ARCHITECTURE MVC

Plan

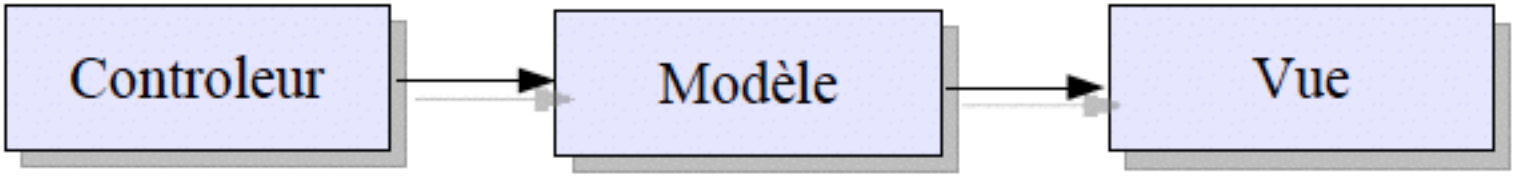
- Motivation
- MVC
  - Définition
  - Architecture (Model - Vue - Controleur)
  - Avantages - Inconvénients
  - MVC et patrons de conception
- Variantes-Dérivés du MVC
  - MVC2
  - MVP
  - MVVM

Historique

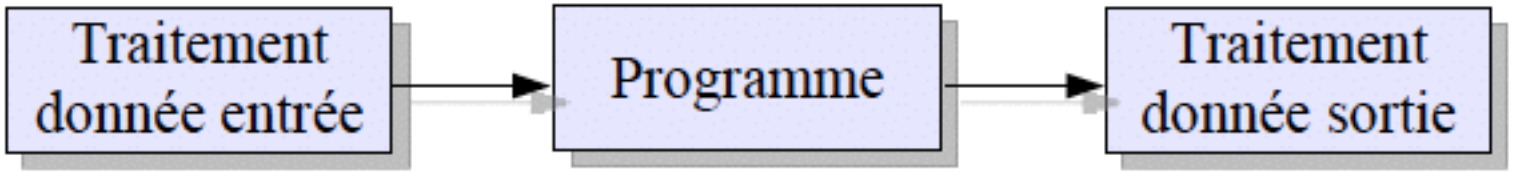
- L'idée de séparer l'interface utilisateur du reste du code est apparu à la fin des années 1970, lorsque la principale modification d'interface commençait a devenir incontournable:
  - Passage de l'interface textuelle à une interface « graphique »
  - Comment continuer à faire fonctionner les logiciels existants en utilisant ce nouveau type d'affichage?
- En 1979 Trygve Reenskaug (équipe Smalltalk), proposa une solution pour répondre au problème:
  - Met en évidence le problème d'interdépendance des données ainsi que le besoin de consulter les données
  - Initialement appelée MVE (Model View Editor) puis MVC (Model View Controller)

Motivation: Pourquoi le MVC ?

- Décomposer une interface graphique en trois parties distinctes



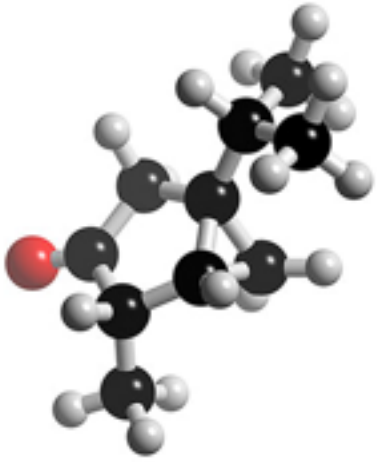
- Schéma calqué sur le traitement classique d'un programme



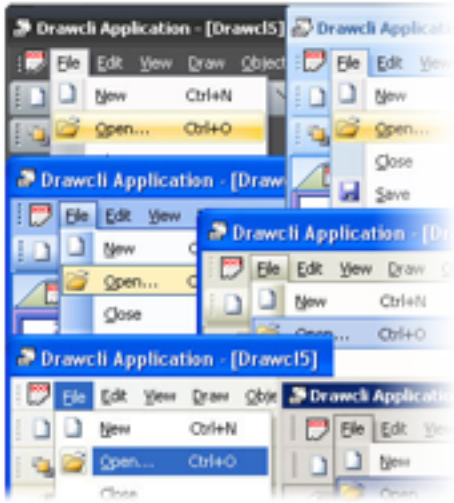
MVC

- Le Modèle-Vue-Contrôleur (MVC) est un patron de conception très utilisé dans la conception des IHM, en particulier les sites et applications Web.
- Souvent présenté comme patron d'architecture
- Il présente un grand intérêt dans le travail en équipe et la répartition des tâches.
- Il permet de mieux structurer son code source et de le séparer en trois parties distinctes.

Model:



View:



Controller:

Principe général

- But
  - Isoler la donnée elle-même de sa présentation
  - Distinguer la consultation de la modification
  - Couplage le plus faible possible
- Principe
  - Separate Of Concern
  - Les 3 composantes suivantes d'une donnée sont distinguées et séparées
    - Le modèle (sa structure)
    - La vue (sa représentation pour affichage)
    - Le contrôleur (les moyens de modifier la valeur)

Le Modèle (Model)

- Le **Modèle** est un ensemble de composants contenant les données et les opérations logiques permettant la manipulation de ces données.
  - Représente le comportement de l'application
  - Les données métiers ainsi que la communication vers la structure de données (SGBDR, NoSQL, XML,…).
- Rôle:
  - Encapsuler les données de l'application
  - Effectuer les traitements sur ces données
  - Etre indépendant des vues et contrôleurs
- Conséquences
  - L'interface du modèle propose
    - La mise à jour des données
    - La consultation des données
  - Résultats du modèle dénués de toute présentation
  - Maintenir une liste d'écouteurs (vues et/ou contrôleurs se déclarent comme écouteurs)
  - Prévenir les écouteurs lorsque la donnée est modifiée
  - Implantation du design pattern Observer: Model, est Observable

La Vue (View)

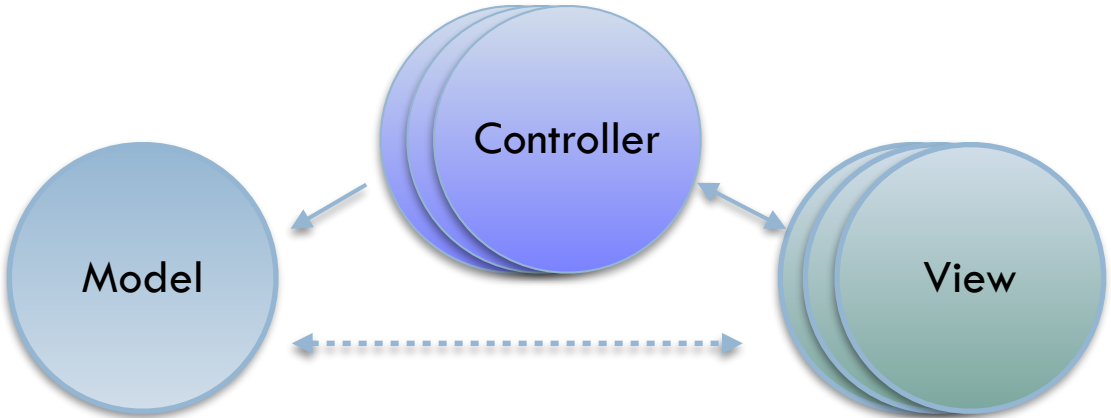
- La vue est une représentation visuelle des données de l'application (UI)
- Rôle
  - Présenter les données du modèle
  - Se maintenir à jour lorsque le modèle est modifié
  - Récupérer tous les évènements de l'utilisateur.
  - Transmettre au Controller les évènements utilisateur.
- Conséquences
  - Plusieurs vues possibles pour les mêmes données
  - Doit s'enregistrer comme écouteur au niveau du modèle (selon l'implémentation)
  - La vue n'effectue aucun traitement

