Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2022 – 2023
ATL

ATL – Ateliers Logiciels

Patron d'architecture MVC

Introduction

> **Consignes**
> Ce document récapitule les notions de base concernant le *design pattern* (patron d'architecture) *Modèle-Vue-Contrôleur*.

1 Modèle-Vue-Contrôleur

Modèle-vue-contrôleur est une architecture divisée en trois parties :

- **le modèle** : contient la logique métier, elle traite les données de l'application ;
- **les vues** : présentation visuelle de l'état du modèle ;
- **les contrôleurs** : déclencheurs d'actions à effectuer sur le modèle et/ou sur une vue.

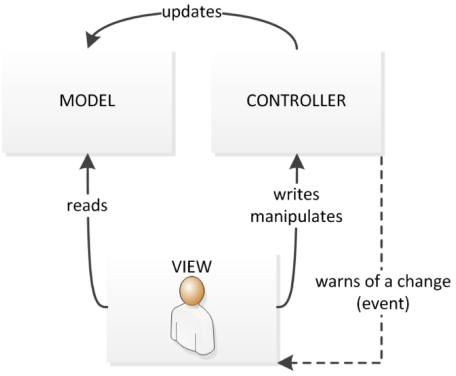
Ce découpage permet d'isoler les différentes parties du logiciel en suivant les principes de séparation des responsabilités (*Separation of Concerns* ou SOC en anglais). Des modifications apportée sur une partie n'auront, idéalement, aucune conséquence sur les autres.

1.1 Hiérarchie des interactions

Dans une architecture MVC, les interactions utilisateur passent toujours par une classe de **vue**.

La vue répond à toute interaction en la transmettant au **contrôleur**, qui traduit cette interaction en un appel de méthode du modèle après avoir éventuellement vérifié la validité de l'interaction.

Une fois que le **modèle** a été contacté par le contrôleur, les différentes classes de modèle se mettent à jour selon la logique métier du système (règles d'un jeu, logique d'une base de données, ...).
Une fois le modèle mis à jour, le modèle informe la vue de faire un nouvel affichage avec les nouvelles informations.



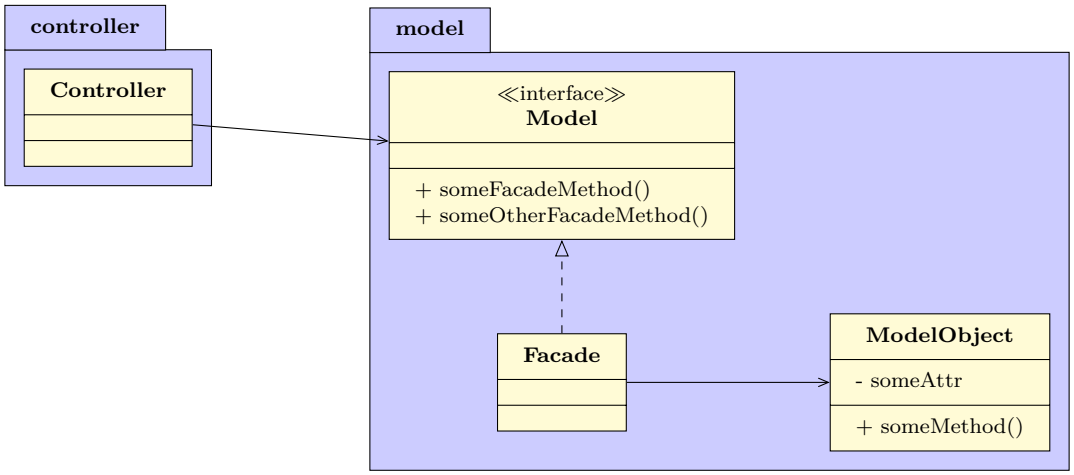
Version du document
17 octobre 2022

2 Encapsulation du modèle

Pour assurer une encapsulation maximale du modèle dans une application utilisant une architecture MVC, on utilisera le *design pattern* **Façade**.

Ce *design pattern* consiste en l'emploi d'une classe (appelée classe de façade) qui est le seul point d'accès au sein d'un package donné à partir de l'extérieur de ce package. Cette classe de façade définit donc l'ensemble des appels qui peuvent être faits au modèle, et les transmets aux autres classes de modèle selon les besoins.

Pour encore améliorer l'encapsulation, on dote souvent la façade d'une *interface* ; cette interface est le type déclaré de la façade du point de vue du contrôleur, et les méthodes d'interface sont dès lors les seules méthodes auxquelles le contrôleur peut accéder.



3 MVC avec Observateur-Observé

Pour plus d'informations, consultez la fiche sur le design pattern Observateur-Observé.

Le *design pattern* **Observateur-Observé** a pour objectif d'automatiser la mise à jours d'éléments d'une application lorsque d'autres éléments de l'application ont reçu des modifications.

Dans le cas de l'architecture MVC, ce patron de conception permet donc d'automatiser la mise à jour de la vue une fois que le modèle à terminé ses modifications. Certaines classes de vue deviennent dès lors **Observateurs**, alors que certaines classes du modèle deviennent **Observées**.

Comme toujours, les classes observateurs s'enregistrent auprès de leurs objets observés à leur création. Par la suite, les objets observés notifient leurs observateurs lorsqu'un changement de leur état se produit. Du moment que le *design pattern* est correctement implémenté, *cela n'induira pas de problèmes d'encapsulation.*

Un conseil : choisissez judicieusement vos observateurs ! S'il est possible de séparer des composants de vue et de leur faire chacun observer un élément spécifique du modèle, cela simpifiera grandement les échanges via Observateur-Observé et vous permettra de respecter plus facilement l'encapsulation des données.

3.1 Diagramme de classes