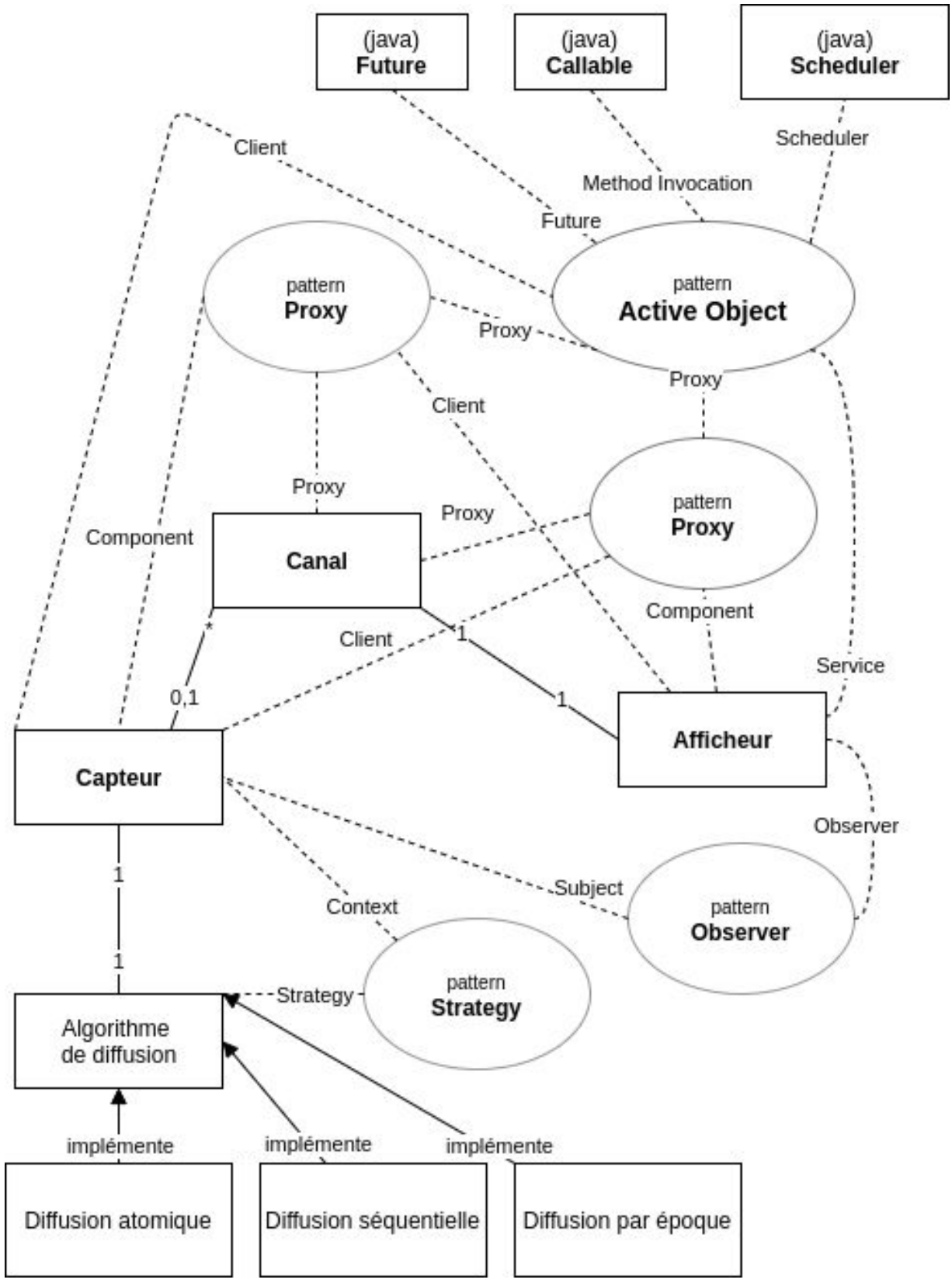
# Patron de Conception: Active Object
## Proxy, Observer & Strategy en Java




## Introduction

Ce projet repose sur les patrons de conception Active Object, Observer, Strategy et Proxy mais aussi sur l'association entre ces différents éléments au sein d'un même programme. Le but étant d'afficher la valeur d'un compteur incrémentale faisant office de capteur sur 4 afficheurs utilisant leur propre canal dont le délai de transmission des informations varie. Ce délai est une simulation du délai pouvant être impliqué lors de la mise à jour des afficheurs ainsi que lors de l'obtention des données de notre capteur.