Le Contrôleur (Controller)

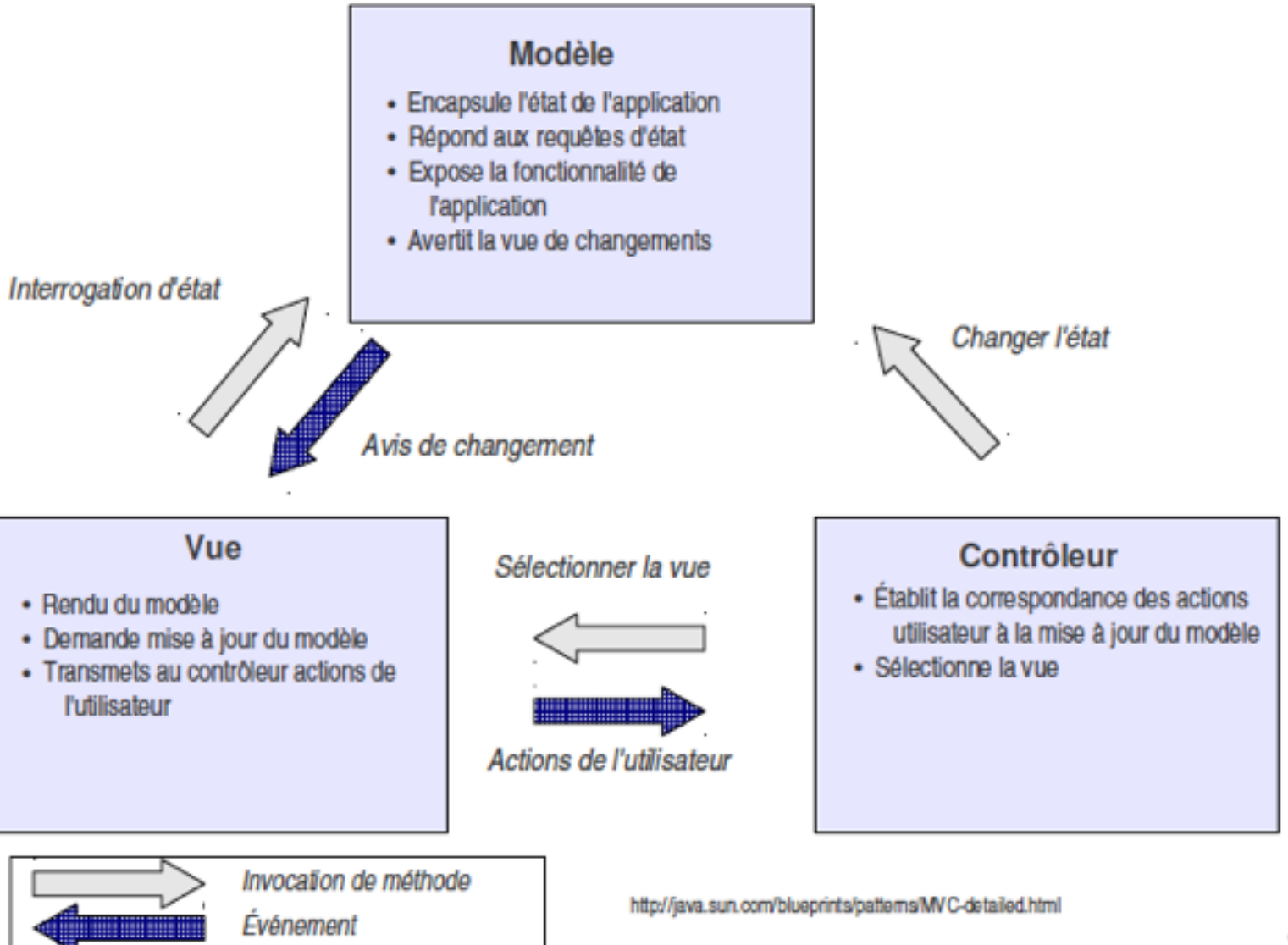
- Le Contrôleur, reçoit les évènements envoyés par la Vue, et lors d'une demande de modification demande au modèle de synchroniser la nouvelle valeur
- Rôle
  - Permettre à l'utilisateur de modifier la donnée encapsulée dans le modèle
  - Reçoit les événements de l'utilisateur
    - Par l'intermédiaire de la vue
    - Il analyse la requête et applique un traitement de contrôle
  - Déclenche les actions à effectuer
    - Il transmet des messages au modèle
    - Si nécessaire, il informe la vue de se mettre à jour (Gestion des événements de synchronisation)
- Conséquences
  - Doit éventuellement s'enregistrer comme écouteur du modèle pour être mis à jour si le modèle est modifié (selon l'implémentation)
  - Doit appeler les accesseurs du modèle en fonction des actions de l'utilisateur

