COA
RAPPORT TP

Chouaib Hentabli
Raiçal Irifi
Master 2 Génie Logiciel
ISTIC - Université Rennes1
Année Universitaire 2016-2017

Sommaire

Année Universitaire 2016-2017 1
1. Introduction : 4
2. Présentation du projet : 4
2.1. Spécification 4
2.2. Les algorithmes de diffusion : 5
2.2.1. Algorithme de diffusion atomique : 5
2.2.2. Algorithme de diffusion séquentiel : 5
2.2.3. Algorithme de diffusion par époque : 5
3. Conception : 5
4. Réalisation : 9
5. Difficultés rencontrées 10
6. Conclusion : 11
Bibliographie : 11

1. Introduction :

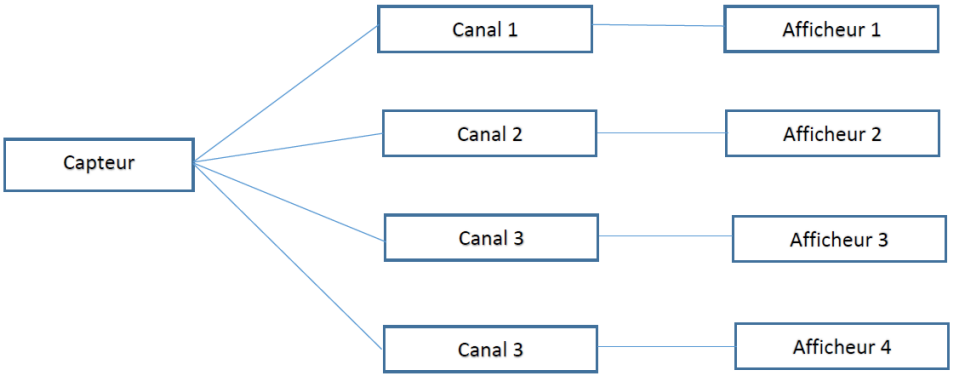
Ce rapport a pour butde présenter le projet qu'on a développé pour le module COA.
Ce dernier, nous a permis de comprendre la mise en œuvre d'architectures asynchrones, via l'implémentation du patron de conception Active Object et l'utilisation du langage Java.

L'objectif du projet est de réaliser un service de diffusion de données de capteur. La solution à construire devait s'appuyer sur des mécanismes de programmation par «threads», les patrons de conception vue dans le module AOC ainsi que le patron de conception Active Object.
Ce dernier, nous a permis de limiter la manipulation des «Threads» Java qui se révèle, le plus souvent, très fastidieuse.

2. Présentation du projet :

2.1. Spécification

Le projet consiste à développer une application en JAVA,qui propose à l'utilisateur un programme permettant la diffusion d'un flot de données sur des différents canaux d'un capteur vers différents afficheurs.



Pour cela, nous devions mettre en place :

- un capteur transmettant des valeurs à des canaux de diffusion.
- Les canaux devaient se charger de créer des « Method Invocation » permettant l'envoi des valeurs sur des afficheurs.
- Les canaux devaient posséder des délais de propagation aléatoires, simulant une latence.
- L'utilisateur peut sélectionner un algorithme de diffusion (atomic, séquentiel ou epoc).

2.2. Les algorithmes de diffusion :

Dans notre projet, nous allons implémenter trois algorithmes :

2.2.1. Algorithme de diffusion atomique :

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. De plus, ces derniers ne doivent occulter aucune valeur. Cela nécessitait donc une action à mettre en place pour bloquer le capteur tant que tous les afficheurs n'ont pas reçu leur valeur.

2.2.2. Algorithme de diffusion séquentiel :

La diffusion séquentielle ressemble trait pour trait à la diffusion atomique à l'exception près que 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. Ce qui a pour conséquence, le non affichage de certaines valeurs.

2.2.3. Algorithme de diffusion par époque :

La diffusion par époque, consiste à ajouter un numéro de version à la valeur transmise. Il n'existe aucune contrainte tant au niveau des afficheurs qu'au niveau du capteur.

