# 4. Conclusion (et remarques)



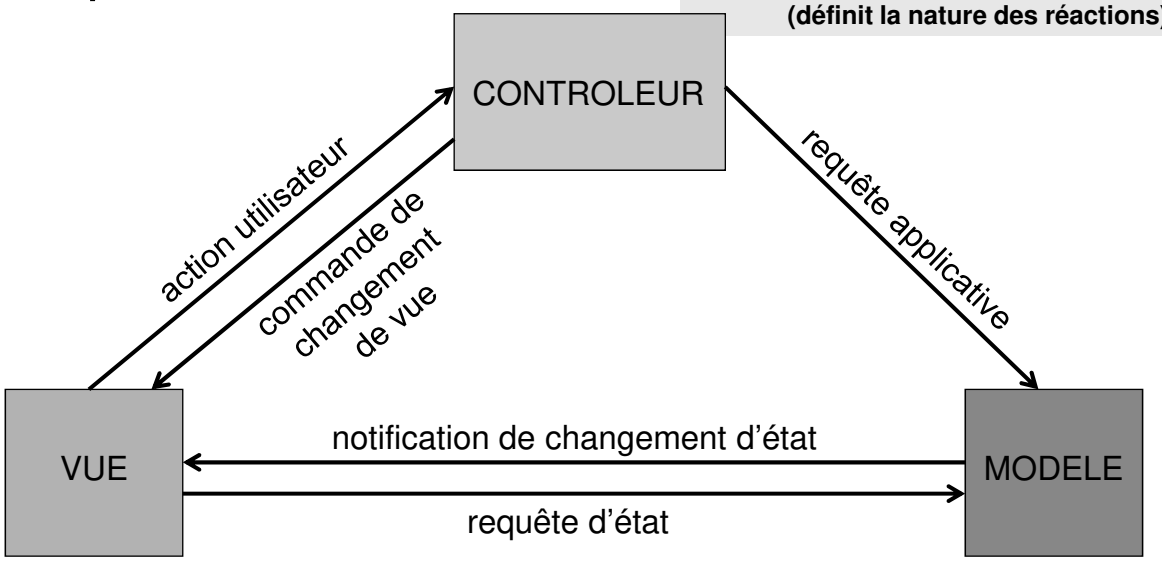


## *Le pattern MVC*

- Problème
  - Dans une application interactive, rendre indépendantes les classes de présentation des classes qui implantent la logique applicative
    - Faciliter la maintenance et la réutilisation
    - Permettre le passage d'une forme de présentation à une autre
- MVC = Modèle-Vue-Contrôleur
  - Origine : Smalltalk-80
    - Structuration de (grosses) applications graphiques
    - Séparation stricte (découplage) entre
      - Les données (le modèle)
      - La présentation *i.e.* l'interface homme-machine (la vue)
      - La logique de contrôle (le contrôleur = intermédiaire dans l'interaction)
  - Ce n'est pas un pattern du GoF mais un pattern (plus complexe) de conception architecturale

# Le pattern MVC

**CONTROLEUR = interface entre vue et modèle**
*** interprète la requête utilisateur et**
*** active les procédures applicatives (définit la nature des réactions)**




**VUE = interface utilisateur (présentation)**
*** gère les entrées-sorties**
**- affichage des données que fournit le modèle**
*** n'effectue aucun traitement**
*** ne gère pas de donnée**
*** il peut y avoir de multiples vues du modèle**
**- elles sont synchronisées**

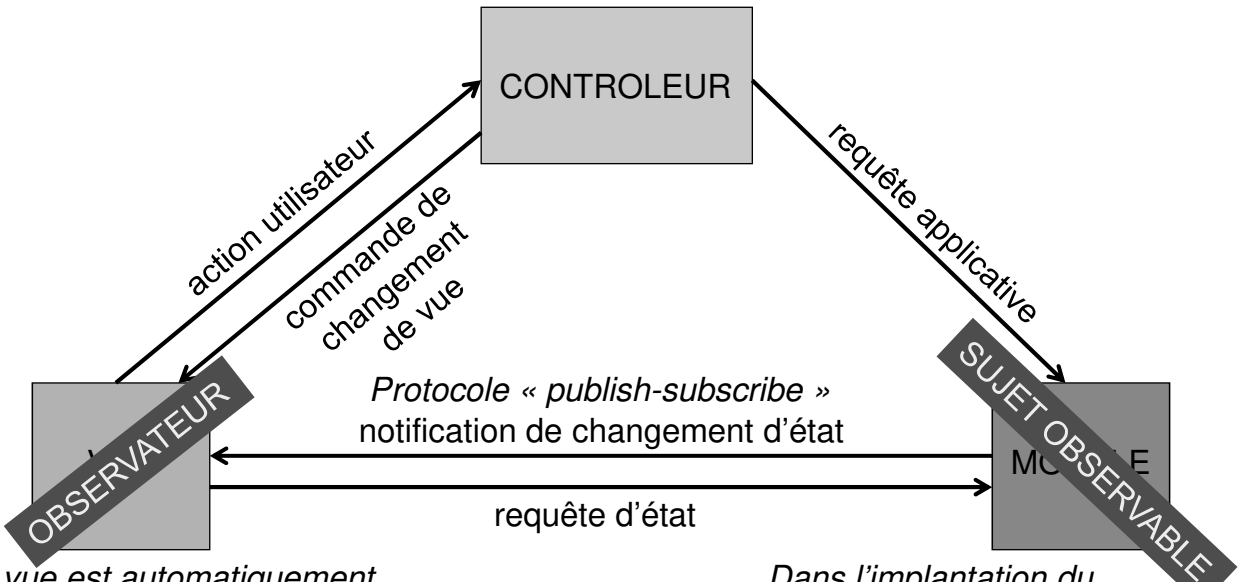
**MODELE = cœur de l'application**
*** gère les données de l'application**
*** définit la logique applicative**
**- traitement des données**
**- interaction avec la BD**





