

- 1 Design Patterns
- 2 General Responsibility Assignment Software Patterns
- 3 Patterns de construction
- 4 Patterns de structuration
- 5 Patterns de comportement
- 6 Model View Controller

# Histoire des Design Patterns

Inspiré de réflexions issues de l'architecture (génie civil) dans les années 70.

## Pattern (Christopher Alexander 1977)

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

## Model View Controller (MVC)

En 1979, TRYGVE REENSKAUG, un des développeurs du langage Smalltalk invente le Modèle Vue Contrôleur ( 69).

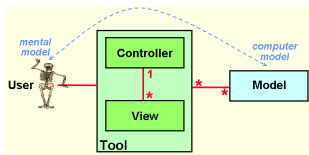


Image provenant de <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>

# Histoire des Design Patterns (suite)

## Le Gang of Four (GoF)

En 1995, le Gang of Four écrit le livre *Design Patterns : Elements of Reusable Object-Oriented Software* publié chez Addison Wesley.

- Le GoF : ERICH GAMMA, RICHARD HELM, RALPH JOHNSON et JOHN VLISSIDES.

## General Responsibility Assignment Software Patterns (GRASP)

CRAIG LARMAN écrit aussi en 1995 *Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development*, publié chez Prentice Hall.

- Les modèles GRASP sont plus de bonnes pratiques que des modèles architecturaux.

# Les 9 modèles GRASP

- Information Expert ( 8)
- Creator ( 10)
- Controller ( 11)
- Low Coupling ( 12)
- High Cohesion ( 14)
- Polymorphism ( 15)
- Pure Fabrication ( 16)
- Indirection ( 17)
- Protected Variations ( 18)

# Contenu du livre Design Patterns : Elements of Reusable Object-Oriented Software

## Catalogue de modèles de conception objet

Le catalogue est organisé en trois rubriques :

- les modèles de construction : comment créer les objets,
- les modèles de structuration : comment définir les relations entre les unités de programme,
- et les modèles de comportement : comment les objets communiquent.

Beaucoup d'autres modèles ont été conceptualisés.

# Les 23 modèles du catalogue du GoF

## Construction

- Abstract Factory ( 20)
- Builder ( 22)
- Factory Method ( 24)
- Prototype ( 26)
- Singleton ( 28)

## Structuration

- Adapter ( 31)
- Bridge ( 33)
- Composite ( 35)
- Decorator ( 37)
- Façade ( 39)
- Flyweight ( 41)

## Comportement

- Chain of Responsibility ( 46)
- Command ( 48)
- Interpreter ( 50)
- Iterator ( 52)
- Mediator ( 54)
- Memento ( 56)
- Observer ( 58)
- State ( 60)
- Strategy ( 62)
- Template Method ( 64)
- Visitor ( 66)

- 1 Design Patterns
- 2 General Responsibility Assignment Software Patterns
- 3 Patterns de construction
- 4 Patterns de structuration
- 5 Patterns de comportement
- 6 Model View Controller

# Information Expert

## Expert en Information

La responsabilité doit être donnée à la classe qui dispose des informations pour l'exercer.

- respecte l'encapsulation.



# Encapsulation ou Information Hiding de David Parnas 1972

## Encapsulation

Cacher les données, donc les champs en programmation objet. Ne les manipuler qu'à travers des méthodes. Ne rendre visible que ce qui est nécessaire.

- Les champs sont **private**.
- Seules les méthodes destinées à être utilisées en dehors de la classe sont **public**.
- améliore la robustesse du code.
- améliore l'évolutivité.

# Creator

## Créateur

Une classe A est responsable de la création d'un objet de la classe B si

- A agrège des objets de B ;
- A contient des objets de B ;
- A utilise des objets de B ;
- A possède les données nécessaires à l'initialisation des objets de B.

# Controller

## Contrôleur

Le contrôleur reçoit et coordonne les messages systèmes.

