Modèles de Conception

Olivier Baudon
Université Bordeaux 1
13 octobre 2002

La référence

Gamma, E., Helm, R., Johnson R. and Vlissides, J.,
« *Design Patterns, Elements of Reusable Object-Oriented Software* »
Addison-Wesley Publishing Company, 1995

Mais aussi
- des sites web,
- des ouvrages sur des contextes particuliers :
  - Java
  - programmation distribuée

Qu'est ce qu'un modèle de conception

- Nom : *permet d'identifier le modèle utilisé*
- Problème à traiter
- Solution : *pas un modèle précis, encore moins une implémentation*
- Conséquences

Description des modèles

Présentation
- Nom et classification
- Intention
- Alias
- Motivation
- Indications d'utilisation

Réalisation
- Structure
- Constituants
- Collaborations
- Conséquences
- Implémentation
- Exemples de code
- Utilisations remarquables
- Modèles apparentés

Trois rôles

- Modèles créateurs processus d'instanciation
- Modèles structuraux composition des classes et des objets
- Modèles de comportement algorithmes et répartition des tâches entre objets

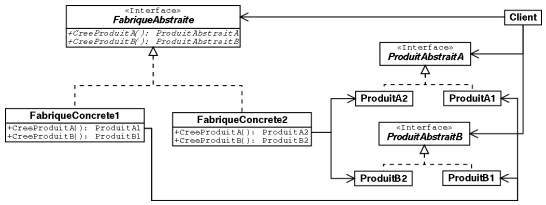
Modèles créateurs

- Fabrique abstraite : *familles de produits*
- Monteur : *création d'un objet composite*
- Fabrication : *délégation aux sous-classes des instanciations*
- Prototype : *instanciation par clonage*
- Singleton : *classe à instance unique avec accès global*

Fabrique abstraite

« La fabrique abstraite fournit une interface pour la création de familles d'objets apparentés ou interdépendants, sans qu'il soit nécessaire de spécifier leur classe concrète »

Fabrique abstraite



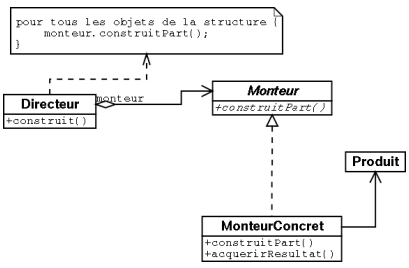
Fabrique abstraite

+ isole les classes concrètes (encapsulation)
+ facilite la substitution de familles de produits
+ favorise le maintien de la cohérence entre les objets
- gérer de nouveaux types de produits est difficile

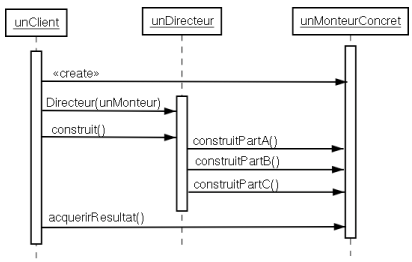
Monteur

« Dissocie la construction d'un objet complexe de sa représentation, de sorte que le même processus de construction permette des représentations différentes »

Monteur



Monteur



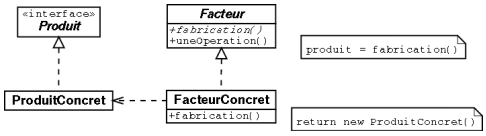
Monteur

+ permet de modifier la représentation interne d'un produit
+ isole le code de construction et de représentation
+ permet un meilleur contrôle du processus de construction

Fabrication

« Définit une interface pour la création d'un objet, mais en laissant à des sous-classes le choix des classes à instancier. La fabrication permet à une classe de déléguer l'instanciation à des sous-classes. »

Fabrication



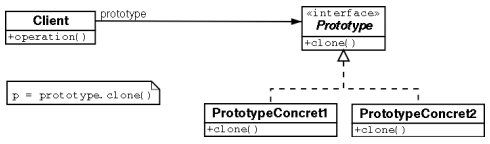
Fabrication

+ procure un gîte aux sous-classes
+ interconnecte des hiérarchies parallèles de classes

Prototype

« Spécifie le type des objets à créer à partir d'une instance de prototype, et crée de nouveaux objets en copiant ce prototype. »