On peut retrouver page suivante un aperçu de l'architecture de notre projet, suivi de précisions sur les implémentations des différents patterns inclus dans celle-ci.


(java)
Future
(java)
Callable
(java)
Scheduler
Client
Scheduler
Method Invocation
Future
pattern
Proxy
pattern
Active Object
Proxy
Proxy
Client
Proxy
pattern
Proxy
Component
Canal
Proxy
Component
Client
1
Service
0,1
1
Afficheur
Capteur
Observer
1
Subject
Context
pattern
Observer
1
Strategy
pattern
Strategy
Algorithme
de diffusion
implémente
implémente
implémente
Diffusion atomique
Diffusion séquentielle
Diffusion par époque

# Pattern Strategy

Comme on le voit ci-dessus, la transmission des données est effectuée par différents algorithmes de diffusion constituant le pattern Strategy. Le choix de l'algorithme aboutissant à des méthodes de transmission qui impacteront les résultats contenus dans les afficheurs.

On retrouve sur ce pattern Strategy trois méthodes de diffusions différentes :

## Diffusion atomique

Dans cette méthode de diffusion, chaque afficheur (observer) enregistre la même suite de données que celles prises par le capteur. Pour cela, on effectue un blocage des appelants de la fonction tant que tous les afficheurs n'ont pas récupéré la valeur précédente du capteur par le canal qui leur est associé. Formellement, on a :

- $\forall$ **x**,**y** : DA(**y**) = DA(**x**) = DC

Avec DA(x) données de l'afficheur x, et DC données du capteur.

## Diffusion Séquentielle

Avec cette méthode de diffusion, tous les afficheurs possèdent à la fin de l'exécution les mêmes données, reçues depuis leur canaux. Contrairement à la méthode atomique, on ne retrouve cependant pas nécessairement les mêmes valeurs que celles du capteur, mais simplement un de ses sous-ensembles.

Techniquement, la diffusion n'est simplement pas réalisée si les tous les afficheurs n'ont pas terminé d'afficher la valeur précédentes du capteur (pas de blocage à l'entrée de l'exécution). Formellement, on a :

- $\forall$**x**,**y**: DA(**y**) = DA(**x**) et DA(**x**) $\in$ DC (et possiblement, pas nécessairement DC $\in$ DA(x))