- Le contrôleur de façade gère l'accès à l'ensemble du système.
- Le contrôleur de scénario ou de session gère les événements contenus dans un même cas d'utilisation.
- Le contrôleur ne fait pas partie de l'interface utilisateur mais est le premier objet qui reçoit les événements qu'elle envoie.
- Le contrôleur n'effectue pas tous les traitements. Il délègue.
- Il correspond au contrôleur du MVC.

# Low Coupling

## Faible couplage

Définir les responsabilités en minimisant les liens entre les différents éléments pour faciliter la maintenance, l'ajout de fonctionnalités et la réutilisabilité.

- Les classes génériques doivent être faiblement couplées.
- Ne pas abuser car le code peut devenir rapidement trop complexe.

# Law of Demeter (Don't talk to strangers)

## Loi de Déméter

Dans une méthode, l'envoi de message se limite à

- l'objet qui invoque la méthode **this** ;
  - un paramètre de la méthode ;
  - un champ de l'objet **this** ;
  - un élément d'une collection qui est un champ de l'objet **this** ;
  - un objet créé dans la méthode.
- 
- Éviter de chaîner les méthodes en invoquant une méthode d'un objet retourné par une autre méthode :  
    `sale .getPayment().getAmount()`
  - La loi de Déméter aide à minimiser le couplage.

# High Cohesion

## Forte cohésion

Les éléments d'une unité fonctionnelle collaborent tous ensemble dans un périmètre bien délimité pour assurer un nombre limité de responsabilités.

- Il est difficile de trouver la bonne mesure pour assurer à la fois la forte cohésion et le faible couplage.

# Polymorphism

## Polymorphisme

Le polymorphisme consiste à utiliser le même nom pour réaliser des fonctions similaires. Le choix de la méthode est déterminé par le type de l'objet qui invoque la méthode.

- Éviter de tester dans le code avec des **if**, le type de l'objet qui invoque la méthode pour choisir la bonne méthode.

# Pure Fabrication

## Fabrication pure

Classe créée artificiellement pour réparer une architecture qui manque de forte cohésion et de faible couplage. Elle prend en charge l'échange des messages concernés et vise à les rétablir.

- encapsule une méthode qui ne trouve pas sa place ailleurs.
- Ne pas en abuser.



# Indirection

## Indirection

Classe qui sert d'intermédiaire entre deux classes pour assurer le faible couplage.

- par exemple, le contrôleur du MVC

# Protected Variations

## Variations protégées

Interface stable destinée à protéger des variations présentes dans le système actuel et dans ses évolutions.

- permet de respecter l'encapsulation.
- permet de respecter le principe ouvert fermé.

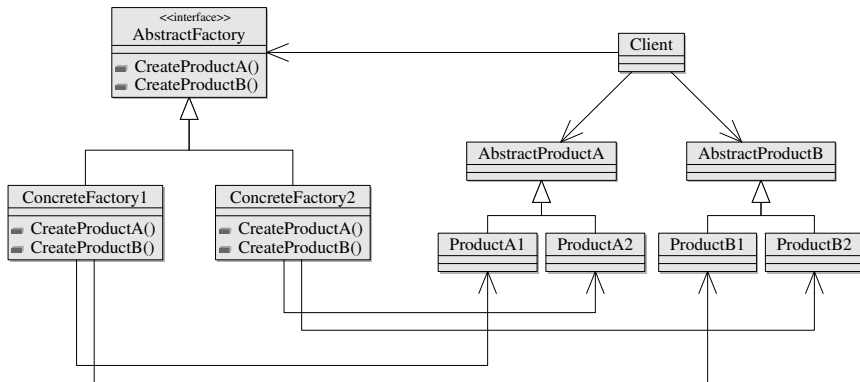
- 1 Design Patterns
- 2 General Responsibility Assignment Software Patterns
- 3 Patterns de construction**
- 4 Patterns de structuration
- 5 Patterns de comportement
- 6 Model View Controller

# Abstract Factory

## Fabrication abstraite

Définir une interface pour créer des familles d'objets sans avoir à spécifier leurs classes concrètes.