Prototype



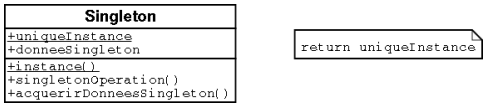
Prototype

+ addition et suppression de produits à l'exécution
+ spécification de nouveaux objets par valeurs modifiables
+ spécification de nouveaux objets par structures modifiables
+ limite le nombre de dérivations de classes
+ facilite le chargement dynamique

Singleton

« Garantit qu'une classe n'a qu'une seule instance et fournit un point d'accès de type global à cette classe »

Singleton



Singleton

+ accès controlé à une instance unique
+ réduction de l'espace des noms
+ raffinement des opérations et de la représentation
+ autorise un nombre variable d'instances
+ souplesse améliorée par rapport aux opérations de classes

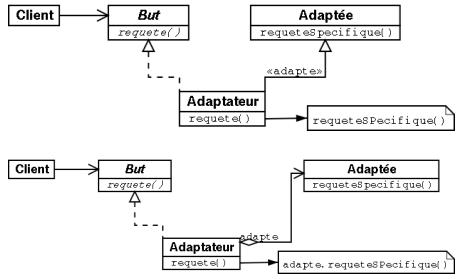
Modèles structuraux

- Adaptateur : *changer d'interface*
- Pont : *dissocier une interface et ses implémentations*
- Composite : *composer des objets en structures arborescentes*
- Décorateur : *Enrichir dynamiquement un objet*
- Façade : *unifier l'interface d'un système*
- Poids mouche : *partager des objets pour en diminuer le nombre*
- Procuration : *fournir un objet de substitution pour contrôler l'accès*

Adaptateur

« Convertit l'interface d'une classe en une autre conforme à l'attente du client. L'Adaptateur permet à des classes de collaborer, qui n'auraient pu le faire du fait d'interfaces incompatibles. »

Adaptateur



Adaptateur

Un adaptateur de classe (par héritage)
- ne peut adapter une classe et ses sous-classes simultanément
+ permet de redéfinir certains comportements de l'*Adaptée*
+ introduit un seul objet et aucun pointeur supplémentaire pour atteindre *Adaptée*

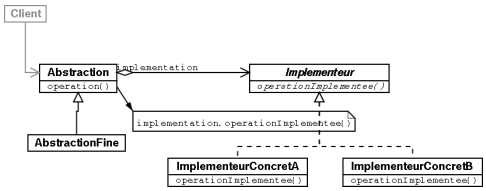
Adaptateur

Un adaptateur d'objet (par délégation)
+ permet à un seul *Adaptateur* d'adapter simultanément *Adaptée* et toutes ses sous-classes.
+ permet d'ajouter des fonctionnalités à tous les *Adaptées* en une seule fois.
- rend plus difficile la surcharge du comportement de l'*Adaptée*. Une telle surcharge impose la dérivation de l'*Adaptée* et impose que l'*Adaptateur* fasse référence à la sous-classe plutôt qu'à *Adaptée* elle-même.

Pont

«découple une abstraction de son implémentation afin que les deux éléments puissent être modifiés indépendamment l'un de l'autre.»

Pont



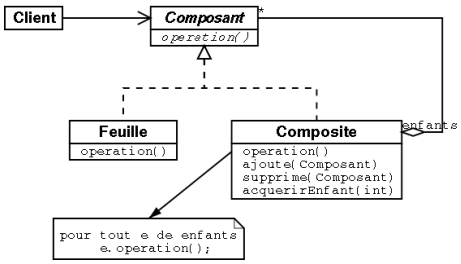
Pont

+ découplage de l'interface et de l'implémentation
+ évite de recompiler l'*Abstraction* lors des changements d'implémentations
+ les hiérarchies d'*Abstraction* et d'*Implementeur* peuvent être étendues indépendamment l'une de l'autre
+ dissimule des détails d'implémentation aux clients tels que les partages d'objets ou le comptage de références.

Composite

« Le modèle Composite compose des objets en des structures arborescentes pour représenter des hiérarchies composant/composé. Il permet au client de traiter de la même et unique façon les objets individuels et les combinaisons de ceux-ci. »

Composite



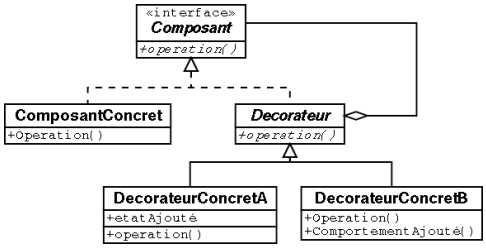
Composite

+ simplifie le niveau client. Le client peut traiter à l'identique les structures composites et les objets individuels
+ rend plus facile l'adjonction de nouveaux types de composants.
- rend difficile l'imposition de contraintes sur la composition d'un composite.