3. Conception :

3.1. Les patrons de conception

Dans ce qui, nous allons présenter les différents patrons de conception utilisées dans notre application :

- **Observer**

Le patron de conception « observer/subject» est utilisé en programmation pour envoyer un signal à des modules qui jouent le rôle d'observateurs. En cas de notification, les observateurs effectuent alors l'action adéquate en fonction des informations qui parviennent depuis les modules qu' ils observent (les « Subject »).

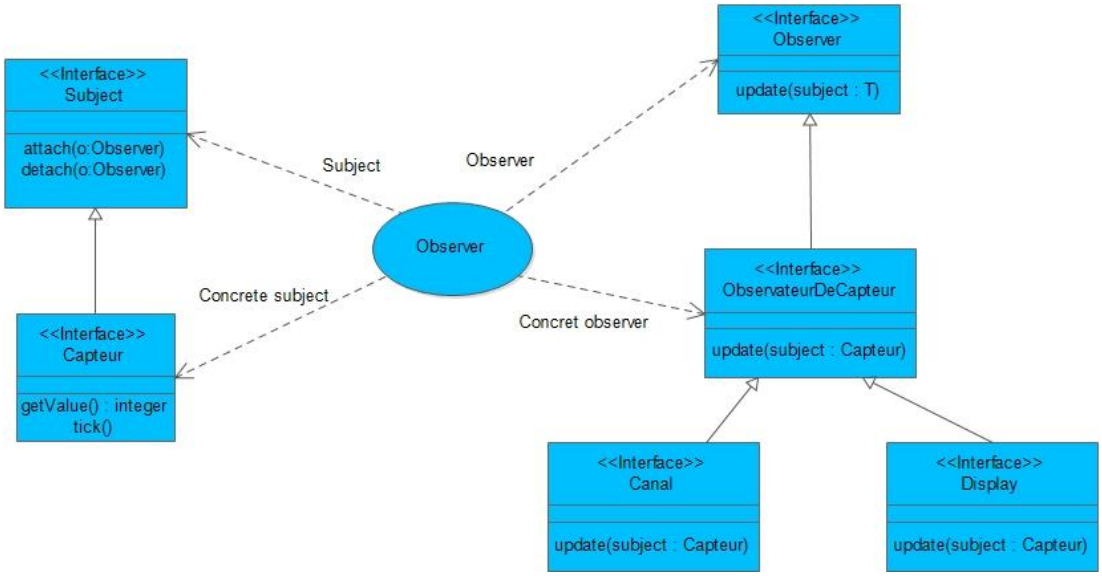


Figure 1: Patron de conception « Observer ».

- **Strategy**

Le patron stratégie est un patron de conception de type comportemental grâce auquel des algorithmes peuvent être sélectionnés à la volée au cours de l'exécution selon certaines conditions. Il est particulièrement utile pour des situations où il est nécessaire de permuter dynamiquement les algorithmes utilisés dans une application.