Caractéristiques

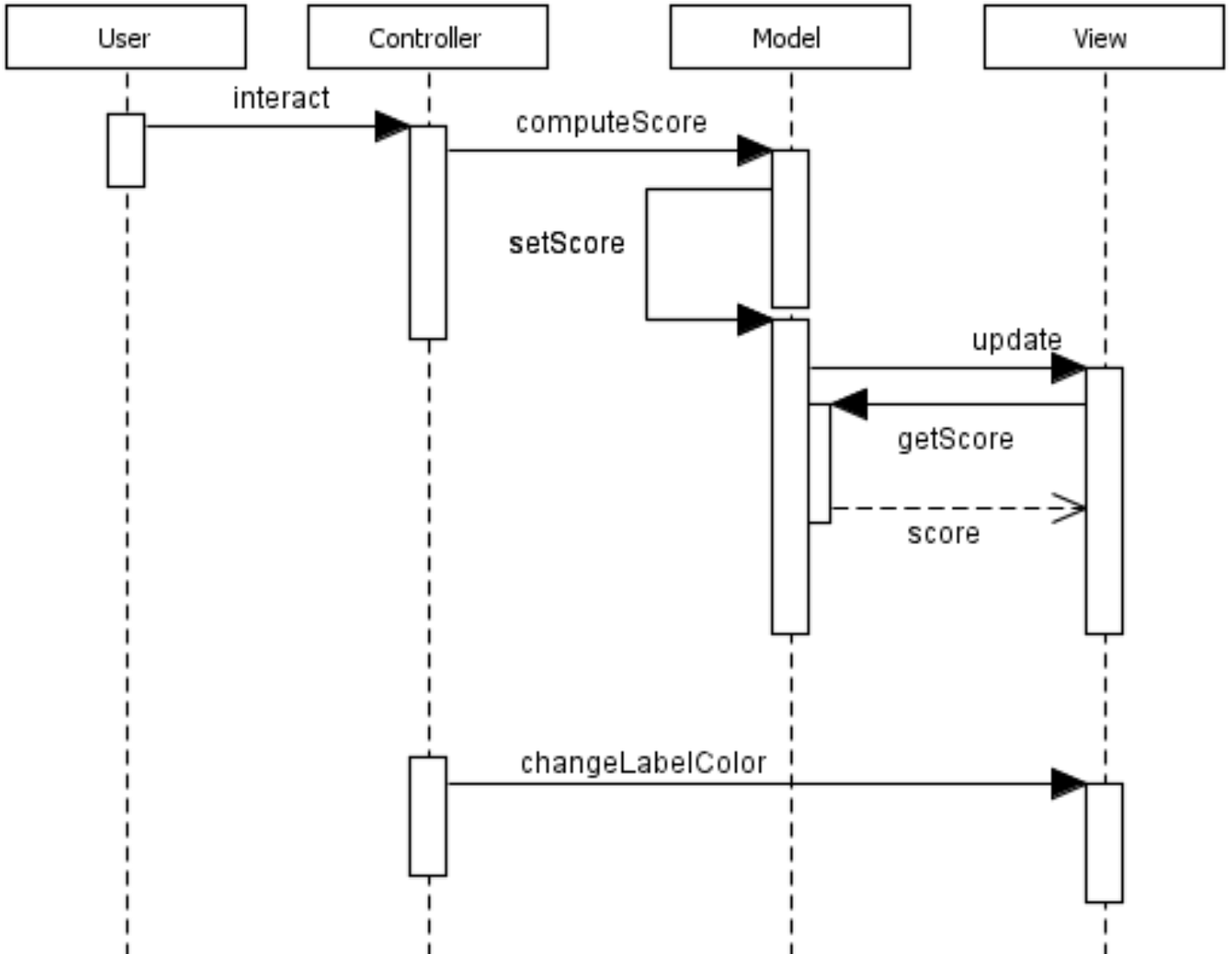
- Les classes du niveau Modèle devraient encapsuler complètement l'information et le comportement liés à la logique d'application
- Les classes de Vue sont responsables des entrées et des sorties, mais ne contiennent pas de données ou de fonctionnalités liées à l'application
- Un modèle peut avoir plusieurs vues et contrôleur
- Un contrôleur peut être associé à plusieurs vues, mais une vue n'ai associée qu'à un seul contrôleur



Architecture MVC



MVC: Exemple de Collaboration



Avantages du MVC (1/2)

- Evite le stockage des données à plusieurs endroits.
- Propose une définition cohérente du modèle, axé sur les processus issus du domaine, et non axés sur les interfaces hommes machines,
- Maintenance facilitée par le découplage vue(s)-modèle
- Possibilité pour le modèle d'informer la vue des modifications incrémentalement (par exemple, ajout d'une ligne dans une table).

Avantages du MVC (2/2)

- Modularité
  - Séparation claire entre les données du programme et la partie graphique affichant les données.
  - Permet une séparation des tâches de développement (une personne peut s'occuper des vues, une autre du modèle, etc.).
  - Permet l'exécution du modèle de façon indépendante de l'interface
- Extensibilité
  - Permet l'ajout de nouvelles vues sans modifications du modèle.
  - Garantit une plus grande portabilité en facilitant la migration du modèle vers un nouveau type d'environnement de gestion d'interface
  - Minimise l'impact de changements dans les spécifications liées à l'interface sur le niveau de la logique d'application

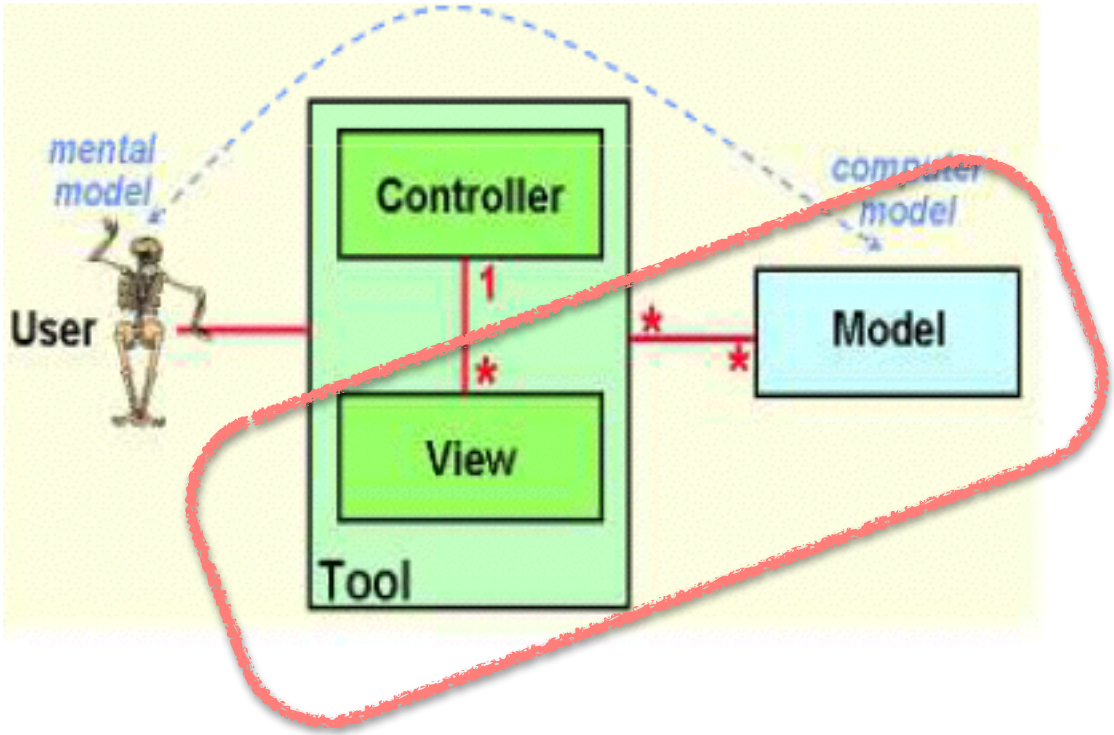
Inconvénients

- L'inconvénient majeur du modèle MVC est qu'il n'est pas adapté pour des projets de petite envergure.
  - Pour respecter la logique de séparation des composant, il faut créer un fichier/répertoire pour chacun de ces derniers, augmentant ainsi de manière inutile la taille de l'application.
- Coût des communications des composants
- Limite potentiellement floue entre les composants

