R5.A.08 : Qualité de développement
(I.Borne)

Cours 4

- Patterns
  - Intention interfaces : Facade, Bridge
- Le pattern d'architecture MVC

Classification des patterns selon leur intention

Intention	Patterns
Interfaces	Adapter, Facade, Composite, Bridge
Responsability	Singleton, Observer, Mediator, Proxy, Chain of Responsability, Flyweight
Construction	Builder, Factory Method, Abstract Factory, Prototype, memento
Operations	Template Method, State, Strategy, Command, Interpreter
Extensions	Decorator, Iterator, Visitor

Intention d'interface

- L'interface d'une classe est l'ensemble des méthodes et attributs qu'une classe permet aux objets d'autres classes d'accéder.
- Facade fournit une interface simplifié à un groupe de sous-systèmes ou un sous-système complexe.
- Bridge se concentre sur l'implémentation d'une abstraction

Le patron Bridge

Bridge découple une abstraction ou une interface de son implémentation afin que les deux puissent varier indépendamment.

Motivation

Développement d'une *to Do* liste pour notre tablette, et on veut de la flexibilité sur sa présentation (liste d'items, contact avec puce, nombre).

On veut pouvoir changer les fonctionnalités de la liste de base ; ajout d'une possibilité de tri, de priorité..

Pour cela on développe un groupe de classes de liste, chacune fournissant une façon pour afficher la liste et organiser les informations.

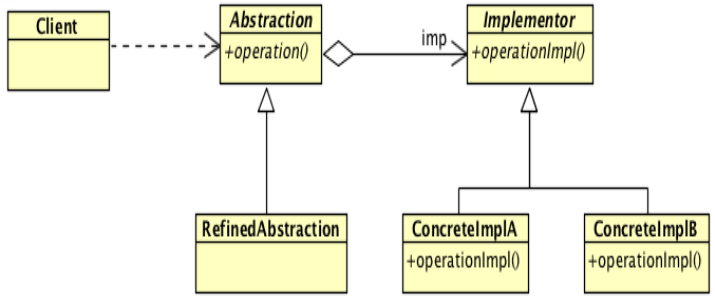
Mais le nombre de combinaisons devient impraticable.

Il est préférable de séparer la représentation de la *To Do* liste de son implémentation.

Quand utiliser le patron Bridge

- Pour éviter un lien permanent entre une structure abstraite et son implémentation concrète.
- Les abstractions et les implémentations doivent être extensibles par sous-classage.
- Le sous-classage est approprié mais on veut gérer les deux aspects du système séparément.
- Tout changement de l'implémentation doit être invisible des clients
- Pour cacher les détails d'implémentation aux clients. Les changements dans l'implémentation devraient de ne pas avoir d'impact sur les clients

Structure du pattern Bridge



Les rôles des classes

- **Abstraction** (abstract class) : définit l'interface abstraite et maintient la référence sur l'implémenteur
- **RefinedAbstraction** (normal class) : étend l'interface définie par l'Abstraction
- **Implementor** (abstract class or interface) : définit l'interface pour les classes d'implémentation
- **ConcreteImplementor** (normal class) :hérite de Implementor ou implémente l'interface Implementor

Bénéfices et défauts de Bridge

Bénéfices

- L'implémentation peut être sélectionnée ou échangée à l'exécution.
- L' abstraction et l'implémentation peuvent être étendues ou composées indépendamment.

