L'architecture MVC : Modèle – Vue - Contrôleur

Génie Logiciel
Licence 3 - Université du Havre
Bruno Mermet

Le Design Pattern[1] <<Observer>>

1. voir http://users.info.unicaen.fr/~bmermet/Enseignement/CoursPDF/designPatterns.pdf

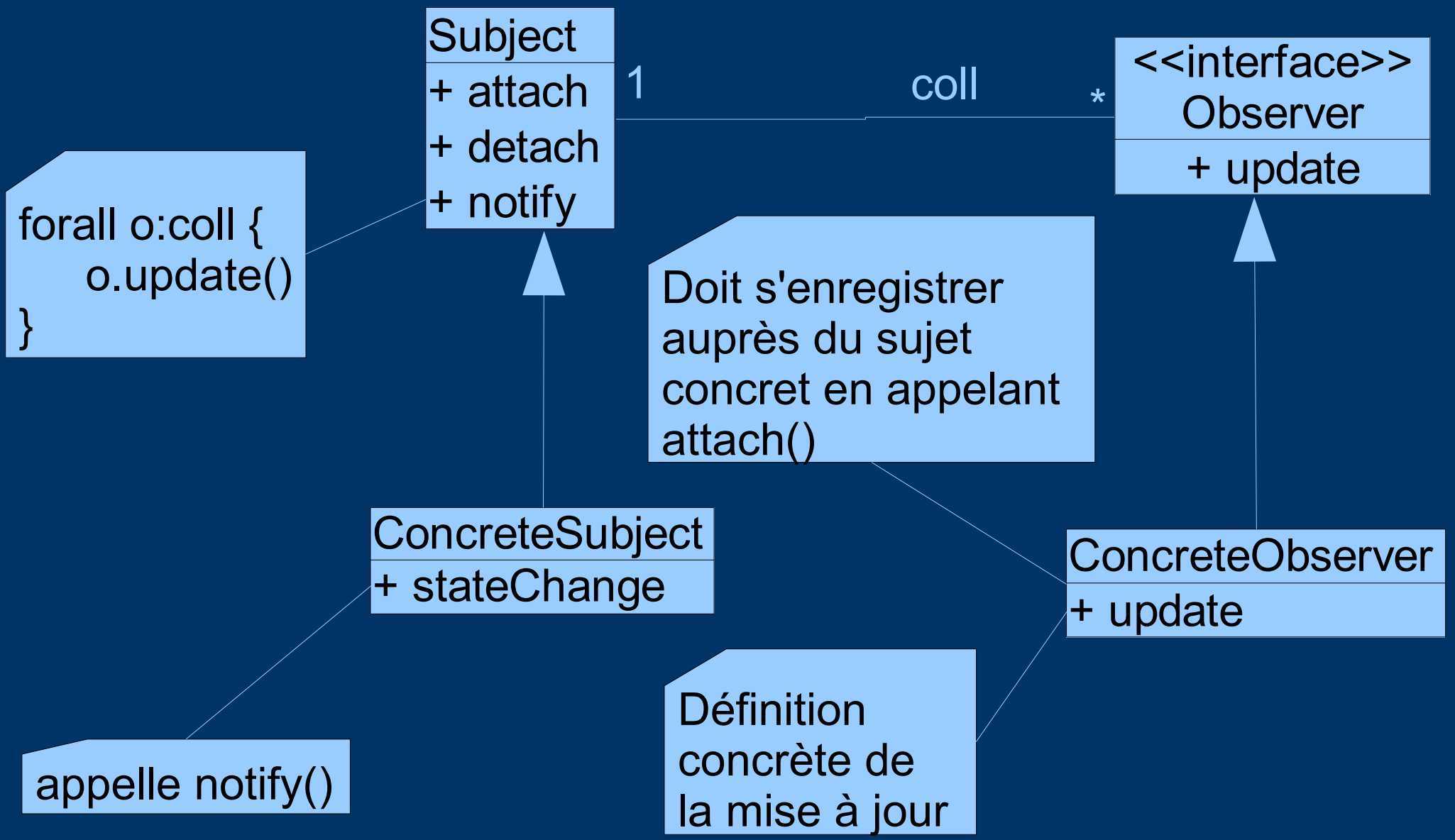
Présentation générale

- Objectif

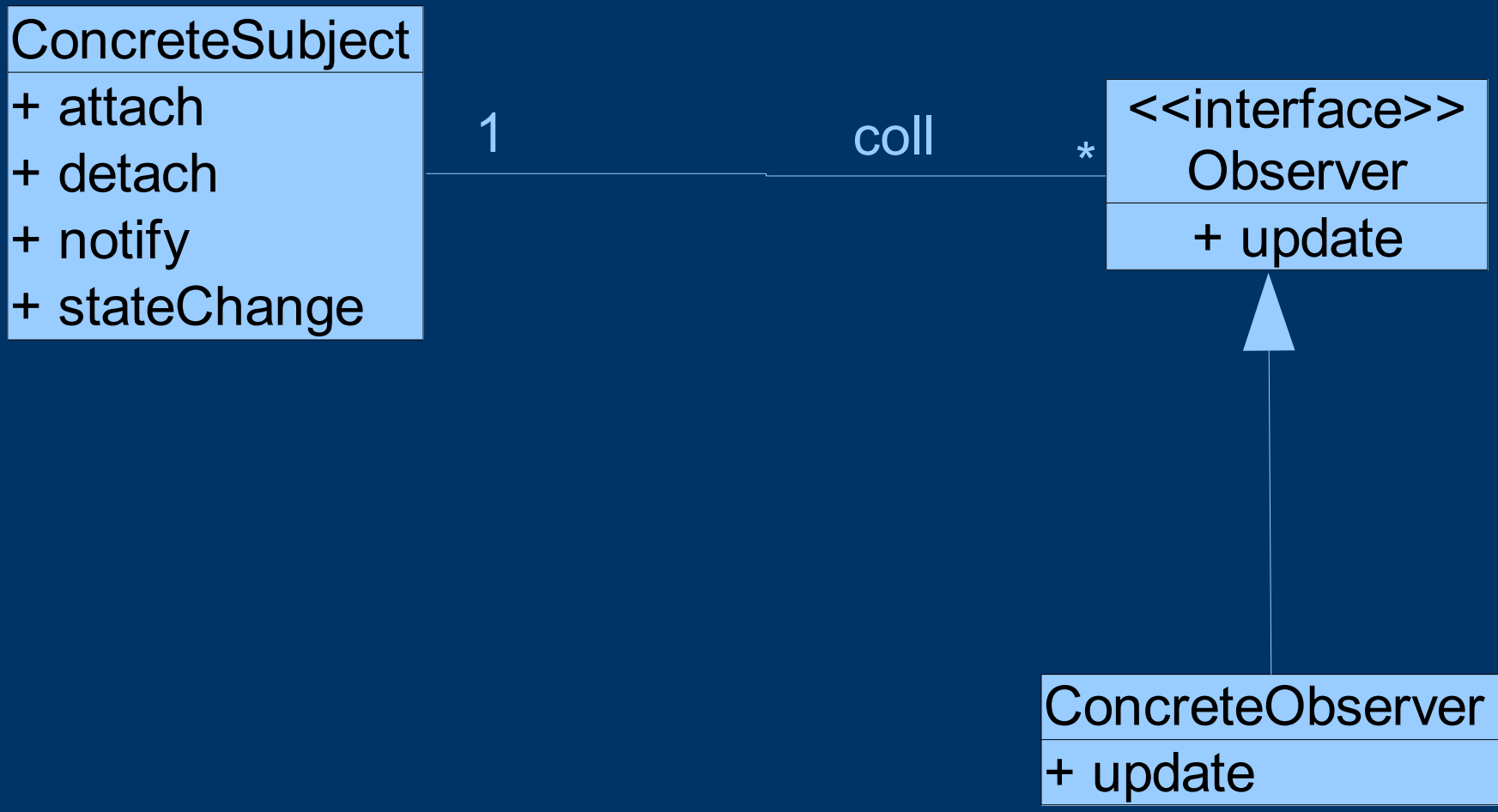
  Permettre à un objet de prévenir un ensemble d'autres objets inconnus au moment de sa conception de certains changements de son état