# Abstract Factory

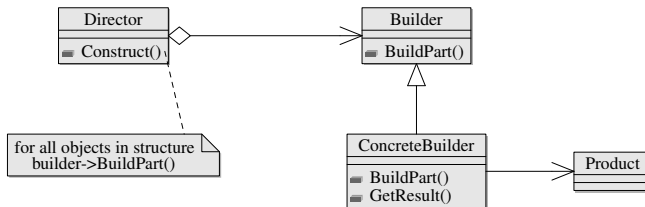


# Builder

## Monteur

Dissocier la construction de la représentation de l'objet. Un même procédé de construction engendre des représentations différentes.

# Builder



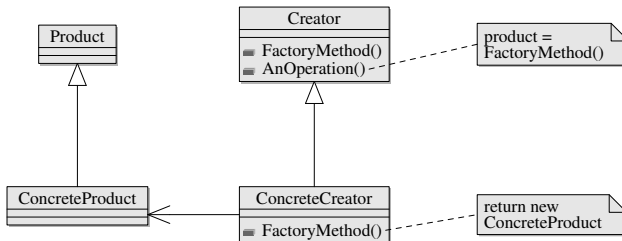
# Factory Method

## Fabrication

Déferer la fabrication d'un objet à des classes dérivées.



# Factory Method

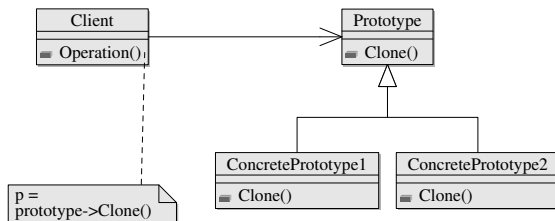


# Prototype

## Prototype

Créer des objets par copie à partir d'un objet prototype.

# Prototype

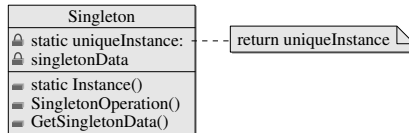


# Singleton

## Singleton

Garantir qu'une classe n'a qu'une instance et fournir un accès global à celle-ci.

# Singleton



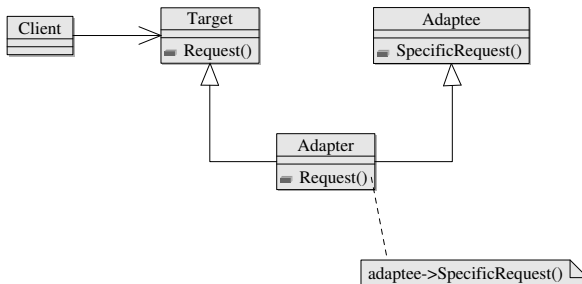
- 1 Design Patterns
- 2 General Responsibility Assignment Software Patterns
- 3 Patterns de construction
- 4 Patterns de structuration**
- 5 Patterns de comportement
- 6 Model View Controller

# Adapter

## Adaptateur

Convertir l'interface d'une classe en une interface utilisable par un client.

# Adapter



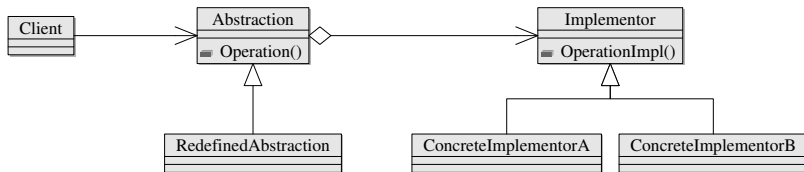


# Bridge

## Pont

Découpler une abstraction et son implémentation pour pouvoir les modifier indépendamment.