Défauts/conséquences

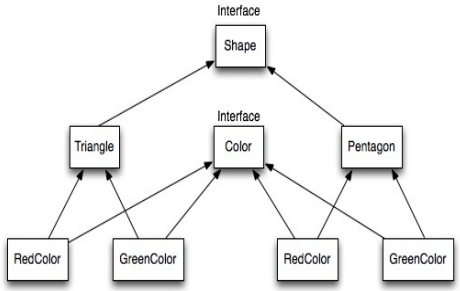
- Il peut y avoir une double indirection
- Il est important de définir proprement quelles responsabilités appartiennent à l'abstraction fonctionnelle et lesquelles appartiennent à la classe d'implémentation interne

Variantes et patterns reliés

- **Automatic Bridge** : implémentation faite pour varier sans action de l'utilisateur final
- **Shared implementatio**n : des classes d'implémentation partagées par plusieurs objets d'application.
- **Single implementation** : parfois il y a une seule classe d'implémentation qui sert à de multiples classes d'abstraction
- Patterns reliés :
  - Adapter des structures similaires mais des intentions différentes.
  - Singleton : utile quand les classes d'implémentation sont partagées.
  - Flyweight : quand la structure d'arbre devient grande pour aider à réduire le nombre d'objets gérés par l'arbre.

Exemple du pattern Bridge

Des formes géométriques auxquelles on souhaite associer des couleurs.
Mélange de hiérarchies à la fois dans les interfaces et dans les implémentations.



Exemple : solution avec Bridge

Utilisation de Bridge pour découpler interface et implémentation
Qui joue le rôle de l'abstraction ?
Qui joue le rôle d'mplémentation ?

Pattern d'architecture MVC

Model-View-Controller (original)

- Le pattern architectural « Model-View-Controller » divise une application interactive en trois composants.
- Le « modèle » : les fonctionnalités de base et les données .
- Les « vues » affichent les informations à l'utilisateur.
- Les « contrôleurs » gèrent les entrées de l'utilisateur.
- Un mécanisme de propagation des changements assure la cohérence entre l'interface utilisateur (view + controller) et le modèle.

Exemple

- Considérons un système d'information pour des élections politiques avec une représentation proportionnelle.
- On a un tableur pour entrer les données et plusieurs sortes de tableau et de graphe représentent les résultats courants.
- Les utilisateurs peuvent intervenir avec le système via une interface graphique.
- Tous les affichages d'information doivent immédiatement refléter les changements des données du vote.
- Il doit être possible d'ajouter de nouveaux moyens de représentation, par exemple l'affectation des sièges de parlementaires.
- Le système doit aussi être portable sur d'autres plates-formes ou d'autres standards d'affichage « look and feel ».

Contexte

Applications interactives avec une interface homme-machine flexible.

Problème

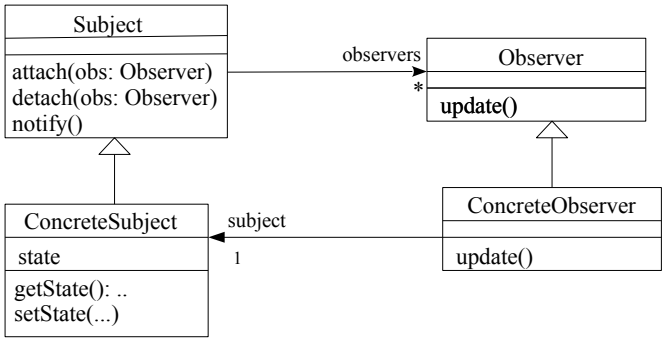
- Quand vous étendez les fonctionnalités d'une application, vous devez modifier les menus pour accéder à ces nouvelles fonctions. Les clients peuvent vouloir des adaptations spécifiques etc ...
- Les forces qui influencent la solution :
  - La même information est présentée différemment dans différentes fenêtres.
  - L'affichage et le comportement de l'application doivent refléter immédiatement les manipulations de données.
  - Les changements de l'interface utilisateur doivent être faciles, et même possibles lors de l'exécution.
  - Le support de différents standards de « look and feel » ou le portage de l'interface utilisateur ne doit pas affecter le code dans le noyau de base de l'application.