Décorateur

« Attache dynamiquement des responsabilités supplémentaires à un objet. Les décorateurs fournissent une alternative souple à la dérivation, pour étendre les fonctionnalités »

Décorateur



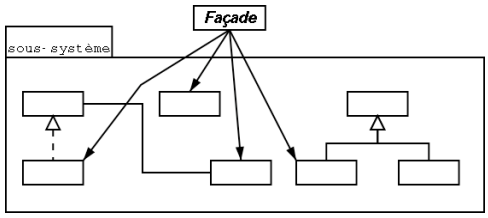
Décorateur

+ plus de souplesse que l'héritage classique
+ évite de surcharger les classes situées les plus en haut de la hiérarchie
- un objet décoré n'a pas la même identité que l'objet de départ
- augmente le nombre d'objets

Façade

« Fournit une interface unifiée à l'ensemble des interfaces d'un sous-système. La facade fournit une interface de plus haut niveau, qui rend le sous-système plus facile à utiliser »

Façade



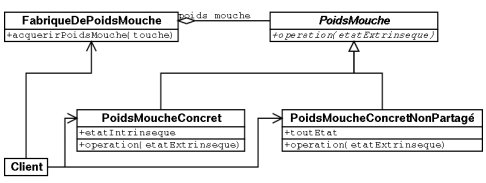
Façade

+ masque les composants du sous-système et le rend plus facile à utiliser
+ favorise un couplage faible entre le sous-système et ses clients
+ n'empêche pas les applications d'utiliser les classes du sous-système si nécessaire

Poids mouche

« Le modèle Poids mouche utilise une technique de partage qui permet la mise en œuvre efficace d'un grand nombre d'objets de fine granularité »

Poids mouche



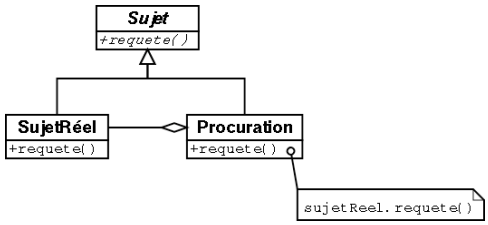
Poids mouche

+ économie de place mémoire
- coûts à l'exécution du fait du transfert des états extrinsèques

Procuration

« Fournit à un tiers objet un mandataire ou un remplaçant, pour contrôler l'accès à cet objet »

Procuration



Procuration

+ introduit un degré d'indirection dans l'accès à un objet
  - permet de cacher qu'un objet réside dans un autre espace d'adresse (cf EJB)
  - permet d'effectuer des optimisations telles que la création d'un objet à la demande
  - permet de vérifier si l'appelant a les permissions requises pour effectuer la requête

Modèles comportementaux

- Chaîne de responsabilités : *éviter le couplage entre l'émetteur et les récepteurs d'une requête*
- Commande : *encapsuler une requête comme un objet*
- Interpréteur : *définir une représentation d'une grammaire ainsi qu'un interpréteur utilisant cette représentation*
- Itérateur : *fournit un accès séquentiel aux éléments d'un agrégat*
- Médiateur : *unifier les modalités d'interaction d'un ensemble d'objets*
- Memento : *saisir et transmettre l'état interne d'un objet sans violer l'encapsulation, pour pouvoir le restaurer*

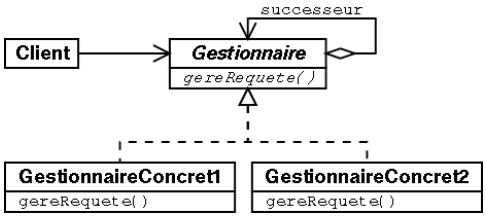
Modèles comportementaux

- Observateur : *définir une interdépendance de type « un à plusieurs »*
- Etat : *permet à un objet de modifier son comportement quand son état interne change, comme s'il changeait de classe*
- Stratégie : *définit une famille d'algorithmes et les rend interchangeables*
- Patron de méthode : *définit le squelette d'un algorithme en en déléguant certaines étapes à des sous-classes*
- Visiteur : *représentation d'une opération applicable aux éléments d'une structure d'objets*