# Bridge



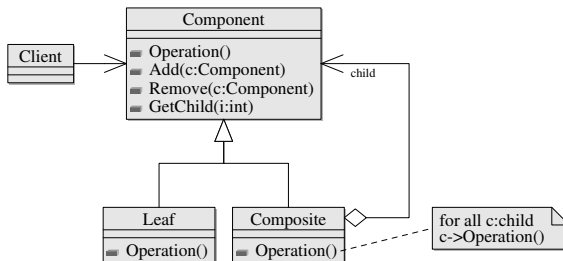
# Composite

## Composite

Organiser les objets en structure arborescente de manière à pouvoir utiliser un groupe d'objets comme s'il s'agissait d'un seul objet.

- Par exemple, les répertoires et les fichiers.

# Composite

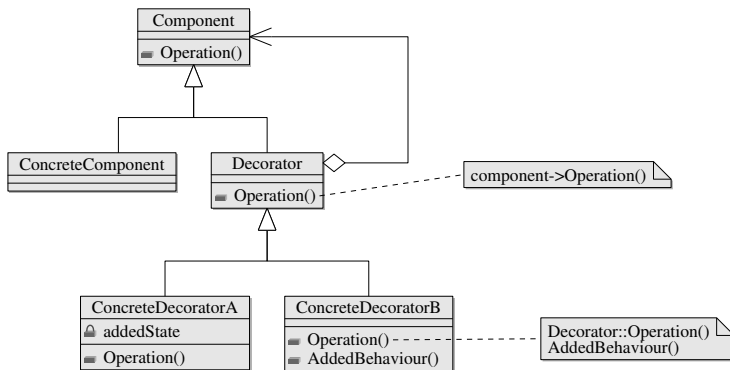


# Decorator

## Décorateur

Attacher des responsabilités supplémentaires à un objet de façon dynamique.

# Decorator



# Facade

## Façade

Fournir une interface unifiée pour un ensemble d'interfaces d'un sous-système.

# Facade

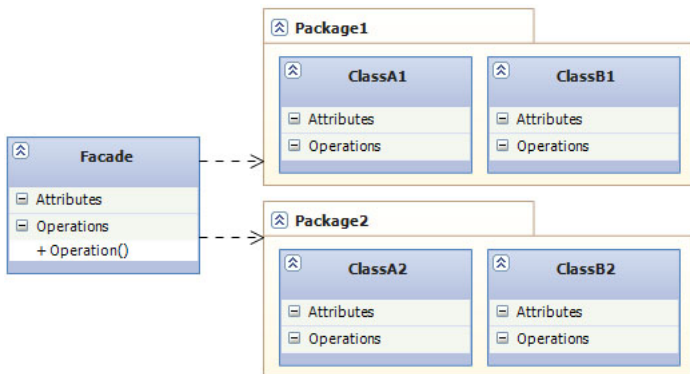


image provenant de

<http://www.codeproject.com/Articles/438922/Design-Patterns-of-Structural-Design-Patterns>

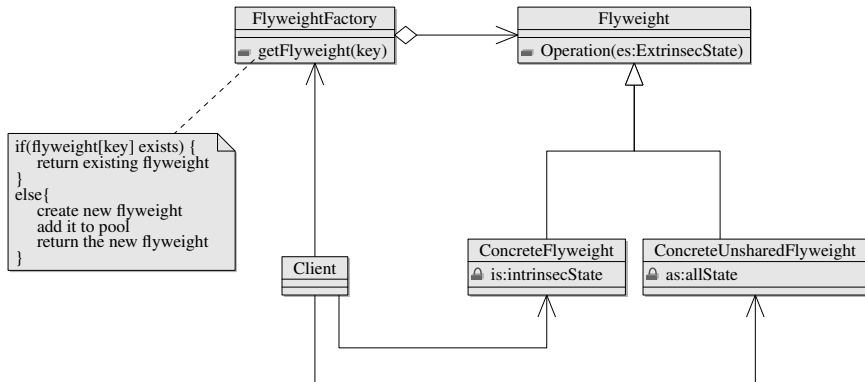


# Flyweight

## Poids mouche

Alléger l'occupation mémoire en factorisant les parties communes des objets et en les instanciant qu'une seule fois et gérer les parties qui diffèrent.