- Problème
  - Certains changements d'état d'un objet O sont susceptibles d'en intéresser d'autres
  - Ces autres objets sont inconnus lors de la conception de O
- Solution
  - Faire gérer à O une liste d'*observateurs* (ou *écouteurs*) et les prévenir lors des changements d'états intéressants

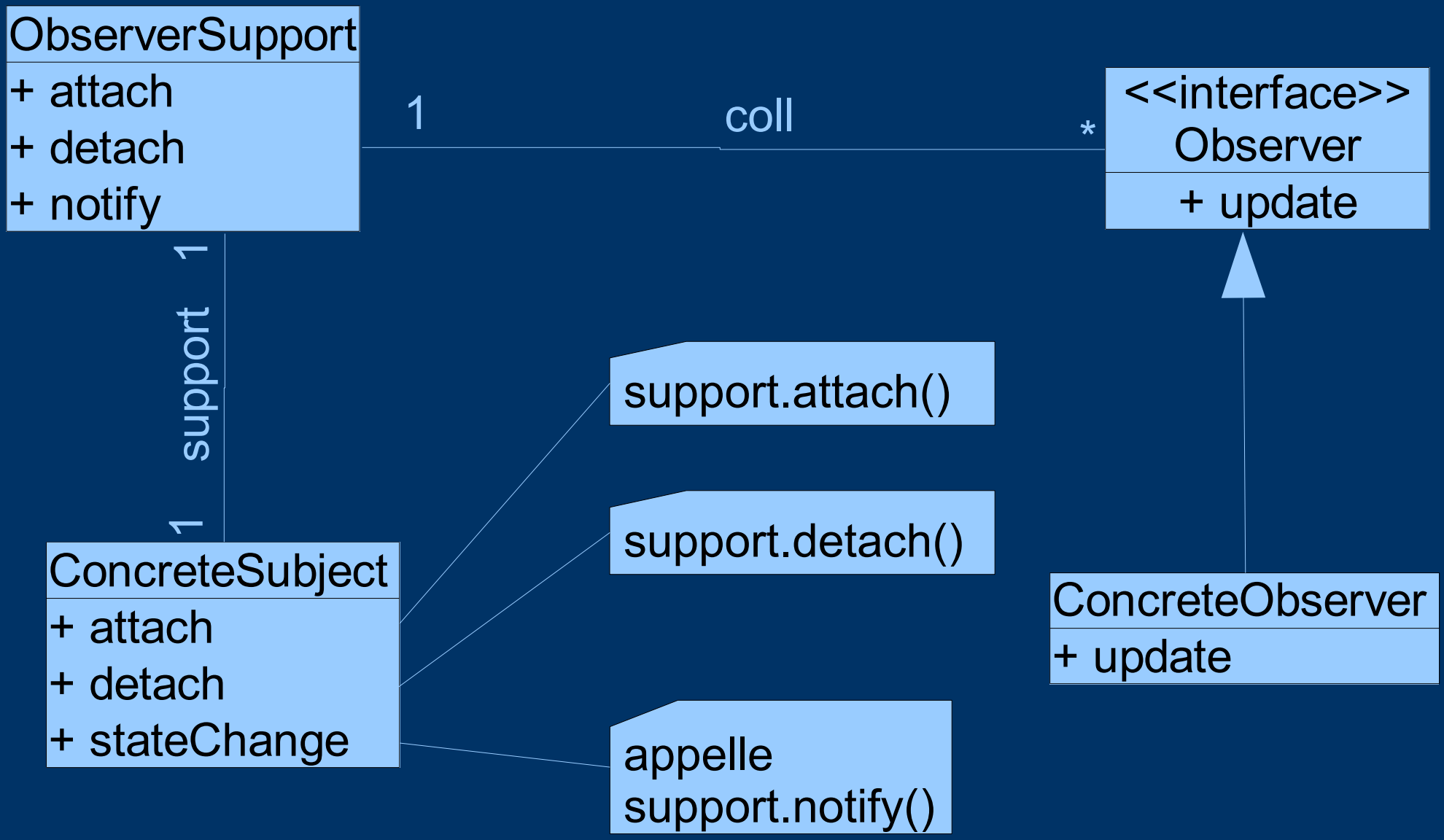
Architecture générale



Architecture simplifiée



Architecture à réutilisation