## Diffusion par époque

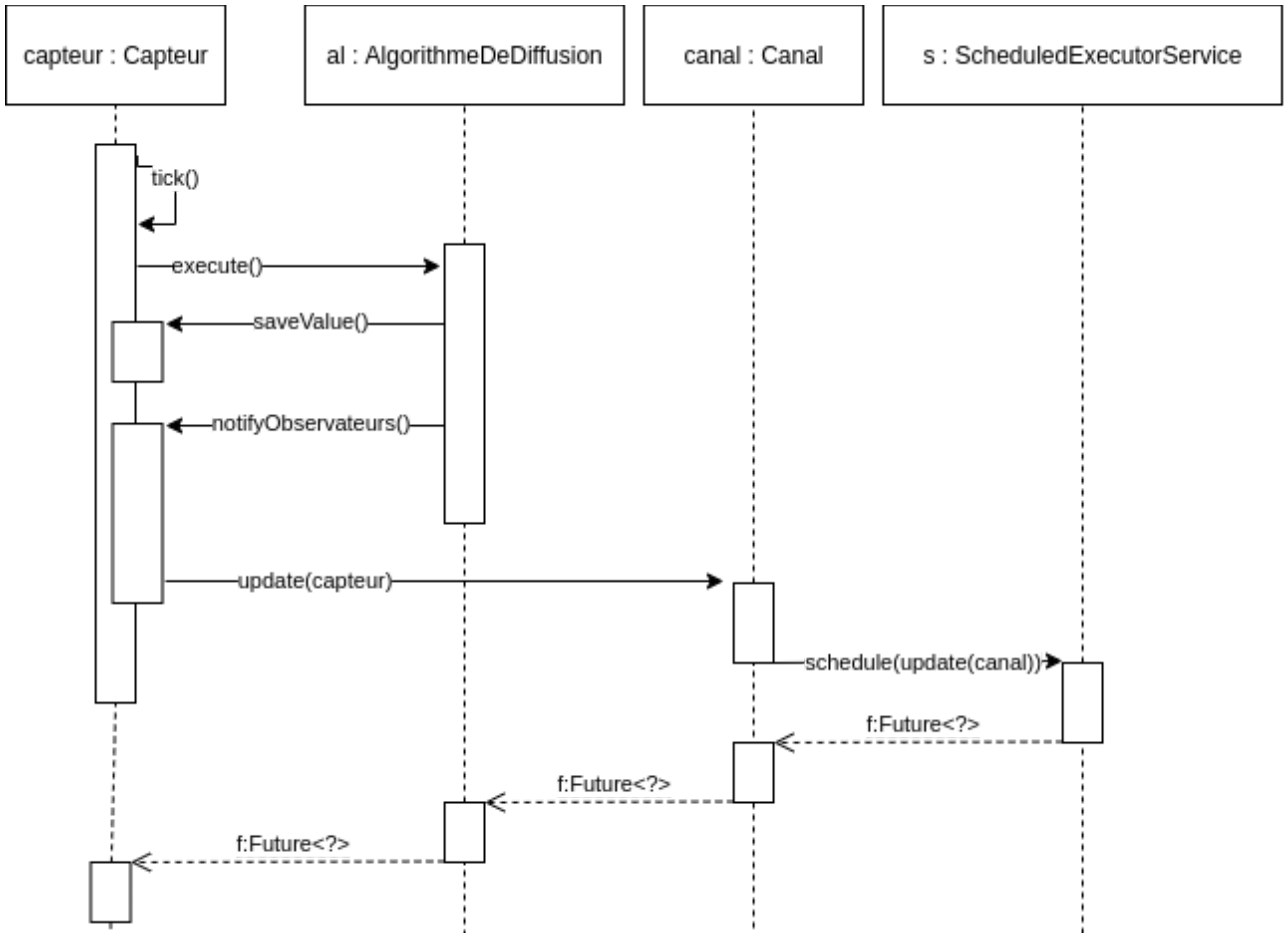
Pour cette troisième méthode, nous retrouvons simplement des données pouvant être différentes pour chaque afficheur. La diffusion des données est réalisée sans aucun blocage entre les différents appels. Ceci résulte dans la réception de valeurs “périmées” par l’afficheur, dûe aux délais de circulation des informations inclus dans les canaux. Ces valeurs sont filtrées lors de la réception et ne sont pas ajoutées aux données de l’afficheur.

On a donc simplement : $\forall\ \mathbf{x},\mathbf{y},i : ( (\mathbf{x} > \mathbf{y}) \Rightarrow DA(i)[\mathbf{x}] > DA(i)[\mathbf{y}] )$ et $(DA(i) \in DC)$