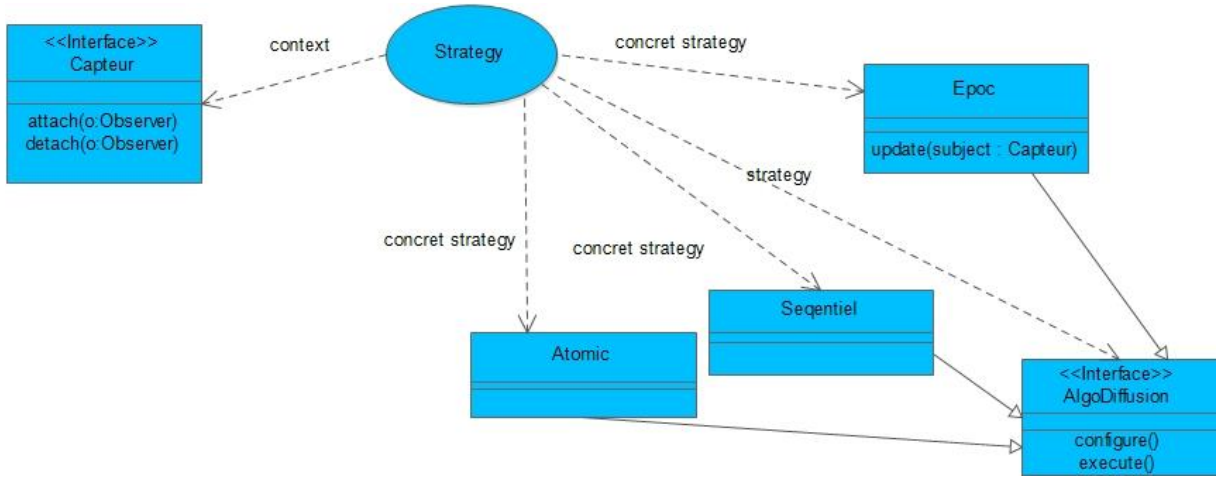


Figure 2 : Patron de conception « Strategy ».

- **Proxy**

Un proxy est une classe se substituant à une autre. Par convention et simplicité, le proxy implémente la même interface que la classe à laquelle il se substitue. L'utilisation de proxy ajoute une indirection à l'utilisation de la classe à substituer.

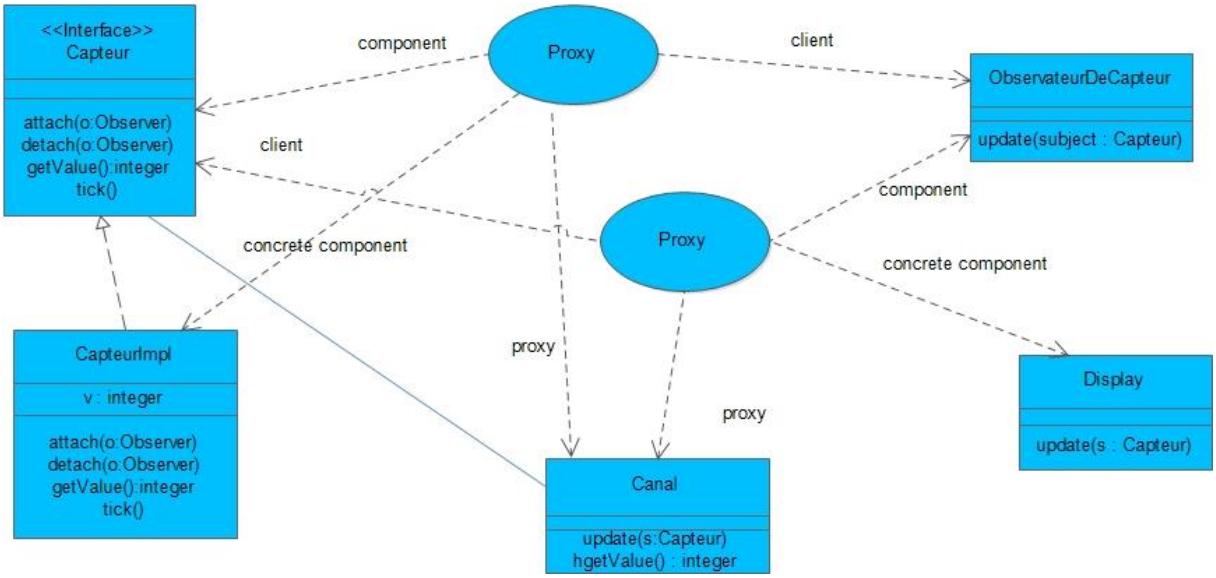


Figure 3 : Patron de conception « Proxy ».