# Flyweight

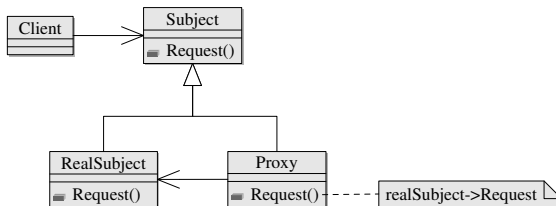


# Proxy

## Procuration

Fournir une classe qui peut se substituer à une autre classe pour contrôler l'accès aux méthodes de cette dernière.

# Proxy



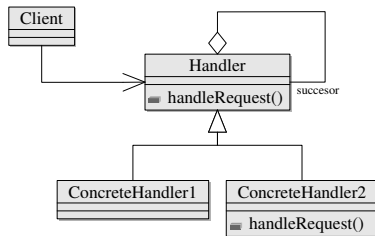
- 1 Design Patterns
- 2 General Responsibility Assignment Software Patterns
- 3 Patterns de construction
- 4 Patterns de structuration
- 5 Patterns de comportement
- 6 Model View Controller

# Chain of Responsibility

## Chaîne de responsabilité

Éviter de coupler l'émetteur et le destinataire d'une requête en permettant à plusieurs objets de la prendre en charge. Les objets sont chaînés jusqu'à l'objet qui traite la requête.

# Chain of Responsibility



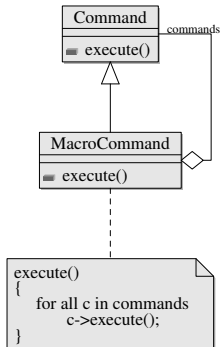
# Command

## Commande

Encapsuler une requête dans un objet. Permettre de paramétrer les clients avec plusieurs requêtes, files d'attente, historique, d'assurer le traitement des requêtes réversibles.



# Command



# Interpreter

## Interpréteur

Représenter la grammaire d'un langage et réaliser une analyse syntaxique.

# Interpreter

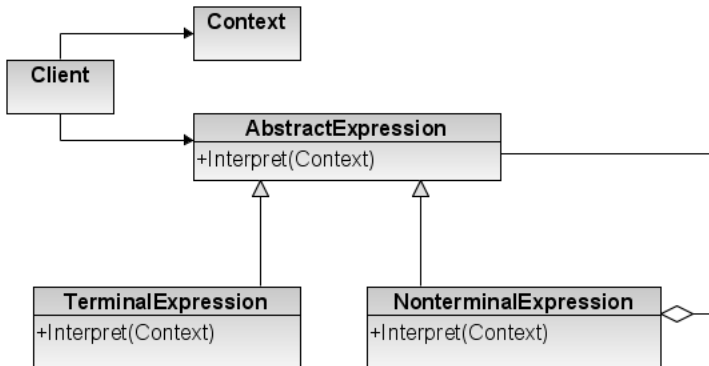


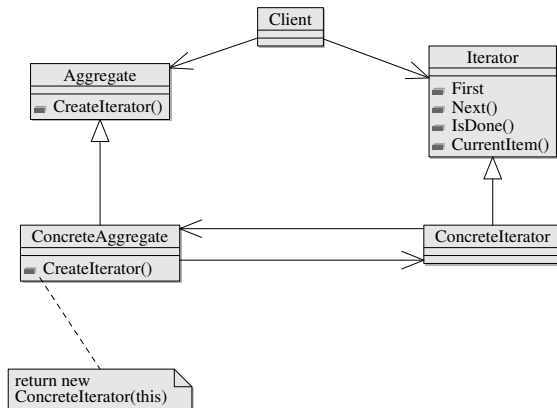
Image provenant de <https://javaobsession.wordpress.com/page/2/>

# Iterator

## Itérateur

Accéder en séquence aux éléments d'un objet de type agrégat sans révéler sa représentation sous jacente.