MVC et Patrons GoF

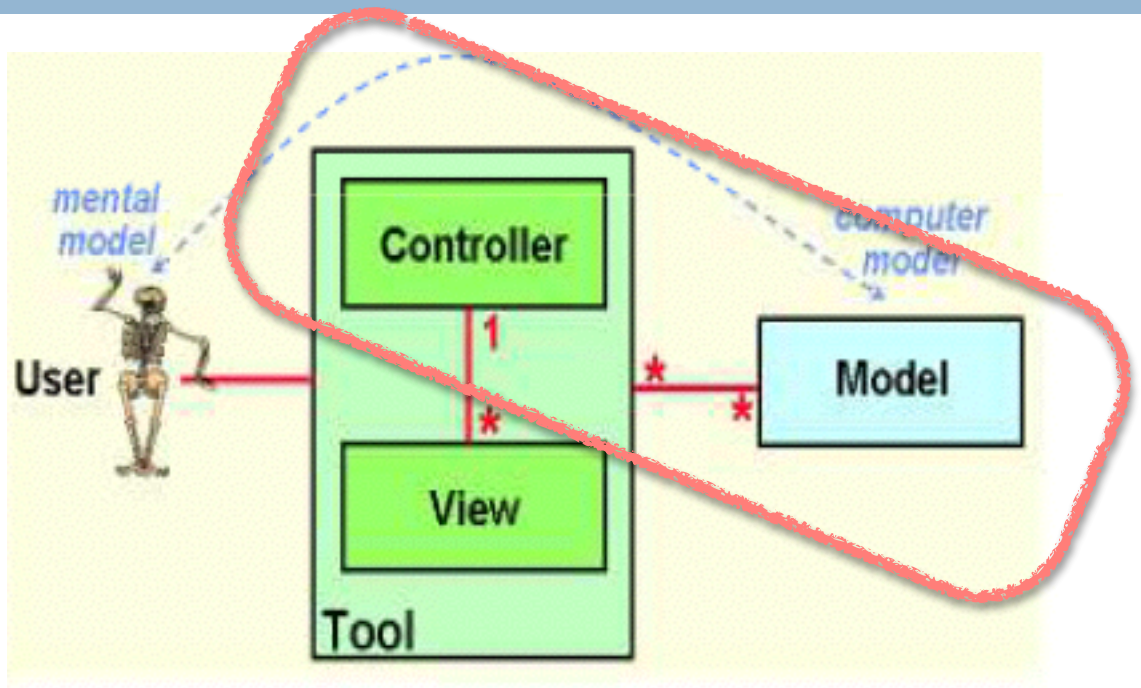
- La communication entre le Modèle et la Vue du modèle MVC est un exemple de nombreuses situations nécessitant une communication indirecte entre des entités d'un système.
- Ce type de communication nécessite l'introduction de Patrons spécialisés : Patron Observer, Patron Mediator.
- D'autres patrons sont aussi utilisés pour les interfaces graphiques: Stratégie, composite, decorator, etc.

MVC -Observer (Vue-Modèle)



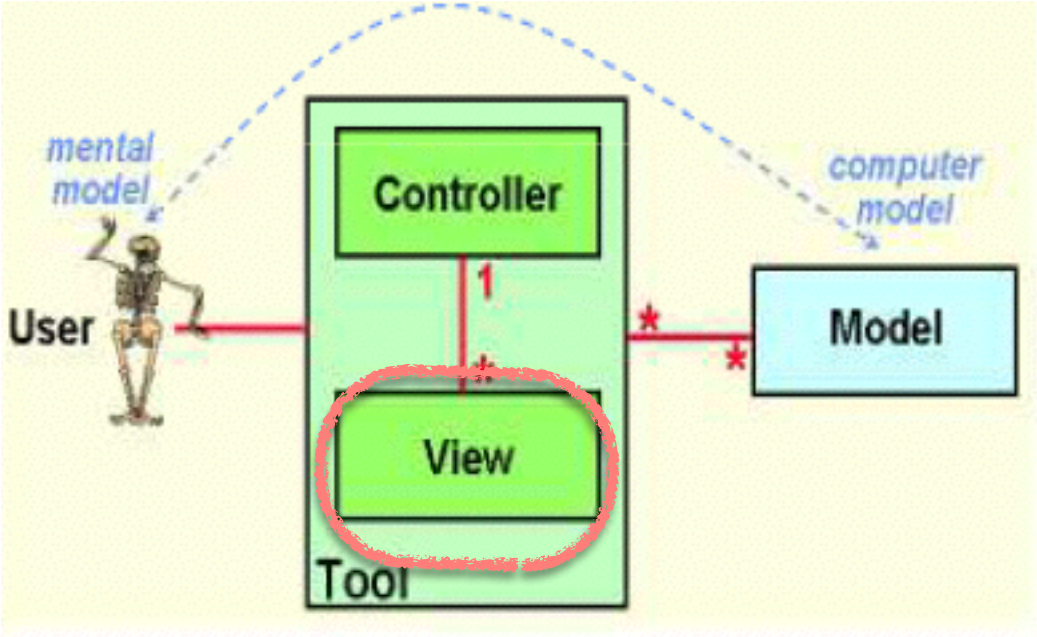
- L'application doit pouvoir gérer plusieurs vues simultanément.
- Dès que le modèle est modifié, les vues se redessinent de manière adéquate.

MVC-Observer (Contrôleur-Modèle)



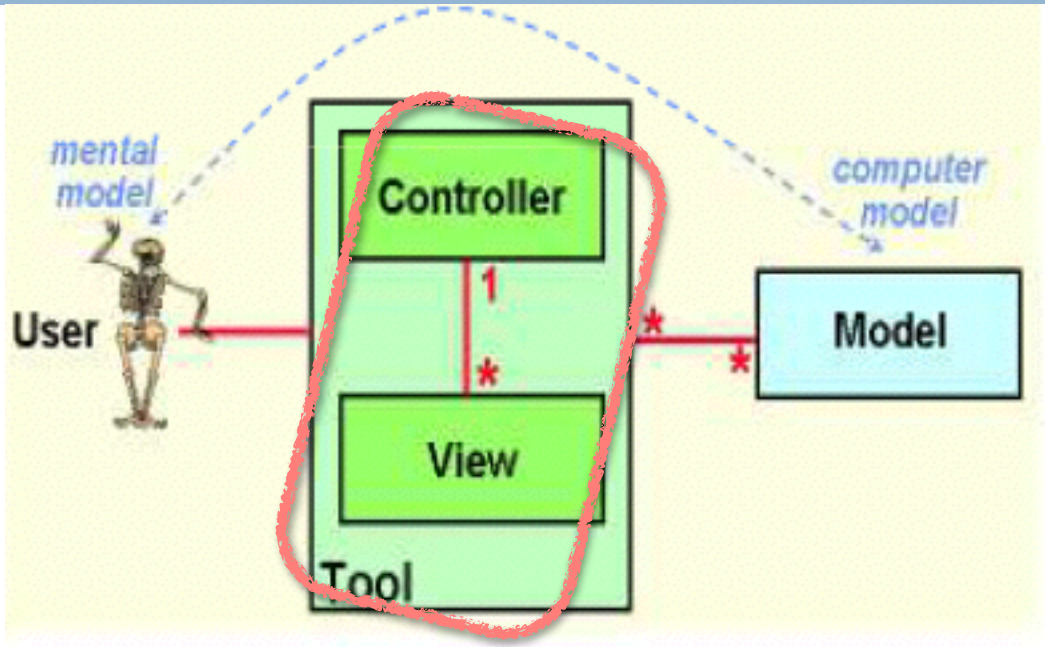
- Il peut arriver que l'état du modèle ait un impact sur les contrôleurs. Par exemple, il se peut qu'en fonction du modèle, certaines options ne soient pas disponibles (menus, cases à cocher...).
- Les contrôleurs doivent donc être avertis de toutes les modifications du modèle.