- **Active Object :**

Le patron de conception Active Object permet de découpler les méthodes d'exécution des méthodes d'invocation pour améliorer la concurrence et simplifier l'accès synchronisé d'un objet qui réside dans son propre thread de contrôle. Le principe est de pouvoir faire communiquer les objets entre eux de manières à ce que les messages échangés soient asynchrones. Sa grande force est de permettre au développeur de limiter la programmation parallèle classique, à l'aide de « Threads » Java par exemple. En effet, ce patron de conception est capable de gérer de manière automatique les actions asynchrones sans que le développeur n'ait à se soucier des éventuels problèmes de synchronisation et d'accès concurrent sur une même variable. Il en résulte que l'ordre d'exécution de méthodes peut différer de leur ordre d'invocation par exemple.

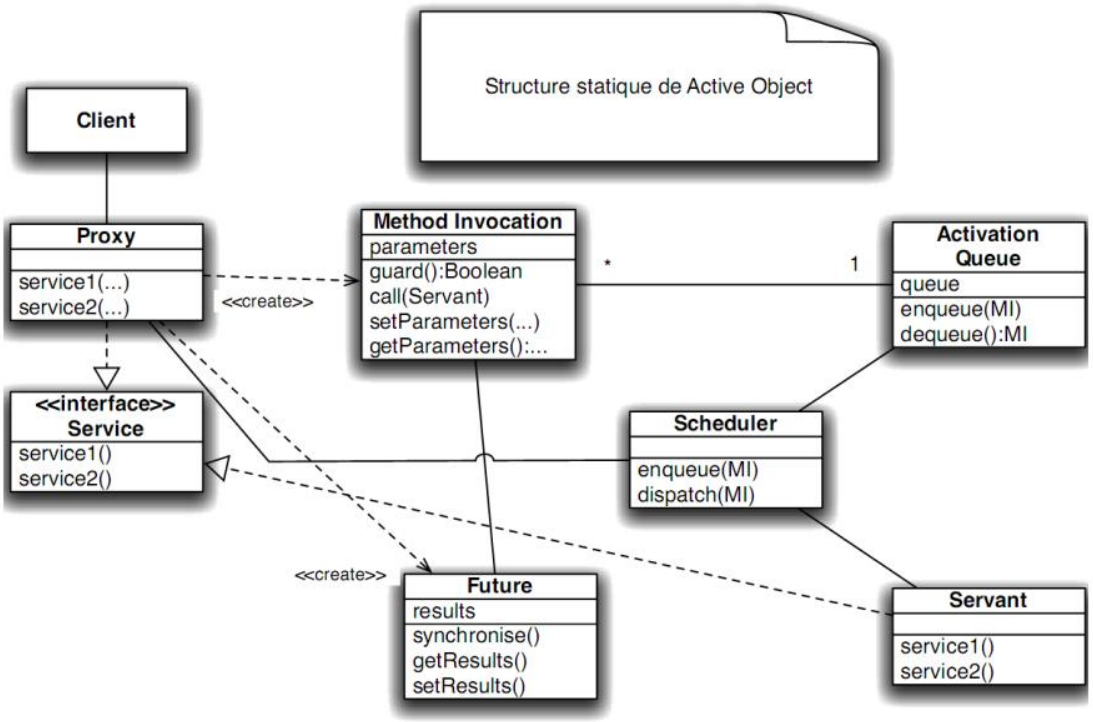


Figure 4 : Patron de conception « Active Object » (structure statique).

D'après la figure ci-dessus, on constate que « Active Object » définit plusieurs rôles :