Solution

- MVC divise une application interactive en processus, sortie et entrée.
- Model
  - Le composant modèle encapsule les données et les fonctionnalités du noyau de base.
- Views
  - les composants vue affichent l'information à l'utilisateur. Une vue obtient les données du modèle.
  - Chaque vue possède un composant contrôleur associé.
- Controllers
  - Les contrôleurs reçoivent les entrées, comme des événements qui encodent les mouvements de la souris, l'activation des boutons de la souris ou les entrées au clavier.
  - Les événements sont traduits en des requêtes de service pour le modèle ou la vue.
  - L'utilisateur interagit avec le système seulement via les contrôleurs.

Propagation : Patron Observer

Le mécanisme de propagation des changements est décrit par le pattern *Publisher-Subscriber* appelé aussi le pattern *Observer*.



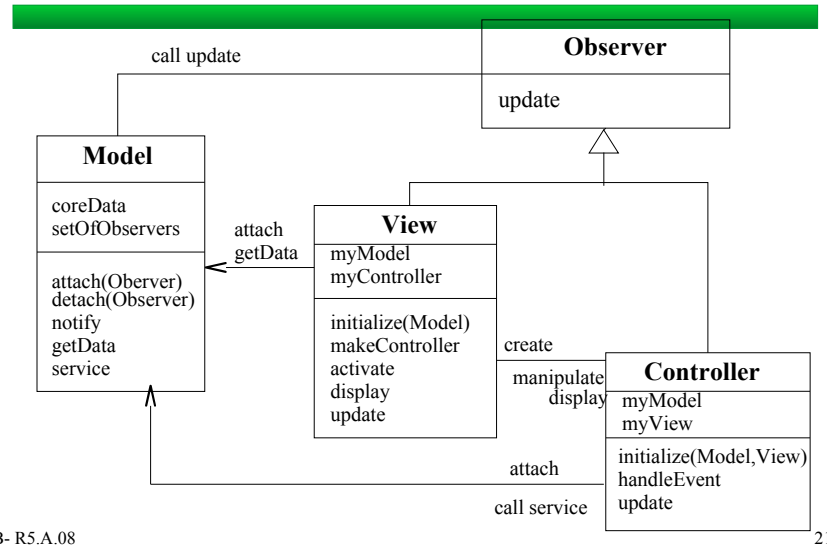
Structure

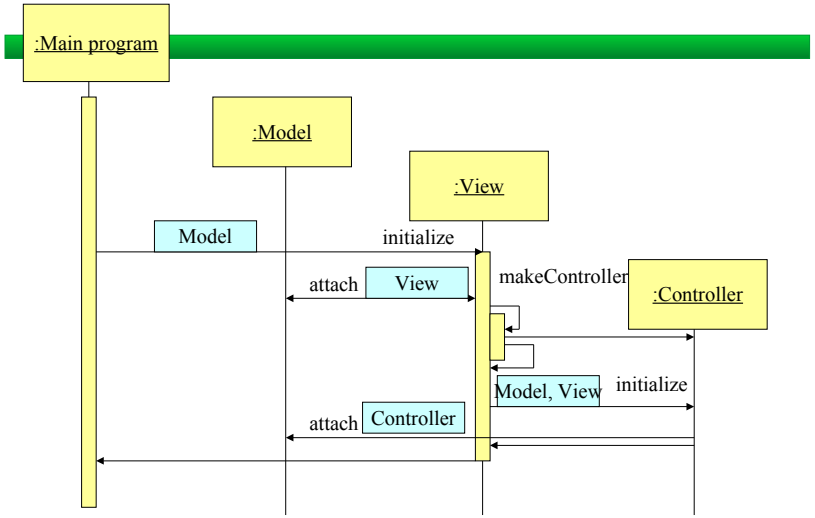
Classe Model	**Collaborateurs**
Responsabilités • Fournit les fonctionnalités de l'application • Enregistre les vues dépendantes et les contrôleurs • Avertit les composants dépendants des changements de données	• View • Controller