L'architecture MVC

Principe général

- But
  - Isoler la donnée elle-même de sa présentation
  - Distinguer la consultation de la modification
- Principe

  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

- Rôle
  - Encapsuler les propriétés d'une donnée
  - Être indépendant des vues et contrôleurs
- Conséquences
  - Définir les accesseurs aux propriétés de cette donnée
  - Maintenir une liste d'écouteurs (vues et contrôleurs se déclarent comme écouteurs)
  - Prévenir les écouteurs lorsque la donnée est modifiée
  - ➔ Implantation du design pattern *Observer*

La vue

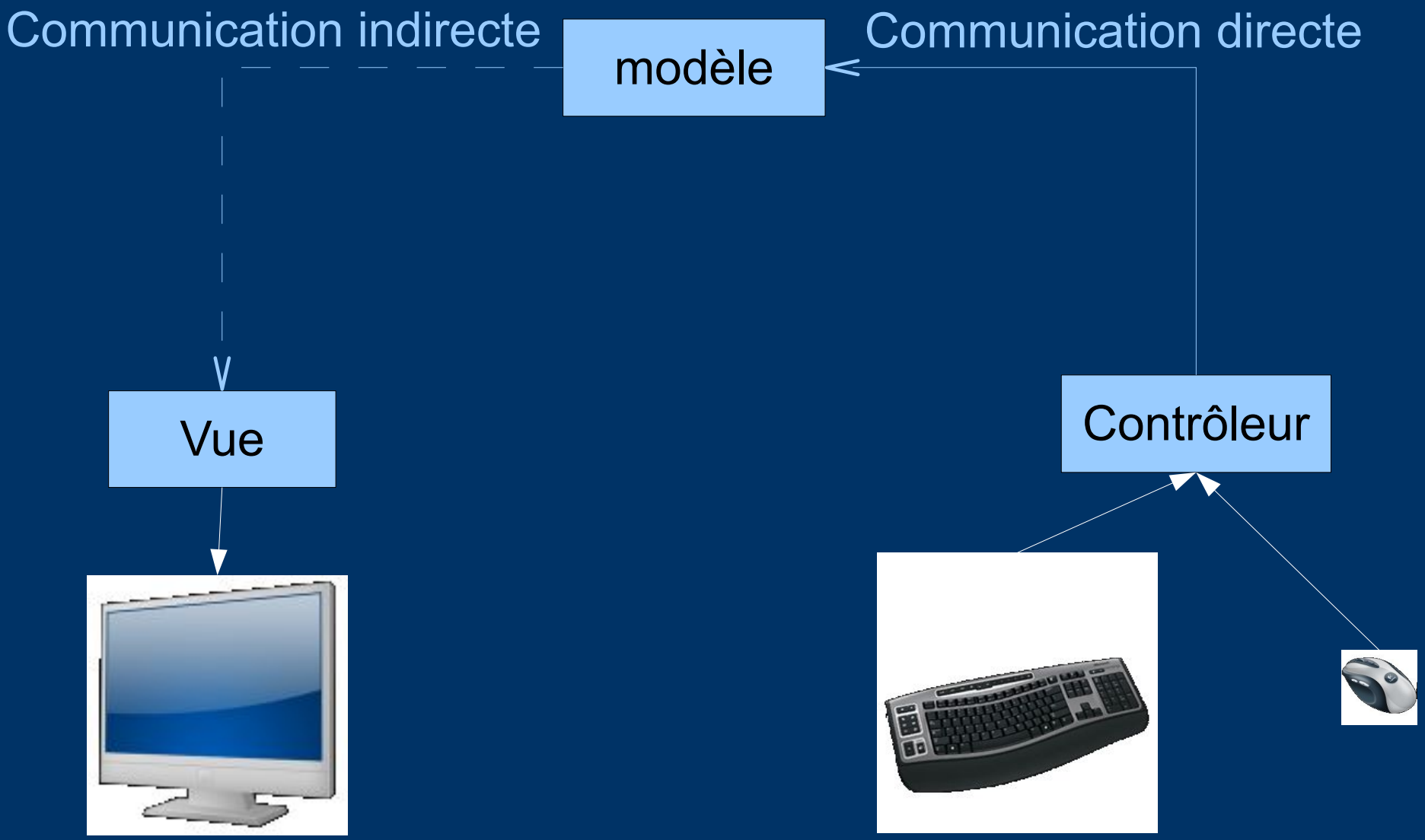
- Rôle
  - Représenter la donnée encapsulée via un modèle
  - Se maintenir à jour lorsque le modèle est modifié
- Conséquences
  - Doit s'enregistrer comme écouteur au niveau du modèle

