

# MVC

*Cette présentation a pour but de vous faire découvrir les aspects importants et fortement utilisés du design pattern MVC.*

## 1 INTRODUCTION

---

MVC est un design pattern (modèle de conception ou DP) de conception d'interface utilisateur permettant de découpler le modèle (logique métier et accès aux données) des vues (interfaces utilisateur [présentation des données et interface de saisie pour l'utilisateur]).

'Découpler' les vues du modèle signifie que l'on puisse les faire communiquer alors que l'on a minimisé la connaissance qu'ils ont l'un de l'autre. Cela entraîne que des modifications de l'un n'auront, idéalement, aucune conséquence sur l'autre ce qui facilitera grandement la maintenance des applications.

Attention, ce DP peut être appliqué de manières fort différentes mais nous en abordons le principe avant de voir des exemples simples d'application.

## 2 PRINCIPE

---

- **Modèle:** gère les données et reprend la logique métier (le modèle lui-même peut être décomposé en plusieurs couches mais cette décomposition n'intervient pas au niveau de MVC). **Le modèle ne prend en compte aucun élément de présentation!**
- **Vue:** Elle a pour rôle d'afficher à l'utilisateur les données provenant du modèle. Elle peut aussi recevoir les actions de l'utilisateur. **Aucun traitement – autre que la gestion de présentation - n'y est réalisé.**
- **Contrôleur:** son rôle est de favoriser le découplage entre la vue et le modèle: il traite les événements en provenance de l'interface utilisateur (vue) et

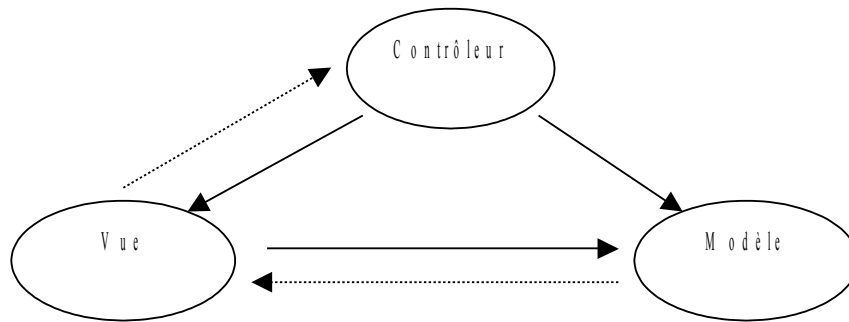
soit il les transmet au modèle pour le faire évoluer

soit il les transmet à la vue pour lui faire modifier son aspect visuel (pas de modification des données affichées mais des modifications de présentation (couleur de fond, affichage ou non de la légende d'un graphique, ... )).

Le Contrôleur 'connaît' la (les) vues qu'il contrôle ainsi que le modèle. Il pourra appeler des méthodes du modèle pour réagir à des événements (demande d'ajout d'un client p.ex.), il pourra faire modifier à la vue son aspect visuel. Pour cela, le contrôleur sera à l'écoute d'événements survenant sur les vues.

La vue observera le modèle qui l'avertira du fait qu'une modification est survenue. Dans ce cas, la vue interrogera le modèle pour obtenir son nouvel 'état' et procédera à la mise à jour de l'affichage.

Nous pouvons schématiser cela comme suit en percevant bien que nous pouvons avoir plusieurs contrôleurs, plusieurs vues mais, dans un premier temps, un seul modèle.



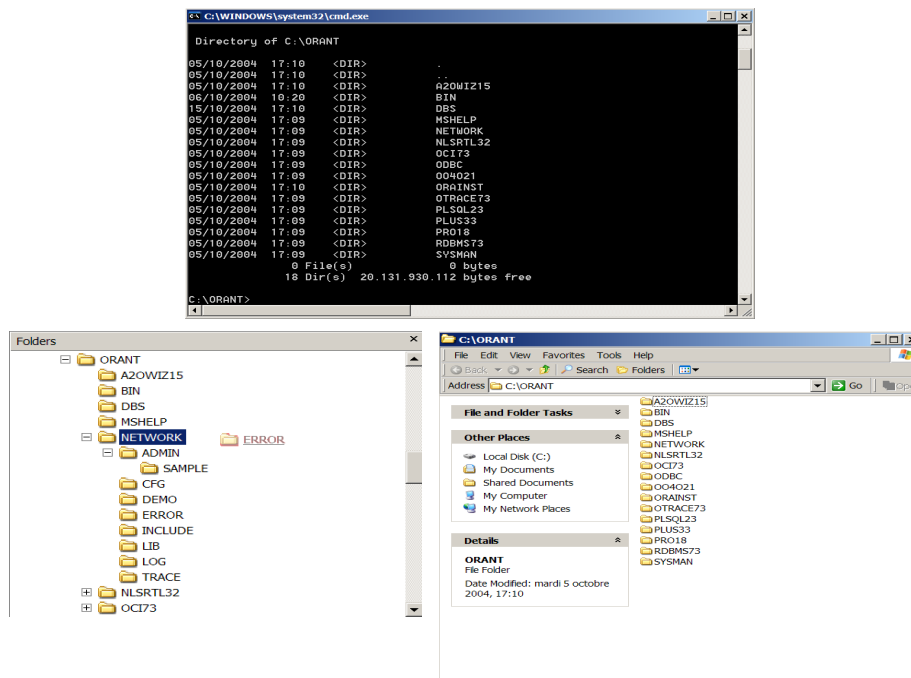
Les traits pointillés représentent une écoute (sens écouté – écouteur).