MVC-Composite (Vue-Vue)



- On considère que chaque composant peut faire l'objet d'une vue et que la vue en question peut contenir des sous-vues représentant les sous-composants

MVC-Strategy (Contrôleur-Vue)

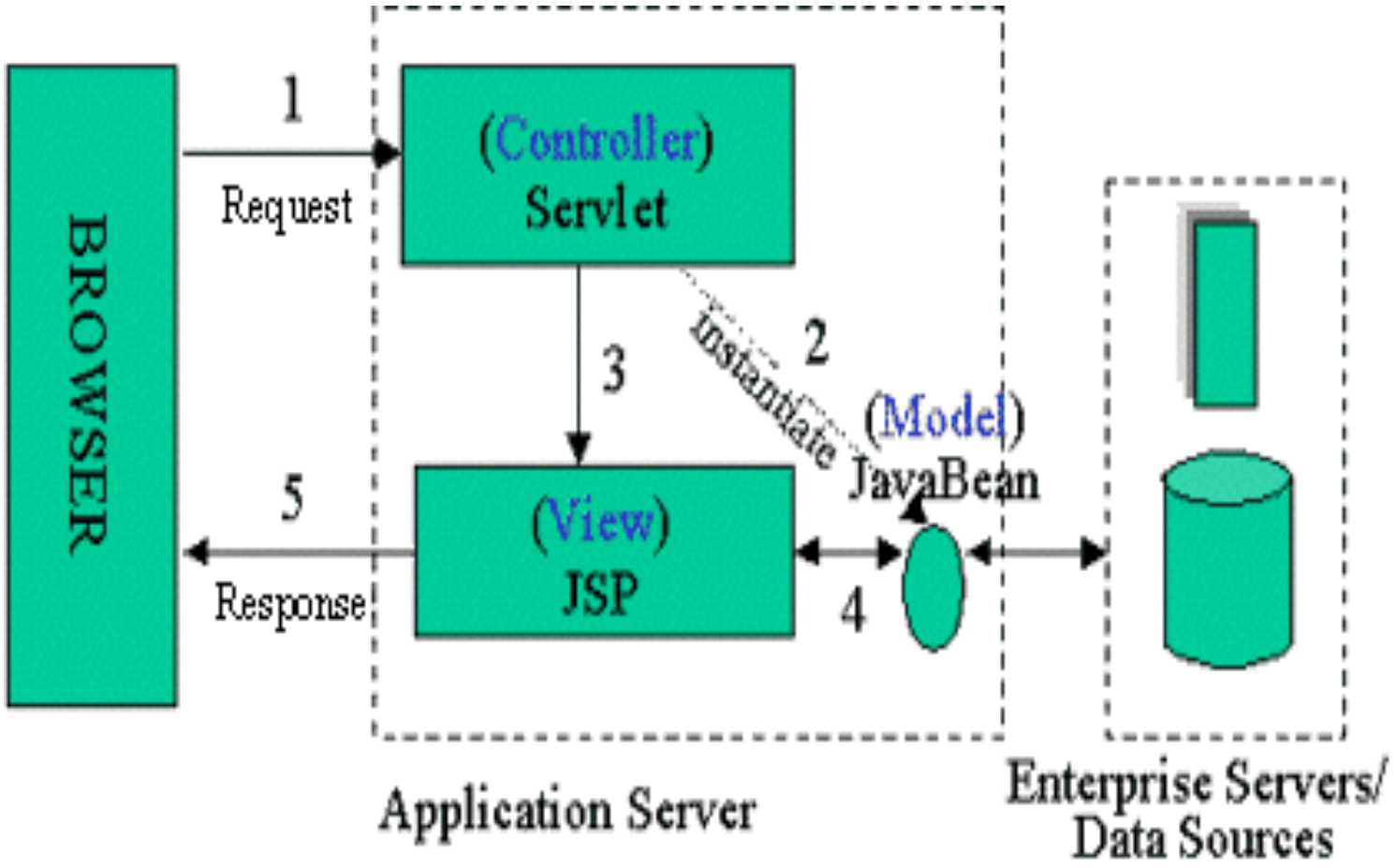


- La manière dont un contrôleur agit sur le modèle dépend de la vue à laquelle il est associé (stratégie)
- Les contrôleurs doivent donc être avertis de toutes les modifications du modèle.

Bonnes pratiques à appliquer

- L'erreur la plus courante est de donner trop d'importance au contrôleur
  - Notez : il reçoit une requête, applique un traitement de contrôle, délègue les opérations et retourne une réponse.
- Quelques conseils
  - Toutes les opérations de validations des données d'un utilisateur sont à externaliser dans un service de validation dédié.
  - Les fonctions d'aides et utilitaires à externaliser aussi.
  - Ne pas mettre du code HTML directement dans le contrôleur (même dans la réponse). Retournez une vue le cas échéant.
  - Le traitement des données métiers est bien sûr à effectuer dans le modèle.
  - Créer autant de contrôleurs que de responsabilité (Authentification, Utilisateur,…)
  - Créer autant de modèles que d'entité de votre base de données (Langue, Article, Catégorie…)