Le contrôleur

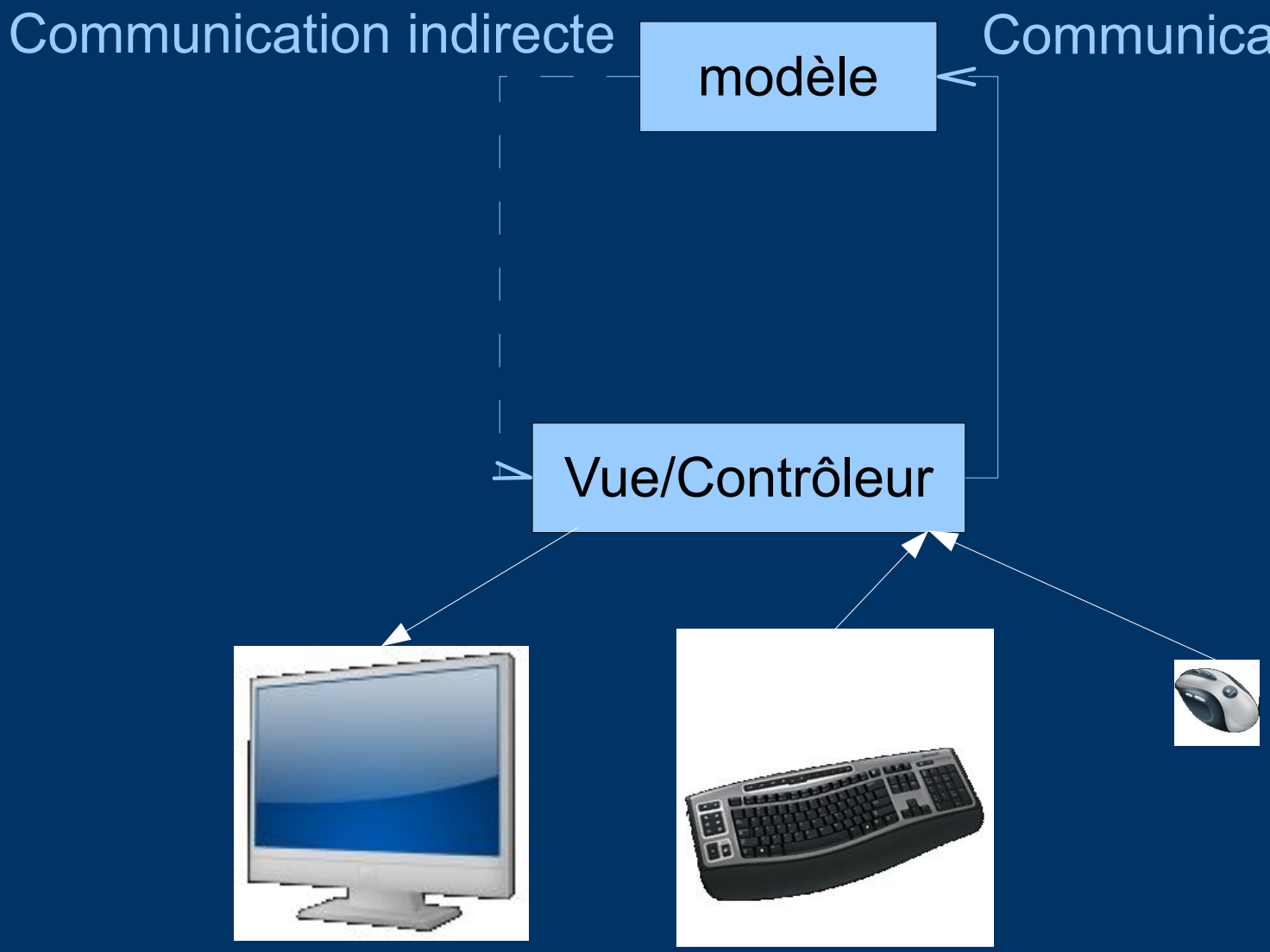
- Rôle
  - Permettre à l'utilisateur de modifier la donnée encapsulée dans le modèle
- Conséquences
  - Doit éventuellement s'enregistrer comme écouteur du modèle pour être mis à jour si le modèle est modifié
  - Doit appeler les accesseurs du modèle en fonction des actions de l'utilisateur

