Design patterns et architectures logicielles

Étude de cas : le design pattern *State*

Filière : SUD-Cloud & IoT

Travail réalisé par :

- Chrif El Asri Hanane
- El Gamous Khalid

Encadrant:

Prof. Abdeslam EN-NOUAARY

2020/2021

Introduction

État est un patron de conception comportemental qui permet de modifier le comportement d'un objet lorsque son état interne change. L'objet donne l'impression qu'il change de classe. Lors de cette étude de cas on va comprendre comment il fonctionne.

I. Problématique

Le principe repose sur le fait qu'un programme possède un nombre *fini* d'*états*. Le programme se comporte différemment selon son état et peut en changer instantanément. En revanche, selon l'état dans lequel il se trouve, certains états ne lui sont pas accessibles. Ces règles de changement d'état sont appelées *transitions*. Elles sont également finies et prédéterminées.

On peut appliquer cette approche aux objets. Imaginons une interface graphique formée de trois boutons (push, pull et exit) et un canevas et dont le comportement change en fonction de l'état courant du canevas et du bouton appuyé par l'utilisateur (voir la figure ci-après). La couleur du canevas est par défaut rouge et la transition entre les couleurs se fait comme suit:

- Bouton push : rouge →bleu → vert → noir → rouge.
- Bouton pull : rouge → vert → bleu → noir → rouge.

II. Solution triviale

La solution triviale est de rédiger un code qui traite tous les cas possibles avec if ou switch. Si on appuie sur le bouton push alors la fonctionnalité va dépendre de l'état actuel du canevas ; s'il est rouge donc il va changer sa couleur pour devenir bleu, s'il est par contre vert il va changer en noir et ainsi de suite.

Cette solution nous fournit la fonctionnalité demandée. En revanche, plus le programme devient complexe et contient plus de conditions et de possibilités plus cela devient très difficile à maintenir. En plus, un code avec de multiples conditions if n'est pas une bonne idée au niveau de performance et de temps d'exécution. On essaie généralement et surtout dans le domaine du *machine learning* d'éviter autant que possible d'utiliser les opérateurs conditionnels (if ou switch).

III. Solution avec State

Le patron de conception état propose de créer de nouvelles classes pour tous les états possibles d'un objet et d'extraire les comportements liés aux états dans ces classes.

Plutôt que d'implémenter tous les comportements de lui-même, l'objet original que l'on nomme contexte, stocke une référence vers un des objets état qui représente son état actuel. Il délègue tout ce qui concerne la manipulation des états à cet objet.

Pour faire passer le contexte dans un autre état, on doit remplacer l'objet état par un autre qui représente son nouvel état. On ne peut le faire que si toutes les classes suivent la même interface et si le contexte utilise cette dernière pour manipuler ces objets.

Cette structure ressemble de près au patron de conception *Stratégie*, mais il y a une différence majeure. Dans le patron de conception état, les états ont de la visibilité entre eux et peuvent lancer les transitions d'un état à l'autre, alors que les stratégies ne peuvent pas se voir.

IV. Autre exemple

Les boutons de du smartphone fonctionnent différemment selon l'état de l'appareil :

- Si le téléphone est déverrouillé, appuyer sur des boutons lance différentes fonctionnalités.
- Si le téléphone est verrouillé, appuyer sur n'importe quel bouton envoie sur l'écran de déverrouillage.
- Si la batterie du téléphone est faible, appuyer sur n'importe quel bouton montre l'écran de charge.

Conclusion

On constate qu'il est préférable d'utiliser le patron de conception état lorsque le comportement de l'un des objets varie en fonction de son état, qu'il y a beaucoup d'états différents et que ce code change souvent. Aussi si l'une des classes est polluée par d'énormes blocs conditionnels qui modifient le comportement de la classe en fonction de la valeur de ses attributs.