Les traits pleins représentent des appels de méthodes.

### 3 UN EXEMPLE PRATIQUÉ QUOTIDIENEMENT

Plusieurs vues différentes nous sont offertes par un SE sur son File System. Un contrôleur pourra recevoir au travers de l'interface utilisateur

- les événements de mise à jour du modèle (par exemple: demande de l'utilisateur de la suppression d'un fichier, d'ajout d'un répertoire, ...) et le transmettra au modèle. Les vues observatrices du modèle seront prévenues de la modification du modèle et l'interrogeront pour mettre à jour leur affichage.
- les événements de modification de présentation des données (par exemple: demande par l'utilisateur de la présentation détaillée - plutôt que par icône - de la liste des fichiers et des répertoires) seront transmis à la vue qui modifiera sa présentation en tenant compte de la demande.



Il est à remarquer que si différents File System (cf. NTFS et FAT) peuvent coexister, il suffit qu'ils présentent le même interface pour que les vues et les contrôleurs soient réutilisables.

Remarquons, comme dans les images ci-dessus, que nous pouvons avoir plusieurs vues abonnées simultanément au même modèle.

Si les vues doivent refléter en « temps réel » les changements d'état du modèle, elles devront être abonnées (par le contrôleur) comme observateur (écouteur) du modèle.

## 4 MISE EN OEUVRE EN JAVA (TD8)

---

Nous allons nous contenter, pour l'exemple de réaliser une application d'addition de deux naturels mettant en oeuvre MVC.

Le modèle (très simple dans ce cas-ci), devra offrir une interface permettant aux vues de l'observer (inscription et désinscription de la liste des observateurs et méthode d'obtention de l'état du modèle). Il aura aussi à présenter au(x) contrôleur(s) une interface permettant d'agir sur lui (demande de calcul).

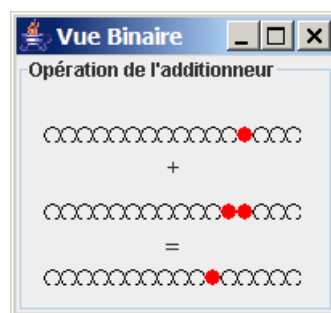
Les vues devront pouvoir s'inscrire et se désinscrire auprès du modèle et lui présenter une interface permettant de leur notifier les évolutions du modèle.

Par simplification, nous décidons que chacune des vues héritera de JComponent.

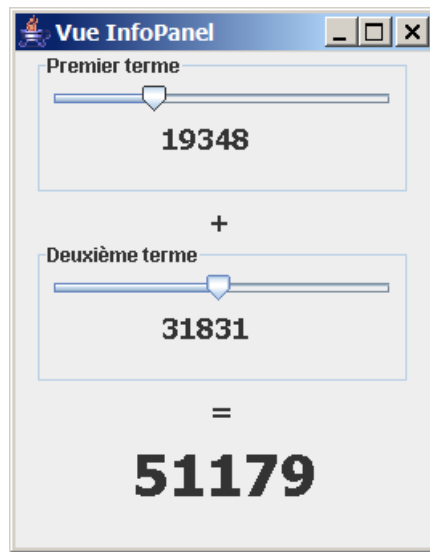
Le terme « contrôleur » est utilisé dans cette présentation pour parler d'une multitude d'objets (les contrôleurs) réagissant à des événements. En pratique, dans l'exemple que nous allons présenter, les contrôleurs seront des classes anonymes à l'écoute de vues ou de partie de vues (boutons, textFields, comboBox, items de menu ...).

Le code l'exemple vous est donné en annexe dans un projet NetBeans. Il vous est demandé de le lire, le comprendre, identifier les contrôleurs. De plus, vous aurez à concevoir deux vues supplémentaires, affichables à la demande :

a) une vue présentant sous forme binaire, les termes et le résultat de l'addition. L'affichage binaire d'un naturel se fera au travers d'un composant NaturelBinaire présentant 16 leds. Attention, si un des naturels à présenter dépasse la capacité d'affichage, le composant correspondant devra faire clignoter l'ensemble de ses leds!



b) une vue présentant les termes de l'addition et permettant de les saisir sous forme d'InfoPanel et présentant le résultat sous forme de JLabel. Attention, contrairement à la vue permettant la saisie qui vous est fournie dans l'exemple, celle-ci devra écouter le modèle pour mettre à jour les termes et le résultat de l'addition courante du modèle.



Veillez à gérer de manière cohérente les exceptions lancées par le modèle et les éventuelles exceptions lancées par les InfoPanel.