MVC et Swing

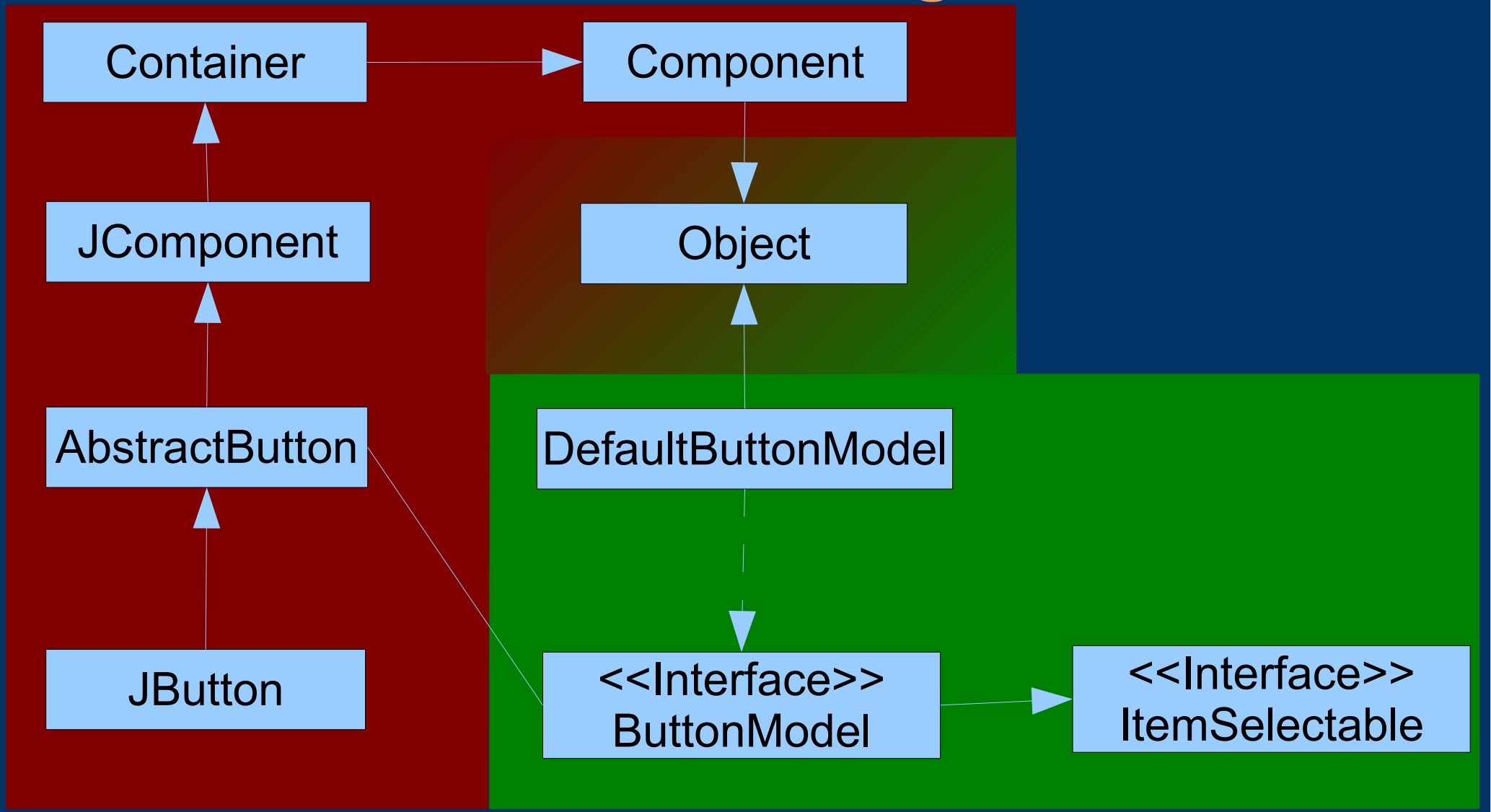
Architecture générale



Architecture swing : vue et contrôleurs fusionnés



Exemple de mise en oeuvre : Les <<boutons>> Swing



Exemple de mise en oeuvre : L'interface ButtonModel

getSelectedObject

addItemListener	removeItemListener
addActionListener	removeActionListener
addChangeListener	removeChangeListener

getActionCommand	setActionCommand
getMnemonic	setMnemonic

isArmed	setArmed
isEnabled	setEnabled
isPressed	setPressed
isRollOver	setRollOver
isSelected	setSelected

Exemple de mise en oeuvre : La classe JButton (extrait)

addActionListener	removeActionListener	fireActionPerformed
addChangeListener	removeChangeListener	fireStateChanged
addItemListener	removeItemListener	fireItemStateChanged