# Iterator

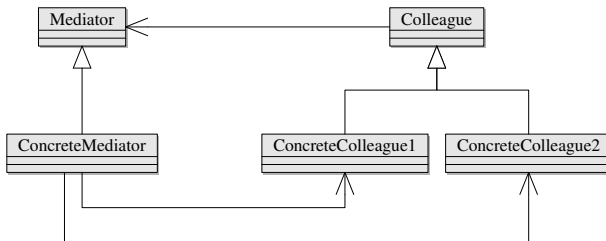


# Mediator

## Médiateur

Encapsuler les modalités d'interaction entre objets pour favoriser les couplages faibles. Permettre de modifier une relation indépendamment des autres.

# Mediator



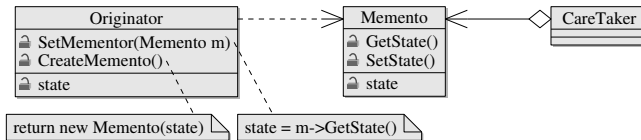
# Memento

## Mémento

Acquérir et délivrer une information sur un état interne d'un objet sans violer l'encapsulation pour pouvoir rétablir l'objet dans cet état ultérieurement.



# Memento

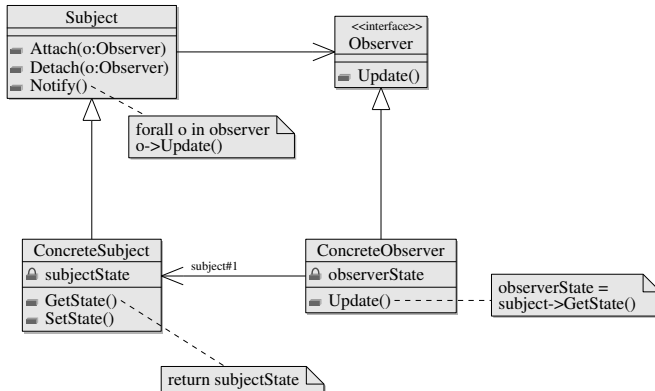


# Observer

## Observateur

Relation de type un à plusieurs : lorsqu'un objet change d'état tous ceux qui l'observent sont avertis et mis à jour.

# Observer

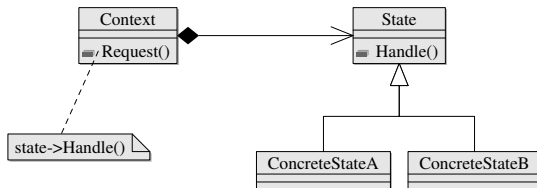


# State

## État

Modifier le comportement d'un objet lorsque son état change. L'objet semble changer de classe.

# State

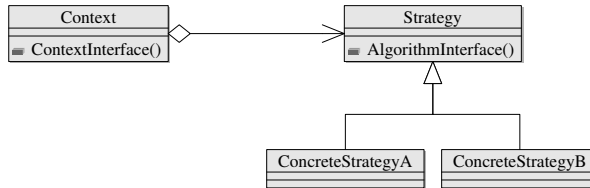


# Strategy

## Stratégie

Encapsuler une famille d'algorithmes pour les rendre interchangeables. Permettre de modifier un algorithme indépendamment des clients.

