Rapport projet AOC

DEMEAUTIS Victor
DELOURME Cedric

Principe du projet

Le but du projet est de constituer une application java simulant une communication à travers un réseau, entre un générateur et plusieurs afficheurs. Pour cela il nous faut créer une communication asynchrone entre le générateur et les afficheurs. La solution technique choisie (et imposée) est d'utiliser le patron de conception ActiveObject.

Active Object

Le but du patron de conception active object est de créer une concurrence entre les communications (dans notre cas) par le biais de méthode asynchrone.

Active Object utilise d'autres patrons de conception telles que :

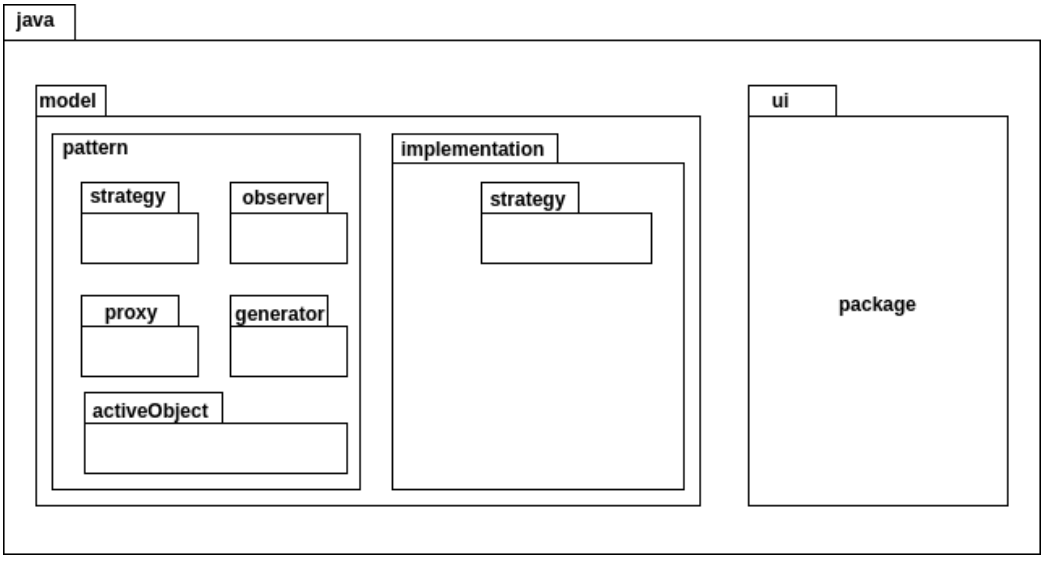
- Observateur
- Proxy
- et dans notre cas nous utilisons aussi Strategie

Choix Technique

Au niveau du choix technique, nous avons décidé d'utiliser JAVA-FXML pour créer l'interface graphique car, c'est pour nous, la seule technologie réellement utilisée dans le domaine des interfaces graphiques réalisées en java.

