DESIGN PATTERN PROXY(Procuration)

1. Présentation du design pattern Proxy

Patrons de structure

Les structural patterns sont des modèles prêts à l’emploi pour les relations entre les classes. L’objectif est de parvenir à une abstraction qui puisse également communiquer avec d’autres approches : programmation d’interfaces par mots-clés.

Les patrons structurels vous guident pour assembler des objets et des classes en de plus grandes structures tout en gardant celles-ci flexibles et efficaces.

 Il en existe 7 et ils permettent de définir comment organiser nos objets. :

* [Adapter](https://anceret-matthieu.fr/2019/03/les-design-patterns-structural-3/4/#adapter)
* [Bridge](https://anceret-matthieu.fr/2019/03/les-design-patterns-structural-3/4/#bridge)
* [Composite](https://anceret-matthieu.fr/2019/03/les-design-patterns-structural-3/4/#composite)
* [Decorator](https://anceret-matthieu.fr/2019/03/les-design-patterns-structural-3/4/#decorator)
* [Façade](https://anceret-matthieu.fr/2019/03/les-design-patterns-structural-3/4/#fa%C3%A7ade)
* [Flyweight](https://anceret-matthieu.fr/2019/03/les-design-patterns-structural-3/4/#flyweight)
* [Proxy](https://anceret-matthieu.fr/2019/03/les-design-patterns-structural-3/4/#proxy)

Un proxy est une classe utilisée à la place d’une autre classe. Contrairement au pattern Adapter, le pattern Proxy va proposer une classe de substitution lorsque la classe réelle n’est pas directement utilisable. Proxy peut donc affecter le comportement, mais pas l’interface. C’est pour cela que la classe réelle va elle aussi implémenter l’interface IProxy.

1. Utilités et cas d’utilisation

Ce pattern est particulièrement utilisé dans les cas suivants :

* Accès à des objets distants (base de données, disque dur réseau…) et donc potentiellement inaccessibles.
* Accès à des objets volumineux ou consommateur en ressources et devant être manipulé avec précaution.
* Accès à des objets nécessitant des droits d’accès spécifiques.

Proxy va donc permettre de gérer correctement le cycle de vie d’un service sans que le client ait besoin de s’en soucier.

Une illustration de la vie courante est le chèque (c’est valable aussi pour la carte bancaire) ! Un chèque et de l’espèce ont le même pouvoir (la même interface) de permettre de payer une tierce personne. Mais un chèque est un proxy pour une (grande) quantité d’espèces. C’est pratique pour le client qui peut éviter de se balader avec beaucoup d’argent liquide et c’est aussi pratique pour le vendeur qui peut facilement et à tout moment convertir son chèque en véritable argent liquide à la banque la plus proche.

Le Proxy Pattern permet de contrôler l'accès à un objet en fournissant un substitut ou un représentant de cet objet. Il peut être utilisé pour ajouter des fonctionnalités supplémentaires, telles que la mise en cache, la gestion des autorisations ou la gestion des appels réseau, sans modifier l'interface de l'objet d'origine. Ce pattern est souvent sous-estimé car il est considéré comme plus complexe à mettre en œuvre que d'autres patterns. Cependant, il offre une grande flexibilité et peut améliorer considérablement la modularité et la maintenabilité du code.

Un proxy est une classe se substituant à une autre classe. Par convention et simplicité, le proxy implémente la même interface que la classe à laquelle il se substitue. L'utilisation de ce proxy ajoute une indirection à l'utilisation de la classe à substituer. Le proxy sert à gérer l'accès à un objet, il agit comme un intermédiaire entre la classe utilisatrice et l'objet.