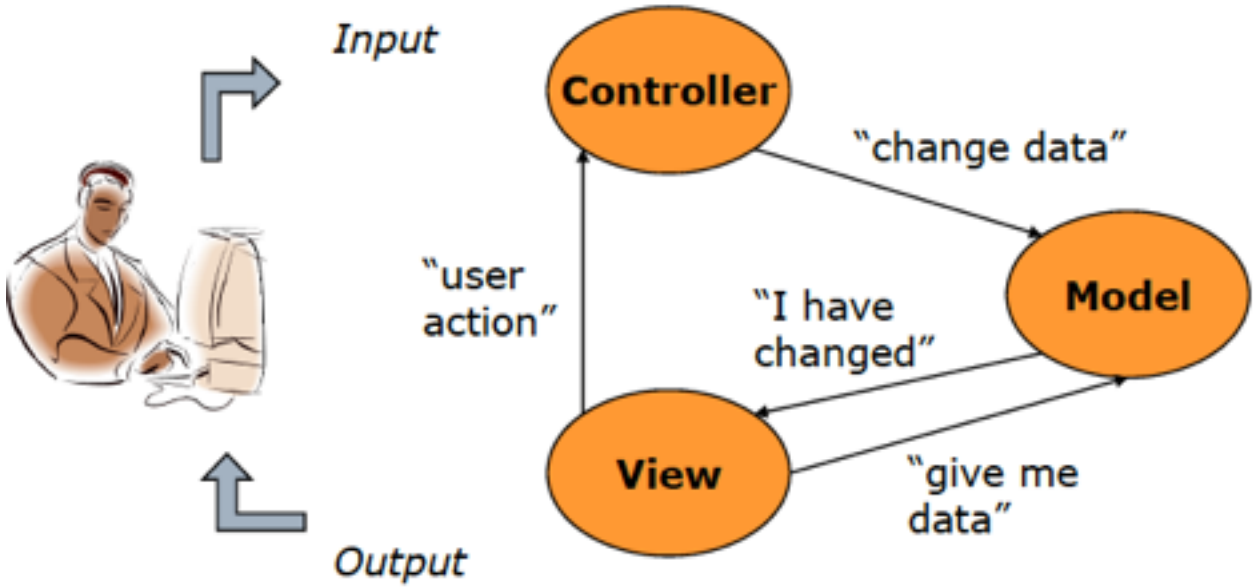
MVC Exemple


BROWSER
1
Request
(Controller)
Servlet
2
instantiate
3
(Model)
JavaBean
5
Response
(View)
JSP
4
Application Server
Enterprise Servers/
Data Sources

Variantes du MVC

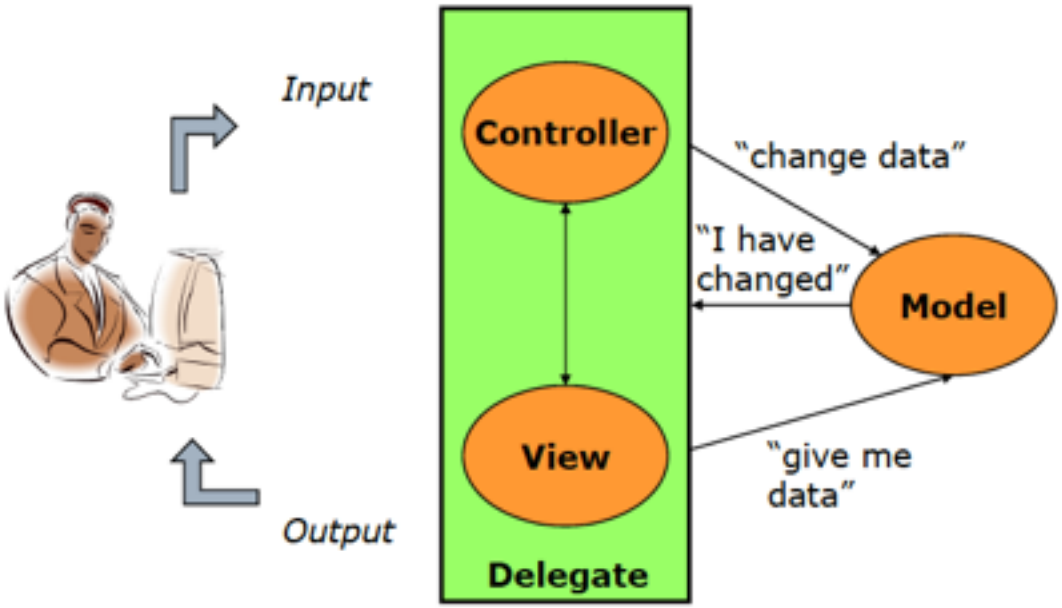
1. Supervising Controller

- Le contrôleur communique avec le modèle et la vue.
- Le modèle et la vue peuvent communiquer directement (vue est observateur du modèle)
- Le modèle notifie ses vues et contrôleurs associés quand un changement d'état survient.
- La vue demande les données du modèle pour mettre à jour la représentation à l'utilisateur
- Le patron MVC original utilise cette variante d'implementation.



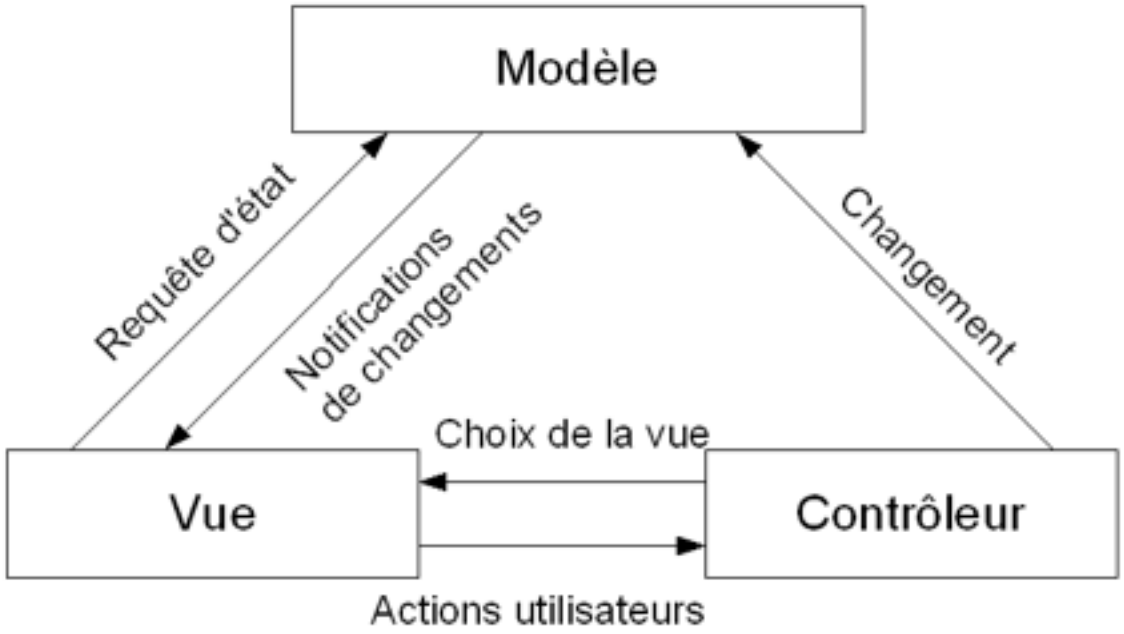
2. Delegate Model

- Le modèle est le même que MVC original
- Delegate est responsable des entrée et sortie: c'est une combinaison de la vue et du contrôleur
- Appelé aussi : UI-Model, Document-View