Organisation du projet

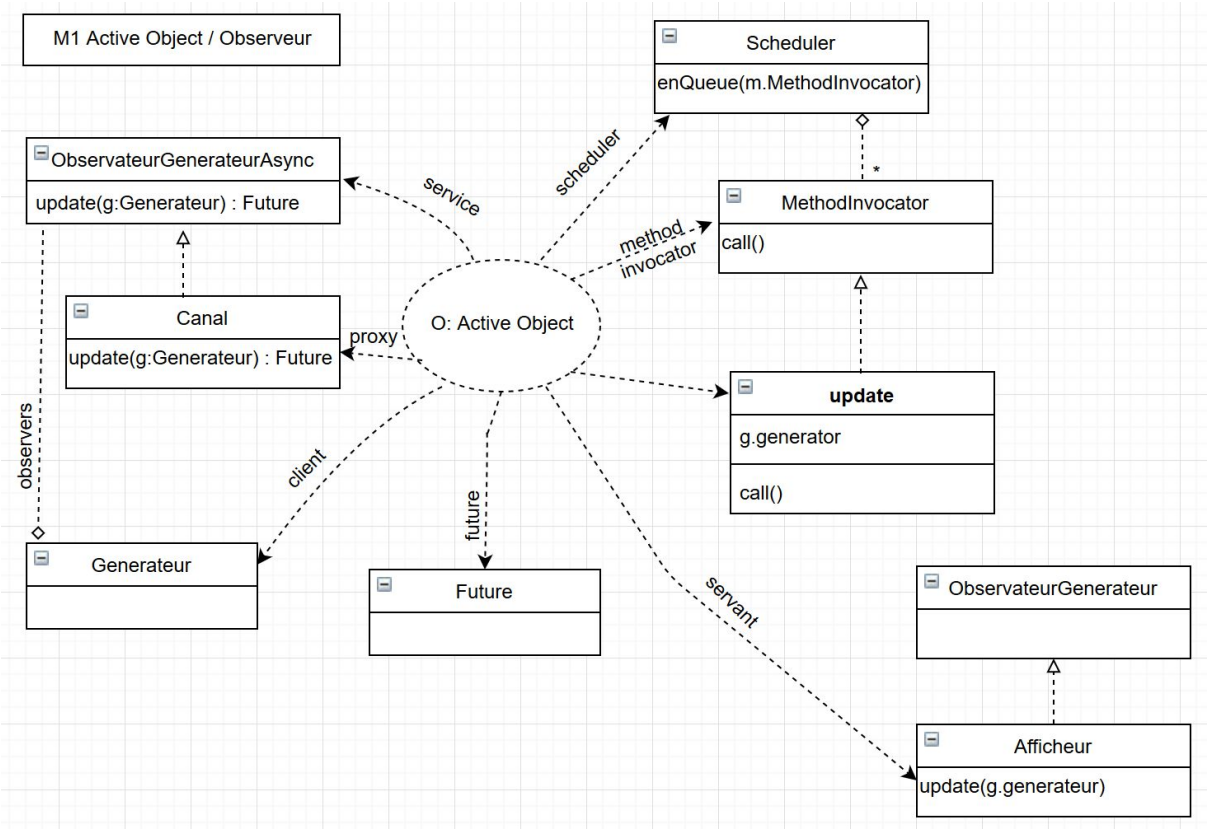
Notre projet est organisé suivant le diagramme de packages suivant :



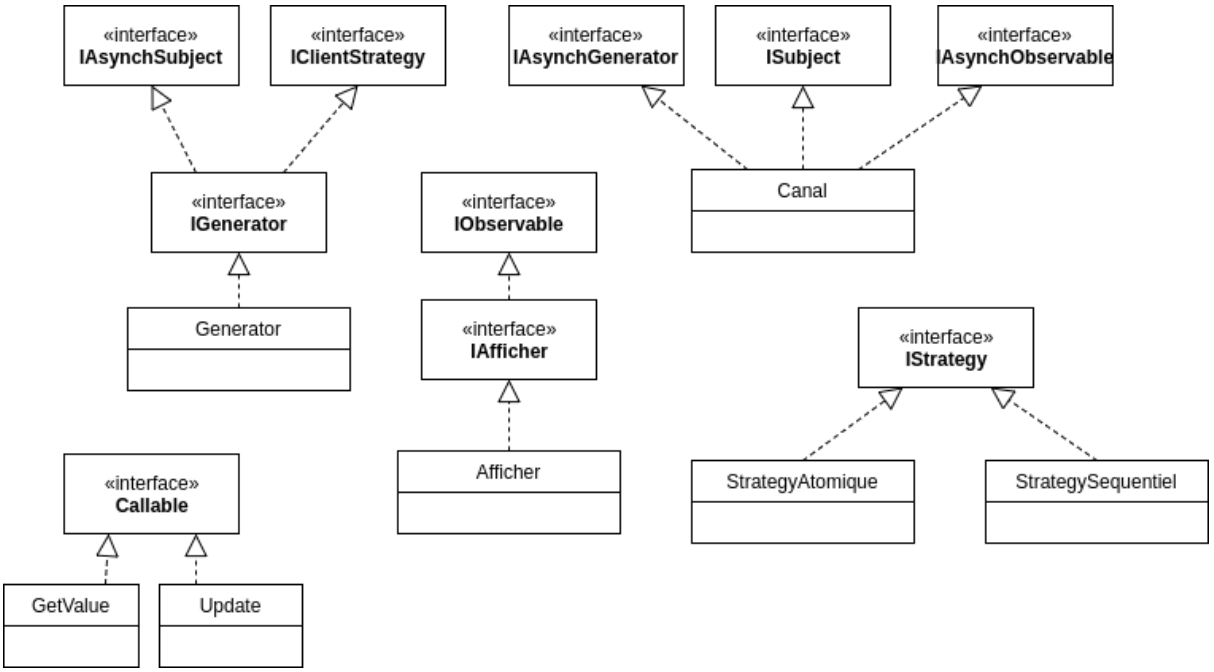
Le package activeObject contient les éléments permettant les appels de fonction de manière asynchrone et la méthode pouvant être lancée depuis de scheduler.