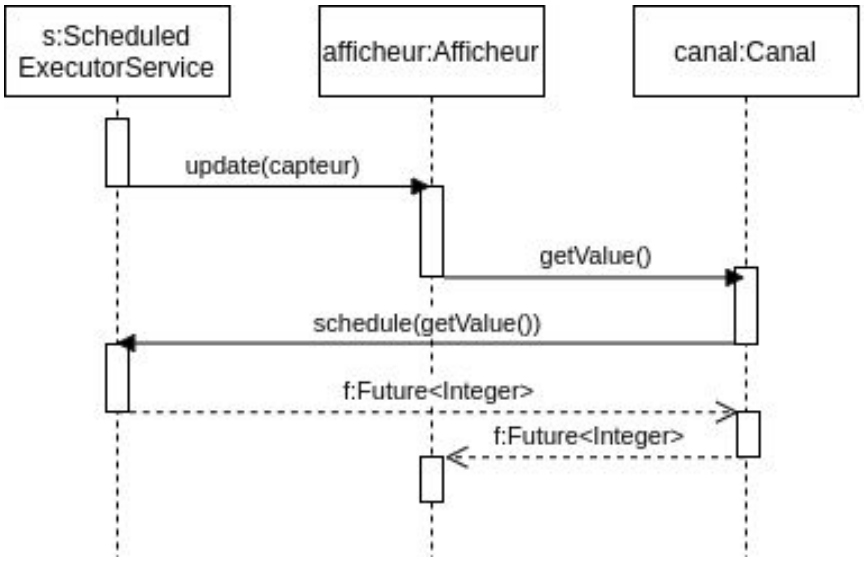
## Pattern Proxy

Active Object nécessite l'emploi de deux proxys entre le Capteur et l'Afficheur. Ces deux proxys sont implémentés dans le Canal. Le premier proxy permet de récupérer la valeur du capteur dans Canal avec la fonction "getValue()" et l'autre proxy se charge de la mise à jour de l'affichage avec la fonction "update()". En vue du caractère asynchrone de ces deux méthodes, elles rendent un Future qui est résolu dans leur client respectif.

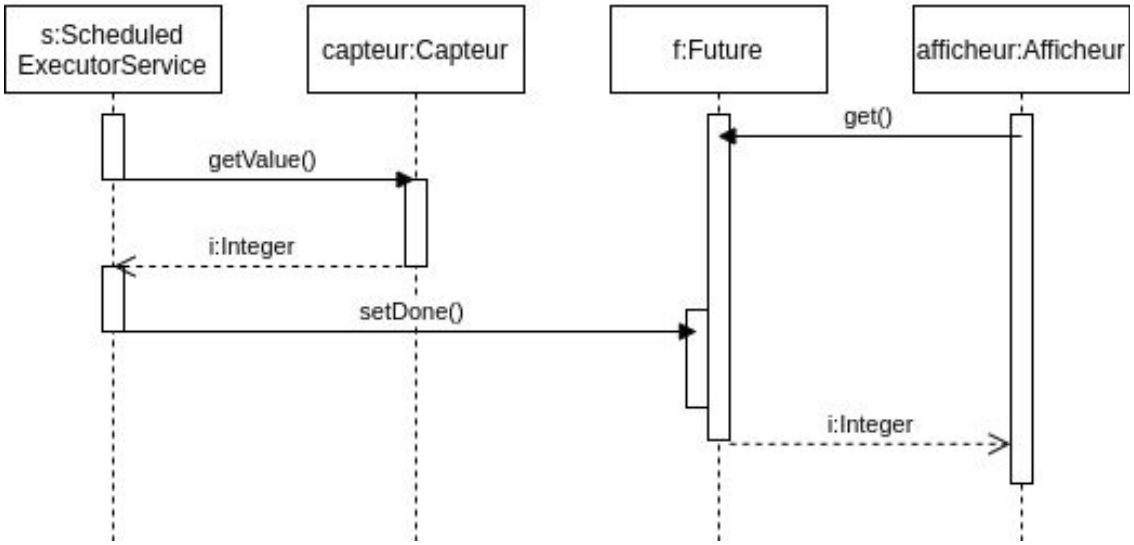
Voici des diagrammes de séquence correspondant au processus permettant d'effectuer **update** et **getValue**:




Sur ce diagramme, on observe le processus mis à l'œuvre lors de l'appel de la fonction tick() du capteur. Celle-ci planifie l'exécution de la méthode update des afficheurs par le biais du ScheduledExecutorService.