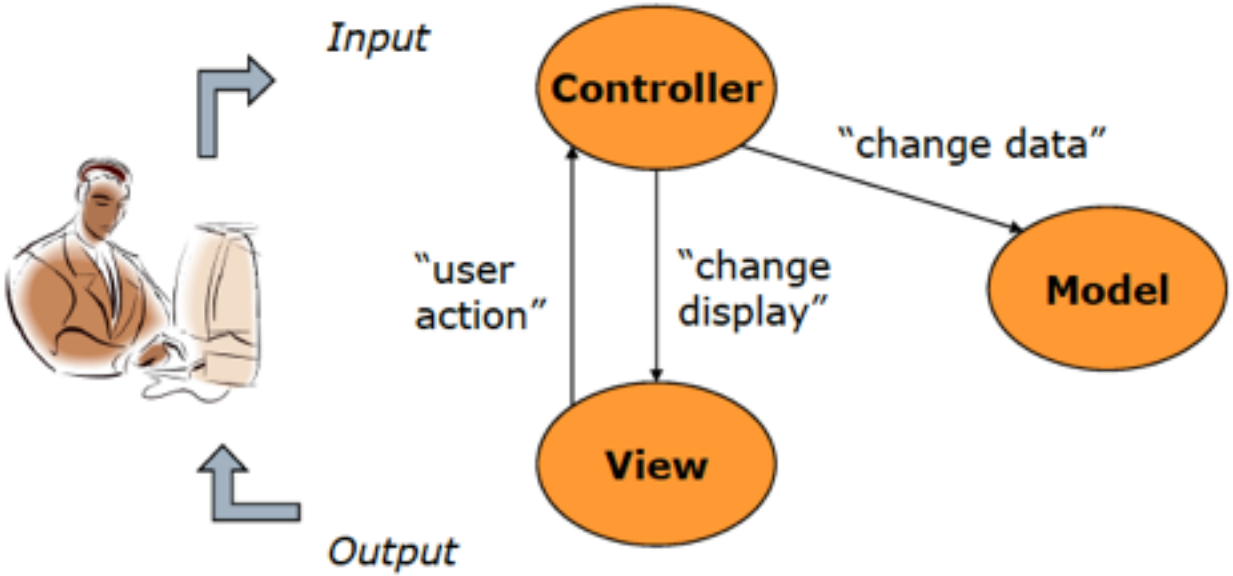
3. MVC 2

- C'est exactement le même modèle de conception MVC à la différence qu'il n'y a plus qu'**un seul contrôleur** qui se charge de rediriger la requête vers le bon traitement et choisir la vue.



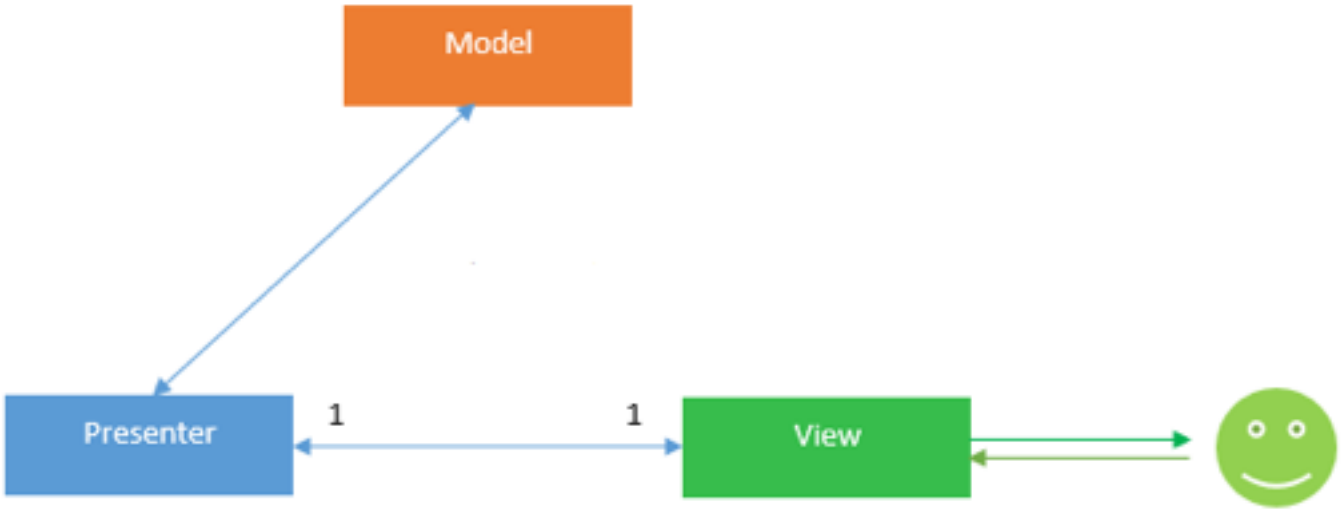
4. Passive View

- Le contrôleur est le seul composant qui peut communiquer avec le modèle et la vue
- Il n'y a pas de communication directe entre Model et View
- Appelé aussi : Model-View-Adapter



5. MVP: Model-View-Presenter

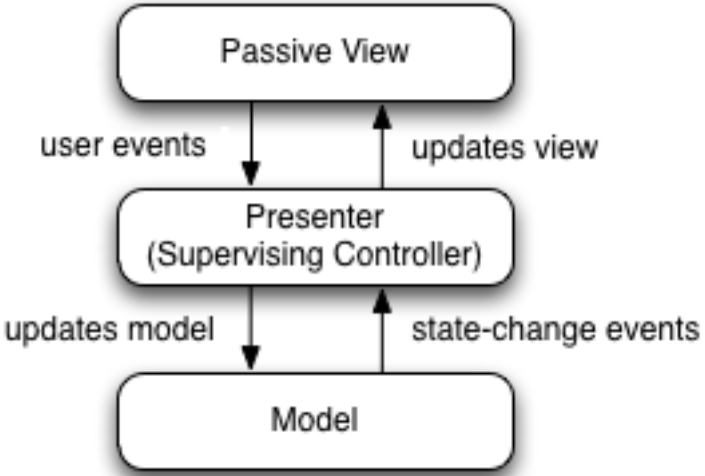
- Le patron le plus utilisé, est le MVP (Modèle Vue Présentation)
  - même si les utilisateurs croient souvent à tord avoir suivi un patron de MVC.
  - le Model et la Vue ont le même rôle dans les patrons MVP et MVC, mais le Presenter diffère par rapport au Controller



MVP Pattern

5. MVP: Model-View-Presenter

- Le Presenter fait l'intermédiaire dans les deux sens entre la Vue et le Modèle, car il va servir à “présenter” les données d’un coté comme de l’autre (Passive View).
- L’utilisateur interagit avec la vue
- Il y a une relation un-à-un entre la vue et le Presenter (une vue/ presenter, un presenter/vue).
- La vue a une référence au Presenter mais pas au modèle
- Ce pattern est très utilisé dans les applications Web ASP.NET



6. MVVM: Model View ViewModel

- En 2004, Microsoft ajoute à sa bibliothèque .NET, un mode MVVM (Model View ViewModel).
  - Leur objectif est l'unification des Vues grâce à un ViewModel.
  - design pattern basé sur MVC et MVP
  - a été largement repris dans le web par des bibliothèques JavaScript.
- L'idée est de faire en sorte que les données modélisées soient communes à toute application et cette généricité de l'interpretation des données est faite dans le ViewModel.