Chaîne de responsabilités

« Eviter le couplage de l'émetteur d'une requête à ses récepteurs, en donnant à plus d'un objet la possibilité d'entreprendre la requête. Chaîner les objets récepteurs et faire passer la requête tout au long de la chaîne, jusqu'à ce qu'un objet la traite. »

Chaîne de responsabilités



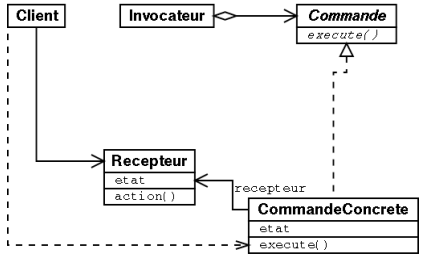
Chaîne de responsabilités

+ réduit le couplage.
+ accroît la souplesse dans l'attribution de responsabilités aux objets.
- la réponse n'est pas garantie.

Commande

« Encapsuler une requête comme un objet, autorisant ainsi le paramétrage des clients par différentes requêtes, files d'attente et récapitulatifs de requêtes, et de plus, permettant la réversion des opérations. »

Commande



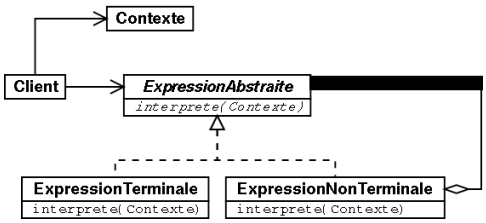
Commande

+ supprime tout couplage entre l'objet qui invoque une opération et celui qui sait comment réaliser cette opération.
+ les commandes sont des objets : elles peuvent être manipulées et étendues comme tel.
+ on peut assembler les commandes dans une commande composite.

Interpréteur

« Pour un langage donné, définir une représentation de sa grammaire, en même temps qu'un interpréteur utilisant cette représentation, pour interpréter les phrases du langage. »

Interpréteur



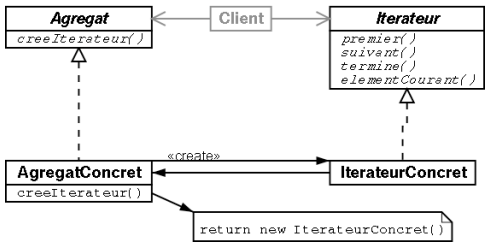
Interpréteur

+ facilite la modification et l'extension de la grammaire
+ le codage de la grammaire est facile
+ facilite l'introduction de nouveaux modes d'interprétation des expressions
- peu adapté aux grammaires complexes. Le nombre de classes peut être élevé et le résultat difficile à maintenir

Itérateur

« Fournit un moyen d'accès séquentiel aux éléments d'un agrégat d'objets, sans mettre à découvert la représentation interne de celui-ci. »

Itérateur



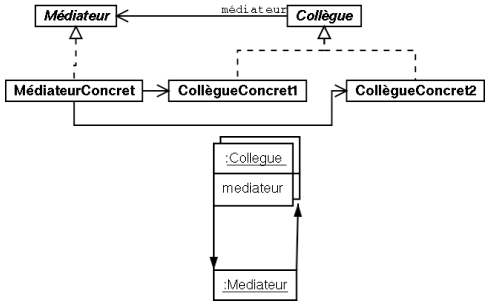
Itérateur

+ permet des modifications dans le parcours des agrégats
+ simplifie l'interface de l'agrégat
+ permet plusieurs parcours simultanés de l'agrégat

Médiateur

« Définit un objet qui encapsule les modalités d'interaction d'un certain ensemble d'objets. Le Médiateur favorise le couplage faible en dispensant les objets de se faire explicitement référence, et il permet donc de faire varier indépendamment les relations d'interaction. »

Médiateur



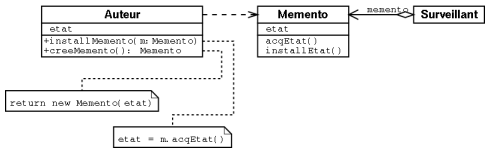
Médiateur

+ limite la création de sous-classes.
+ réduit le couplage entre *Collègues*.
+ simplifie les protocoles objet.
+ formalise la coopération des objets.
- le *Médiateur* échange la complexité des interactions contre la sienne propre. Il peut devenir complexe et difficile à maintenir.