# Strategy



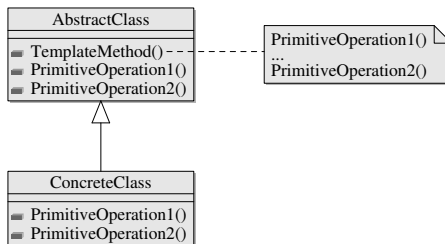
# Template Method

## Patron de méthode

Écrire un algorithme qui repose sur des traitements effectués dans des classes dérivées.



# Template Method

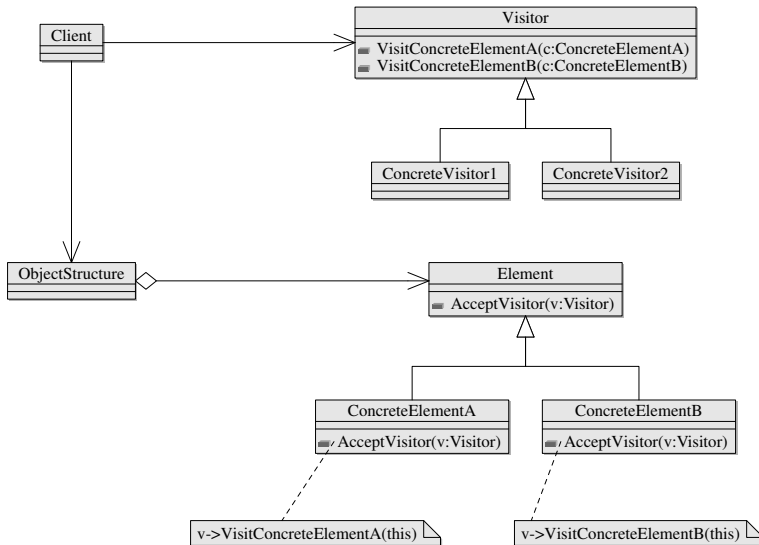


# Visitor

## Visiteur

Définir une nouvelle opération pour plusieurs classes.

# Visitor



- 1 Design Patterns
- 2 General Responsibility Assignment Software Patterns
- 3 Patterns de construction
- 4 Patterns de structuration
- 5 Patterns de comportement
- 6 Model View Controller

# Model View Controller

## Modèle Vue Controller

Séparer le modèle, noyau fonctionnel qui gère les données de l'application, de la vue, composants qui affichent les données pour l'utilisateur et du contrôleur qui envoie les requêtes de l'utilisateur à la vue et au modèle.

- La liaison entre la vue et le modèle utilise le modèle **Observer**.
- La liaison entre la vue et le contrôleur utilise le modèle **Strategy**.
- La vue utilise le modèle **Composite**.
- Le contrôleur utilise le modèle GRASP **Controller**.

# Model View Controller

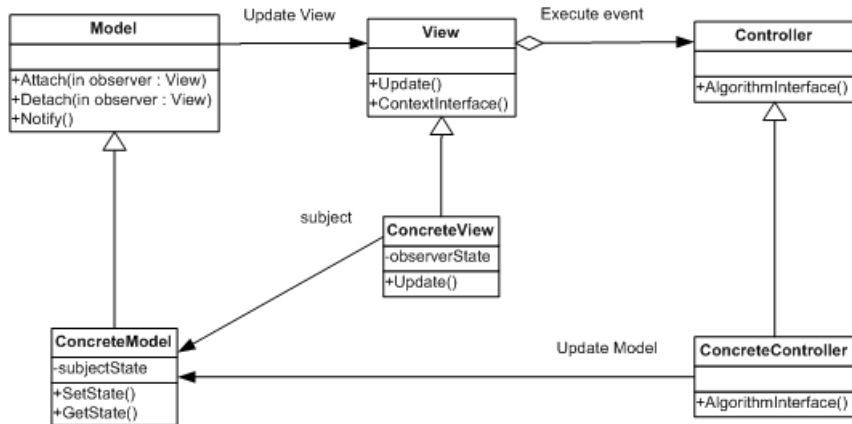


Image provenant de