MVVM: Model View ViewModel

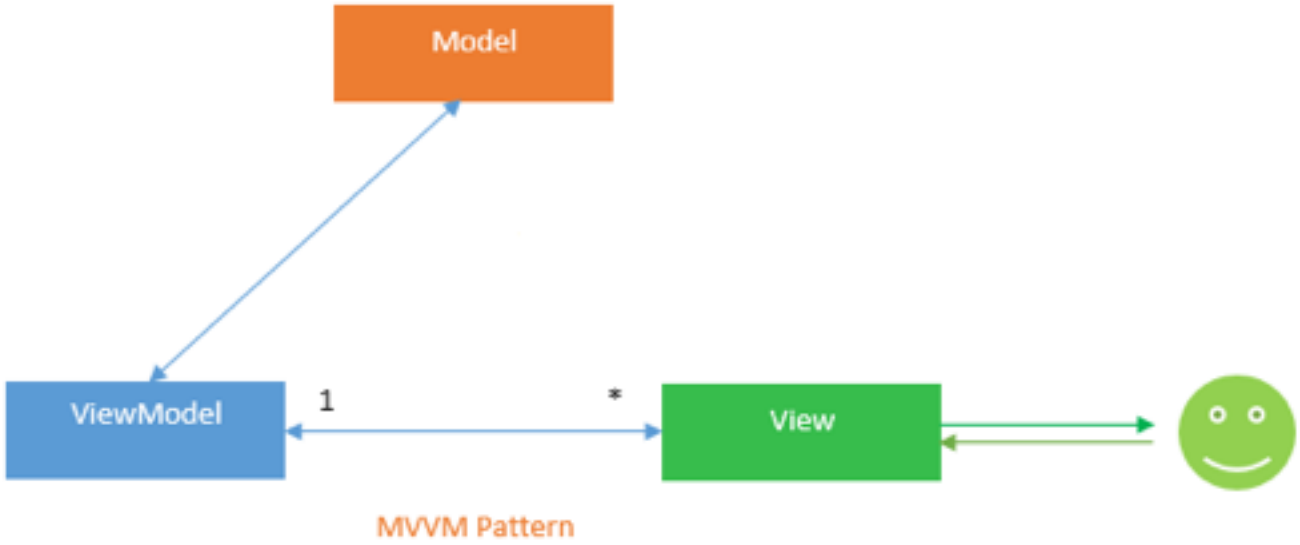
- Intérêt
  - Quand on développe une application web ou autre application d'interface utilisateur, on a tendance à écrire beaucoup de logique UI derrière l'UI elle-même: plusieurs lignes de codes pour gérer les interactions propre au contexte qu'on ne peut pas réutiliser et qui est également difficile à tester
    - Microsoft possède plusieurs langages propriétaires qui utilisent le framework .NET (VB, C#, asp, apsx, …). Donc si chaque Vue hérite du même ViewModel, le code à produire sera moindre.
  - La vue n'aura JAMAIS à traiter des données car le VueModèle en aura la charge. La Vue se contentera de les afficher.

MVVM: Model View ViewModel

- Communication
  - L'un des plus puissants concepts qui vient avec MVVM est celui de **biding** (ou liaison en français). Il s'agit d'un moyen qui permet de mettre automatiquement la vue à jour à chaque fois que le modèle change et inversement de mettre à jour le modèle à chaque fois que la vue change.
    - Microsoft introduit un nouveau composant: le **Binder** qui sert d'intermédiaire entre la Vue et la VueModèle.

MVVM: Model View ViewModel

- Les composantes
  - Model et View : ont le même rôle dans les patrons MVC et MVVM
  - ViewModel : C'est là qu'est représentée la logique spécifique à l'UI. Il permet la liaison entre le modèle et la vue. Il peut être considéré comme un contrôleur spécial qui jouerait le rôle de convertisseur de données.



MVVM: Model View ViewModel

- Avantages
  - Facilite le développement parallèle de l'UI et du modèle.
  - Fait abstraction de la vue et réduit la quantité de code requis pour la lier au modèle.
  - Le ViewModel est plus facile à être passé aux tests unitaires que le code basé sur des événements (peut être testé sans avoir à s'occuper de l'automatisation des interactions avec l'UI).
- Inconvénients
  - MVVM s'implémente mal pour des opérations simples sur l'UI, l'utiliser serait excessif
  - Pour une application plus large, généraliser la Vue devient plus difficile.
  - le data-binding, s'il est mal géré, peut prendre beaucoup de mémoire.

MVC, MVP et MVVM

- MVP et MVVM sont tous les deux dérivés de MVC.
  - La principale différence entre MVC et ses dérivés est la dépendance que chaque composant du modèle entretient avec les autres et la manière dont ils sont étroitement liés.
- En MVC, la Vue est tout en haut de l'architecture avec le contrôleur sous elle:
  - Les vues ont directement accès aux modèles.
  - Exposer le modèle complet à la vue peut entrainer un coût en termes de sécurité et de performances suivant la complexité de l'application.
  - MVVM tente justement d'éviter ces problèmes.

MVC, MVP et MVVM

- En MVP, le rôle du contrôleur est remplacé par la Présentation.
  - Les Présentations sont au même niveau que les vues, écoutant les évènements à la fois de la Vue et du modèle et assurant la médiation entre eux.
  - Contrairement à MVVM, il n'y a pas de mécanisme pour lier les Vues au VueModèle. A la place on compte plutôt sur l'implémentation d'une interface sur chaque Vue autorisant la Présentation à interagir avec elle.

MVC, MVP et MVVM

- MVVM autorise de créer des sous-ensembles spécifiques à la vue d'un modèle qui peut contenir l'information de l'état et de la logique, évitant ainsi d'exposer l'ensemble du modèle à une vue.
  - Contrairement au Présentateur de MVP, un VueModèle n'a pas besoin de faire référence à une vue. La Vue peut se lier à des propriétés du VueModèle qui à son tour expose à la Vue des données contenues dans les modèles.
  - L'un des inconvénients est qu'un niveau d'interprétation est nécessaire entre le VueModèle et la Vue, ce qui peut avoir un certain coût en termes de performance du fait que la complexité peut beaucoup varier.
  - Avec MVC nous n'avons pas ce problème car l'ensemble du modèle est facilement disponible et que ce niveau d'interprétation peut être évité.

Références

- http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf
- Séparation entre Interface Homme Machine et Noyau Fonctionnel. Quelles technologies?Mythe ou réalité ? Romain Huneau, Jérémy Nesmes, Adam Palma, Zhou Ren
- L'architecture MVC : Modèle – Vue - Contrôleur. Génie Logiciel Licence 3 - Université du Havre. Bruno Mermet
- LOG2410 - Conception logicielle, François Guibault, 2006 CHAPITRE 5 Modèle architectural. Ecole Polytechnique Montréal.
- Patrons de conception. Design Patterns. Bruno Bachelet: http://www.nawouak.net