- **Client :** Dans notre application, le patron est utilisé deux fois (une fois pour l'appel de la « Method Invocation update » et une fois pour l'apelle de la « Method Invocation getValue ») et certaines classes n'ont pas le même rôle selon le cas. Ainsi, les classes implémentant l'interface « AlgoDiffusion » et celles implémentant « Afficheur » sont les « Client ».
- **Servant :** Ce sont les classes qui mettent en œuvre les services appelés lors de l'exécution des « Method Invocation », c'est-à-dire celles implémentant « AlgoDiffusion » et « Afficheur ».
- **Future :** Ce rôle est géré par la bibliothèque standard Java. Il représente l'objet retourné après l'appel des services par les « Method Invocation ».
- **Scheduler** : Il planifie l'appel des « Method Invocation » de manière asynchrone, c'est la classe « Scheduler » dans notre cas. Activation Queue : Ce rôle est géré par la bibliothèque standard Java, c'est la file d'attente où sont placées les « Method Invocation » avant leur appel.
- **Method Invocation :** Le concept de méthode est encapsulé dans un objet qui représente les « Method Invocation » du patron. Elles sont chargées de réaliser l'appel du service à leur exécution via leur méthode « Call ». Ce rôle est joué par les classes implémentant l'interface « Method Invocation » de notre implémentation.
- **Proxy :** Un proxy standard, ce rôle est joué par les classes implémentant l'interface « Canal ».

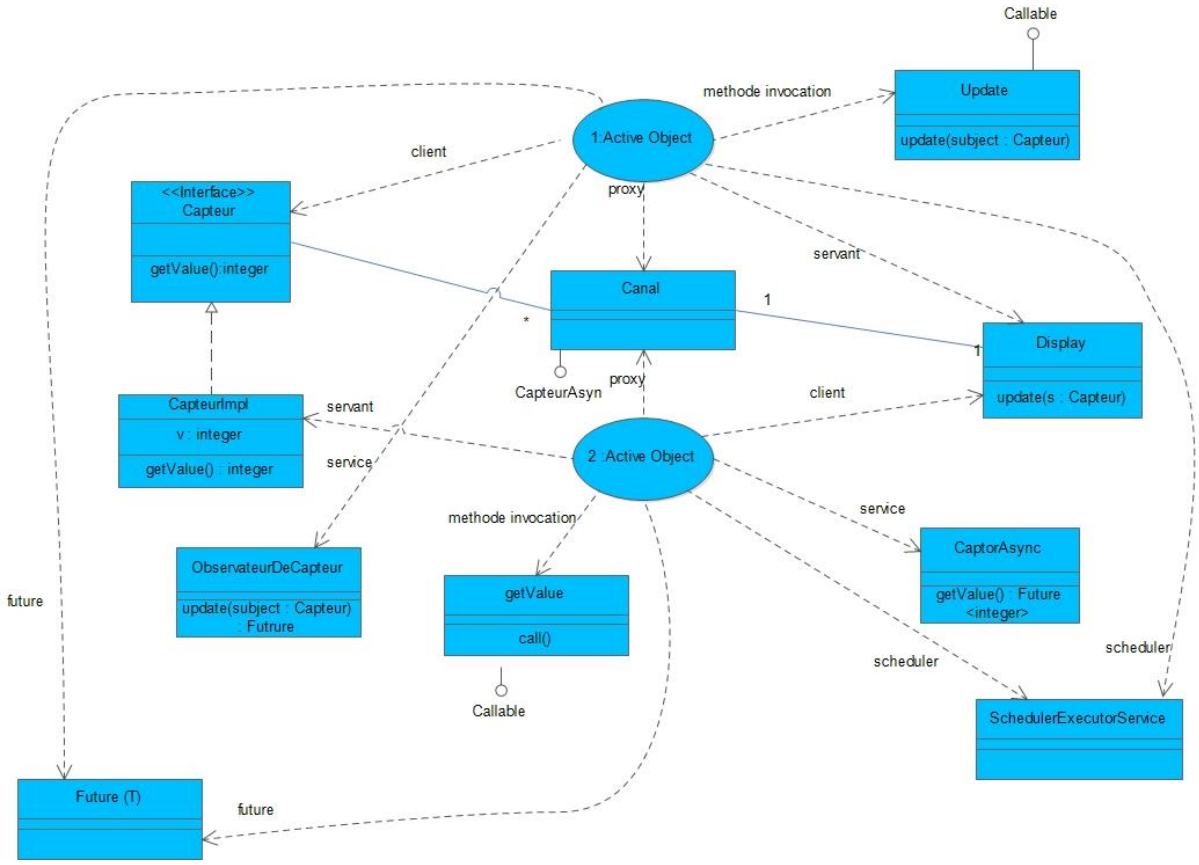


Figure 5 : mise en œuvre d'Active Object

4. Réalisation :

Afin de s'assurer du bon fonctionnement de notre application, nous avons conçu une IHM en utilisant la librairie javaFx. Cette dernière, nous a permet :