Sur le diagramme ci-dessus, on observe la création du Future<Integer> créé par le ScheduledExecutorService. L'afficheur pourra ensuite effectuer un get() sur ce dernier et sera bloqué en attendant la résolution du Future.




Sur ce diagramme de séquence, on retrouve l'exécution du Callable scheduled sur le diagramme précédent. Une fois la valeur du capteur récupérée, le scheduler effectue un setDone() sur notre Future, indiquant à l'afficheur qu'il peut exploiter la donnée.

## Pattern Observer

Dans le cadre de ce programme, le pattern observer est également implémenté. Le capteur faisant évidemment office de Subject, et les afficheurs étant les Observers.

La mise en place de ce pattern est visible dans les diagrammes de séquence listés ci-dessus.

## Résultats des tests

On peut retrouver ci-dessous un rapide aperçu du résultat de nos tests. Les assertions vérifiées sont celles détaillées dans la partie Pattern Strategy. Ces tests peuvent être réalisés depuis la classe “Oracle” du package tests.

### Diffusion atomique :

```
Feb 06, 2021 6:05:52 PM tests.Oracle testAtomique
INFO: Tests diffusion atomique
Contenu de l'afficheur 0 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30
Contenu de l'afficheur 1 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30
Contenu de l'afficheur 2 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30
Contenu de l'afficheur 3 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30
Feb 06, 2021 6:07:42 PM tests.Oracle testAtomique
INFO: Tests diffusion atomique terminés sans avoir rencontré d'erreurs.
```

On constate ici le respect de la propriété suivante :

- ∀ **x**,**y** : DA(**y**) = DA(**x**) = DC

Avec DA(x) données de l’afficheur x, et DC données du capteur, étant les entiers compris sur l’intervalle [1, 30].

## Diffusion séquentielle :

```
Feb 06, 2021 6:05:27 PM tests.Oracle testSequentiel
INFO: Tests diffusion sequentielle
Contenu de l'afficheur 0 : 1, 9, 16, 24,
Contenu de l'afficheur 1 : 1, 9, 16, 24,
Contenu de l'afficheur 2 : 1, 9, 16, 24,
Contenu de l'afficheur 3 : 1, 9, 16, 24,
Feb 06, 2021 6:05:52 PM tests.Oracle testSequentiel
INFO: Tests diffusion séquentielle terminés sans avoir rencontré d'erreurs.
```

On constate ici le respect de la propriété suivante :

- ∀**x**,**y**: DA(**y**) = DA(**x**) et DA(**x**) ∈ DC

Les valeurs de DC étant les entiers compris entre [1, 30].

## Diffusion par époque :

```
Feb 06, 2021 7:11:02 PM tests.Oracle testEpoque
INFO: Tests diffusion par epoque
Contenu de l'afficheur 0 : 6, 7, 8, 9, 10, 12, 13, 16, 17, 20, 21, 24, 26, 27, 29, 30,
Contenu de l'afficheur 1 : 6, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25, 26, 27, 29, 30,
Contenu de l'afficheur 2 : 7, 8, 10, 12, 14, 15, 16, 17, 18, 20, 22, 24, 25, 26, 28, 29, 30,
Contenu de l'afficheur 3 : 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19, 20, 23, 24, 25, 26, 27, 29, 30,
Feb 06, 2021 7:11:27 PM tests.Oracle testEpoque
INFO: Tests diffusion par époque terminés sans avoir rencontré d'erreurs.
```

On constate ici le respect des propriétés suivantes :

- ∀ **x**,**y**,i : ( (**x** > **y**) ⇒ DA(i) [**x**] > DA(i) [**y**] ) et (DA(i) ∈ DC)

Les valeurs de DC étant les entiers compris dans l'intervalle [1, 30].