Diagramme de classe

M1



M3

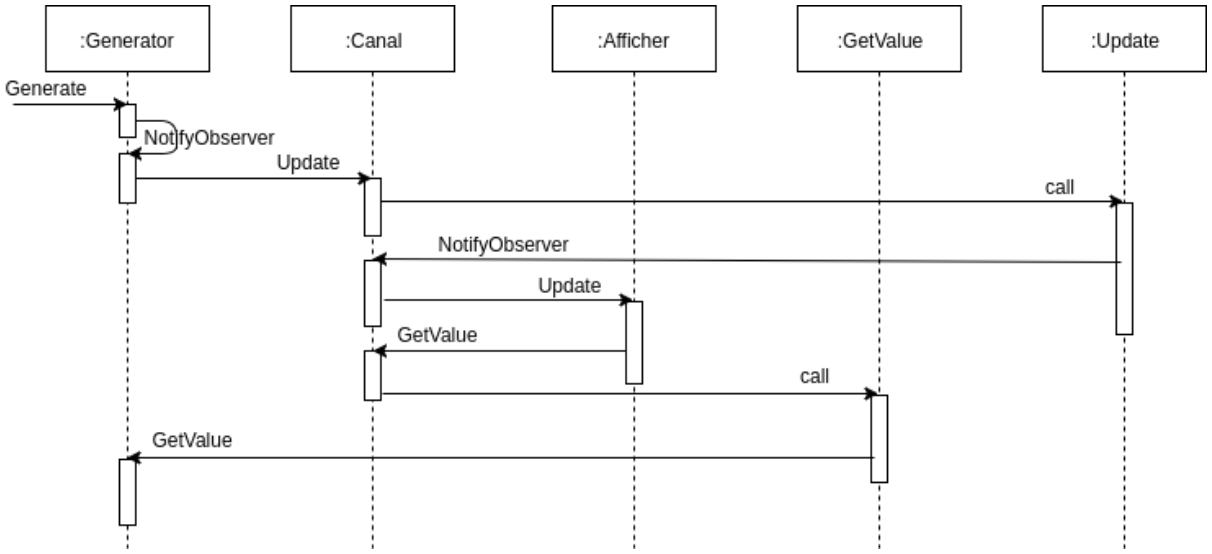


Principe de fonctionnement du programme

Le principe de fonctionnement est le suivant :

- Le Canal et le générateur discutent par la mise en place du patron de conception Observateur.
- La Canal et l'afficheur discutent par la mise en place du patron de conception Observateur.
- C'est le Canal qui crée la latence (simulation d'un dialogue réseau) par le biais des méthodes "update" et "getValue". La latence est créée concrètement par l'appel des objets Callable (Update et GetValue).

Diagramme de séquence



Pour une question de visibilité du diagramme de séquence nous n'avons pas fait apparaître la stratégie ainsi que l'ensemble des flèches de retour. Le principe reste le même, le générateur génère une valeur et notify les observateurs (ici le canal). Le canal programme l'update avec un latence (création de l'asynchrone) qui rappelle le notifyObsever pour informer les afficheurs de la nouvelle valeur. A la suite de cela, l'afficheur demande une nouvelle valeur (getValue) au canal qui programme (avec la latence) la demande au générateur. L''appel du getValue se fait par le biais du getValue Callable.

Problèmes rencontrés

Notre plus gros problème fut la compréhension globale du système. Dans un premier temps il a fallu bien comprendre la communication entre les éléments (Canal, Generator, Afficheur) une fois cette compréhension acquise (M2). L'étape suivante fut la mise en place de la latence et donc, des objects dit "runnable". Cette partie a été assez simple à mettre en place malgré notre non connaissance des outils. Nous avons eu une plus grosse difficulté à

comprendre les strategy, même si cela est annexe dans un premier temps. Nous avons cherché à implémenter une stratégie par affichage alors qu'elle était commune.

L’autre problème ne fut pas dans l’implémentation en elle même, mais dans la création de l’interface. En effet, les interfaces java restent pour nous un peu une découverte étant données que nous ne les utilisons peu, voir pas du tout.

Pour ma part (Victor) j’ai aussi un peu de mal a coder étant donner que je ne connais et utilise que très peu le Java je ne suis pas habitué à la syntaxe et aux objects.