- de s'assurer du bon fonctionnement de l'horloge grâce à l'affichage de la valeur du capteur,
- de voir la latence des canaux,
- de tester les différents algorithmes de diffusion.

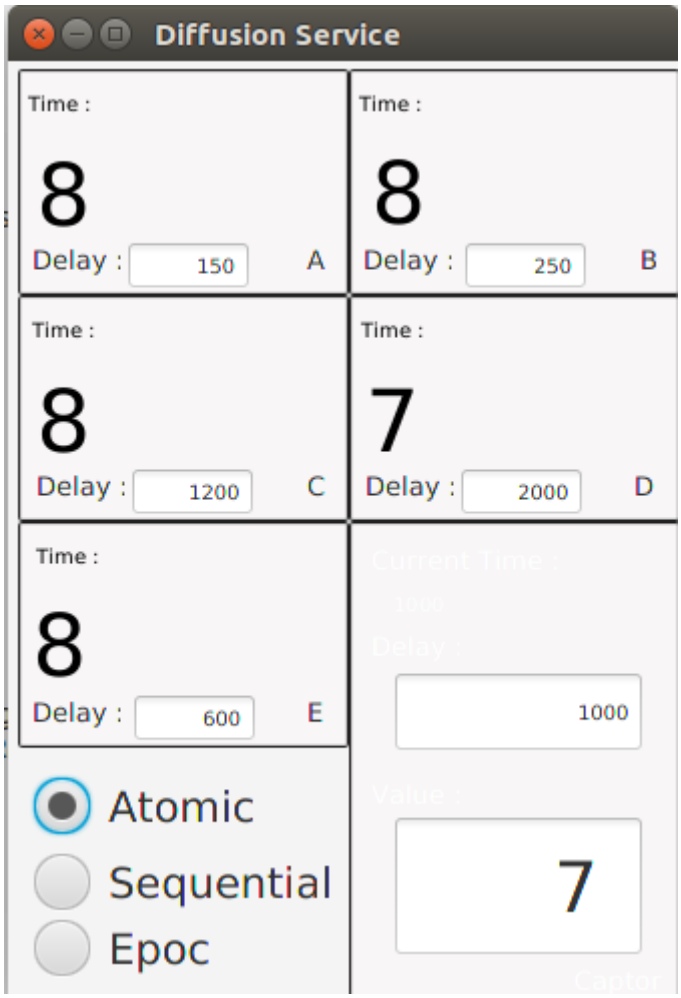


Figure 6 : Interface de l'application

5. Difficultés rencontrées

Au début, nous avons eu des difficultés pour comprendre le fonctionnement de Active Object. Il nous a fallu nous documenter et analyser son fonctionnement au travers de multiples diagrammes de séquences.

Ensuite, nous avons repérer le rôle de chaque patron de conception (proxy, strategy…)

Et de les mettre en œuvre.

Enfin, après avoir terminé la phase d'analyse, on a entamé la phase développement. Cette phase nous pris un peu de temps aussi, et ça est du aux erreurs qu'on a rencontré pendant cette phase.

6. Conclusion :

Ce projet nous a permit, non seulement de nous initier au monde la programmation asynchrone, mais aussi de renforcer notre compréhension des différents patrons de conception vu durant le cours de AOC.
Enfin, ce projet a été pour nous une véritable ouverture sur le monde de la programmation parallèle, monde qui représente certainement l'avenir du développement en tirant parti des architectures multi-cœurs.

Bibliographie :

http://docplayer.fr/226876-Projet-active-object.html