Memento

« Sans violation de l'encapsulation, saisir et transmettre à l'extérieur d'un objet l'état interne de celui-ci, dans le but de pouvoir ultérieurement le restaurer dans cet état. »

Memento



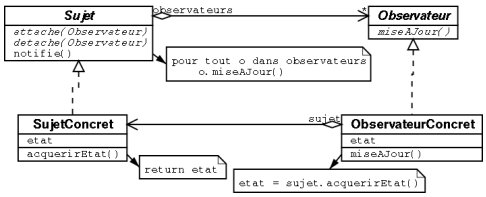
Memento

+ préserve les frontières d'encapsulation.
+ simplifie l'auteur.
- l'utilisation peut être coûteuse.

Observateur

« Définit une interdépendance de type un à plusieurs, de façon telle que, quand un objet change d'état, tous ceux qui en dépendent en soient notifiés et automatiquement mis à jour. »

Observateur



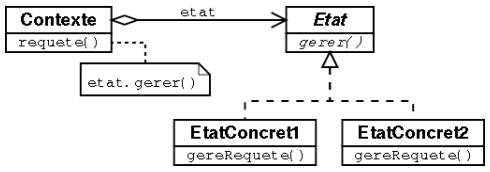
Observateur

+ isole la liaison du *Sujet* vers les *Observateurs* dans la liste des *Observateurs*
- le coût final de la modification d'un *Sujet* est difficile à estimer. Des interdépendances mal contrôlées peuvent entraîner des erreurs : ordre de mise à jour incorrect, circuit …

Etat

« Permet à un objet de modifier son comportement, quand son état interne change. Tout se passera comme si l'objet changeait de classe. »

Etat



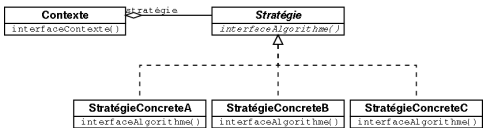
Etat

+ isole les comportements spécifiques d'états et fait un partitionnement des différents comportements état par état.
+ rend les transitions d'état plus explicites.
+ les états peuvent être partagés (*poids mouche*).

Stratégie

« Définit une famille d'algorithmes, encapsule chacun d'entre eux, et les rend interchangeables. Le modèle Stratégie permet aux algorithmes d'évoluer indépendamment des clients qui les utilisent. »

Stratégie



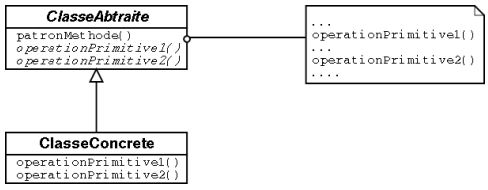
Stratégie

+ les hiérarchies de classes Stratégie définissent une famille d'algorithmes ou comportements réutilisables
+ fournit une alternative à l'héritage
+ dispense de l'utilisation de déclarations conditionnelles
+ fournit un choix d'implémentations pour le même comportement
- il est probable que toutes les stratégies n'utiliseront pas toute l'information passées par le contexte
- prolifération d'objets.

Patron de méthode

« définit, dans une opération, le squelette d'un algorithme, en en déléguant certaines étapes à des sous-classes. Le Patron de méthode permet de redéfinir par des sous-classes, certaines parties d'un algorithme, sans avoir à modifier la structure de ce dernier. »

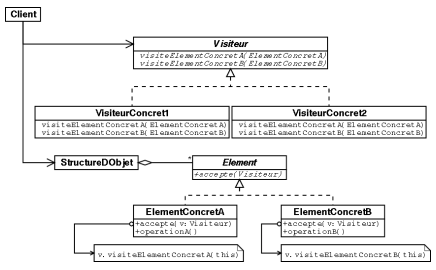
Patron de méthode



Visiteur

« Le Visiteur fait la représentation d'une opération applicable aux éléments d'une structure d'objets. Il permet de définir une nouvelle opération, sans qu'il soit nécessaire de modifier la classe des éléments sur lesquels elle agit. »

Visiteur



Visiteur

+ facilite l'addition de nouvelles opérations
+ rassemble des opérations du même type et isole celles non apparentées
- l'addition de nouvelles classes ElementConcret est difficile
+ un visiteur peut visiter des éléments n'ayant aucune classe parente en commun
+ un visiteur peut collecter des informations sur chaque élément qu'il visite
- oblige à rendre publique la partie de l'état interne d'un élément nécessaire au fonctionnement du visiteur