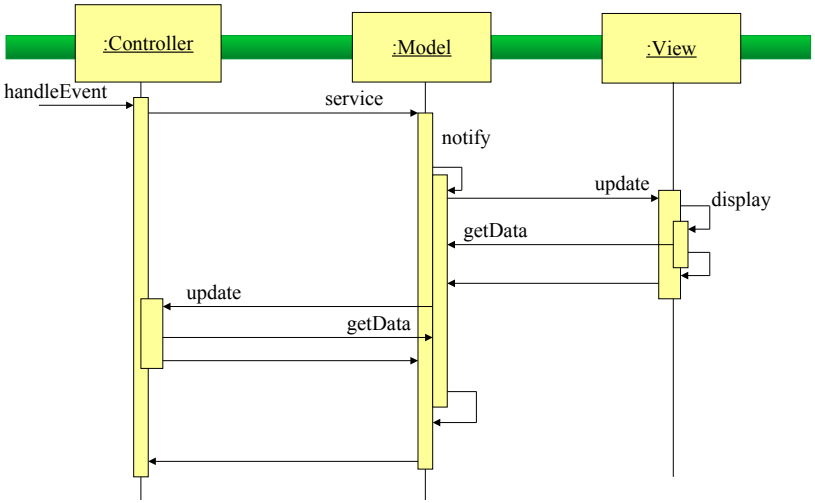
Classe View	**Collaborateurs**
Responsabilités • Crée et initialise son contrôleur associé • Affiche l'information pour l'utilisateur • Implémente les procédures de mise à jour • Recherche les données dans le modèle	• Controller • Model

Classe Controller	**Collaborateurs**
Responsabilités • Accepte les entrées de l'utilisateur comme des événements • Traduit les événements en requêtes de service pour le modèle ou affiche les requêtes pour la vue. • Implémente les procédures de mise à jour si nécessaire	• View • Model

Architecture générale





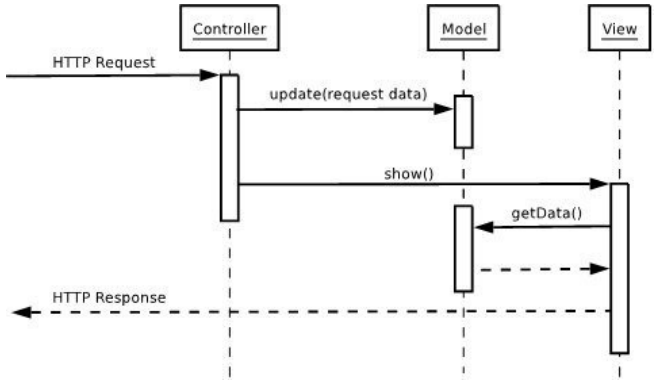


Implémentation

- étape 1: Séparer l'interaction homme-machine du noyau fonctionnel.
- étape 2: Implémenter le mécanisme de propagation de changements.
- étape 3: Concevoir et implémenter les vues.
- étape 4: Concevoir et implémenter les contrôleurs.
- étape 5: Concevoir et implémenter la relation vue-contrôleur.
- étape 6: Implémenter l'initialisation du MVC.
- étape 7: Interaction dynamique des vues.
- étape 8: contrôleurs enfichables (pluggable).
- étape 9: Infrastructure pour les vues et les contrôleurs hiérarchiques.
- étape 10: Découpler encore plus les dépendances du système.

Framework MVC et applications Web

↳ Traitement d'une requête



Evolution de la notion de framework MVC

Ce qu'il doit contenir aujourd'hui :

- L'organisation MVC proprement dite
- Un système d'URL propres, cohérentes et compréhensibles
- L'intégration d'un framework javascript (ergonomie riche)
- Un système de mapping semi-automatique « objets<=> BD »
  - ORM (Object-Relationa Mapping)
- La gestion de plusieurs templates pour les vues
- Des systèmes permettant l'aide au développement

MVC est devenu une norme

Utilisé partout dans le monde du développement :

- ASP.NET MVC
- QT en C++
- Backbone.js en JavaScript
- Angular en TypeScript
- Spring et Struts en Java
- Django avec Python
- ….