# MVC est un modèle composé

- Modèle OBSERVATEUR




*La vue est automatiquement mise à jour quand le modèle change ou informée du changement*

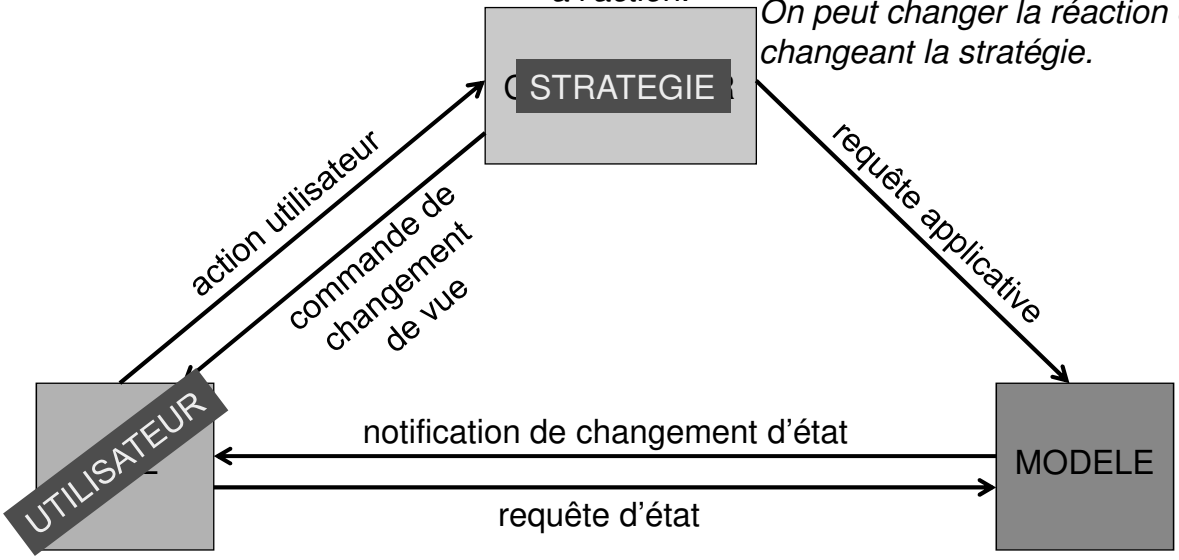
*Dans l'implantation du modèle, il n'y a aucune référence au contrôleur ou à la vue*

# *MVC est un modèle composé*

☛ Modèle STRATEGIE

*Le contrôleur définit comment réagir à l'action.*

*Il fournit une « stratégie » à la vue. On peut changer la réaction en changeant la stratégie.*








# *Patterns, idiomes, anti-patterns*

- Il existe une grande variété de « *design patterns* »
  - Au-delà des GRASP patterns et des 23 patterns du GoF
  - P. ex. « inversion du contrôle » ou « injection de dépendances »…
  - Ou spécifiques à un domaine (p. ex. programmation concurrente ou programmation répartie)
- Idiomes de programmation
  - Techniques d'implémentation spécifiques au langage de programmation (alors que les *design patterns* sont « agnostiques »)
- Et les « anti-patterns »…
  - « Modèles » de mauvais choix et erreurs de conception courantes
    - Exemples : interblocages et famines (synchronisation), plat de spaghetti, duplication de code, surcharge des interfaces utilisateurs…

# *Patterns, boîte à outils et frameworks*

- Un *design pattern*
  - N'est pas une boîte à outils
    - Bibliothèque de composants (classes ou sous-programmes) réutilisables dans un contexte thématique particulier (« métier »)
    - En pratique, on développe le tronc principal de l'application (sans contrainte de conception) et on réutilise les composants de la boîte à outils
  - N'est pas un *framework*
    - Bibliothèque de composants réutilisables qui impose (ou favorise) une certaine architecture pour l'application et mettent en œuvre un design pattern
      - Principe de l'Inversion du contrôle (IoC)
      - Par ex. Struts pour les applications Web (pages JSP / servlets), Spring MVC, Grails…
    - Les *design patterns* sont plus abstraits et moins spécialisés que les *frameworks* (pas d'aspect « métier »)





# *Utilisation des patterns*

- Attention !
  - Un design pattern n'est pas
    - La solution à un problème mais un modèle de solution
    - Une méthode de conception
    - Une règle applicable mécaniquement (rôle du développeur)
  - La maîtrise des design patterns demande
    - Un effort de synthèse (reconnaître, abstraire…)
      - Ils sont nombreux, parfois se ressemblent, ou se composent…
    - De l'expérience (apprentissage, expérimentation)
  - Ne pas surutiliser les design patterns au risque de rendre les applications inutilement complexes
    - Bien identifier ce qui est susceptible d'évolution
    - Privilégier la simplicité, pas l'utilisation de patterns à tout prix
  - En cas de composition de patterns (parfois utile), maîtriser la complexité !

# *Et n'oubliez pas …*

« La seule et unique **constante** du développement c'est …

… le **changement**. »

E. & E. Freeman, Design Patterns – Tête la Première