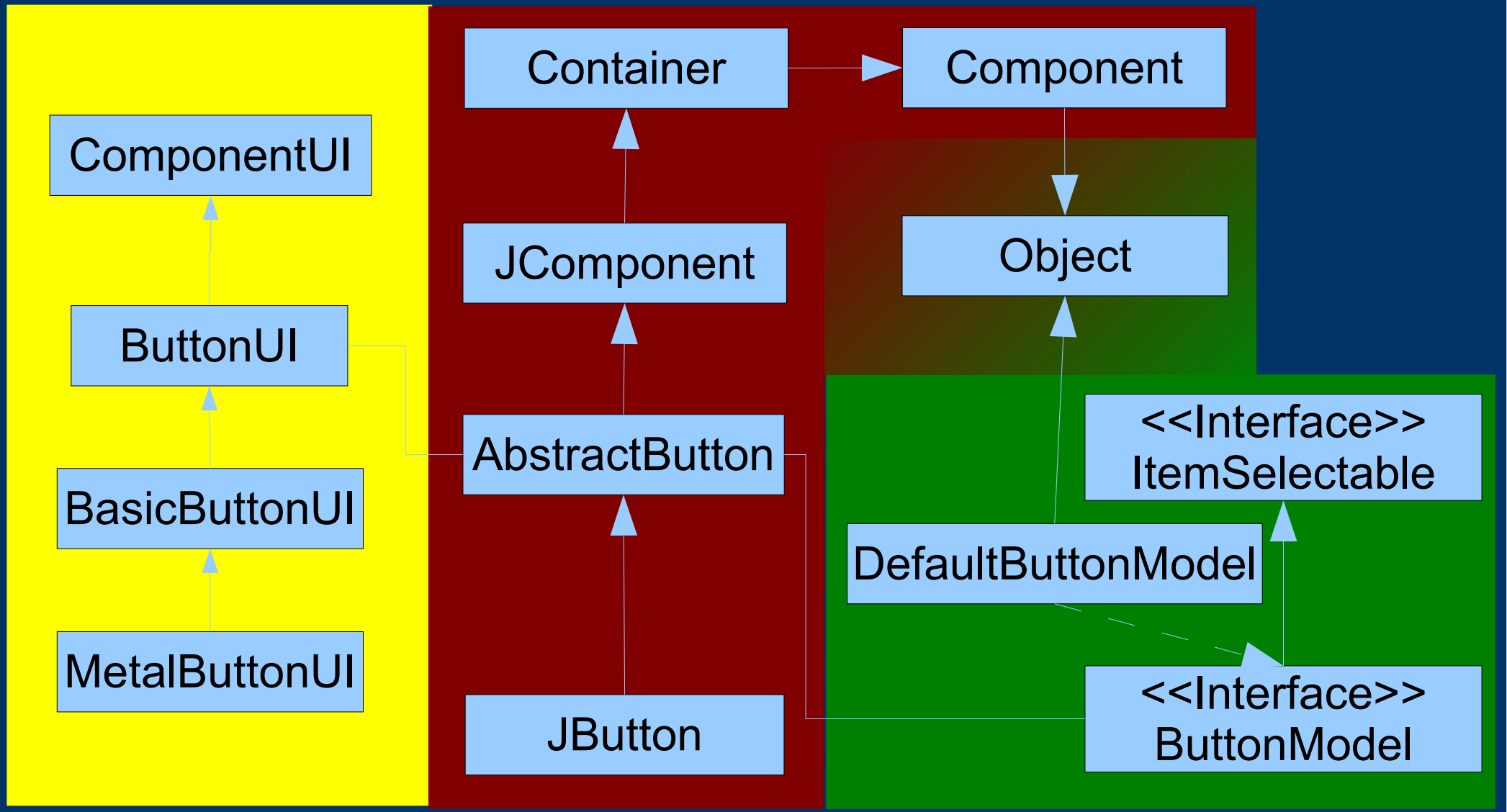
getIcon	setIcon	
getText	setText	
getLabel	setLabel	
getFont	setFont	
paint	repaint	update

isSelected	setSelected
isEnabled	setEnabled

MVC et Swing
Look-and-feel : principes

- Apparence personnalisable ("look and feel")
  - → Affichage d'un composant délégué à une autre classe ("UI delegate")
  - – Exemple
    - Pour un bouton JButton, affichage délégué à une sous-classe de ButtonUI, comme MetalButtonUI
- Implantation
  - – Classes UI transparentes si on ne s'y intéresse pas
    - → Une telle classe est construite par défaut par l'UIManager
  - – Le bouton JButton est un observateur de ce que détecte le ButtonUI et retransmet les infos à ses propres écouteurs

MVC et Swing
Look-and-feel : architecture



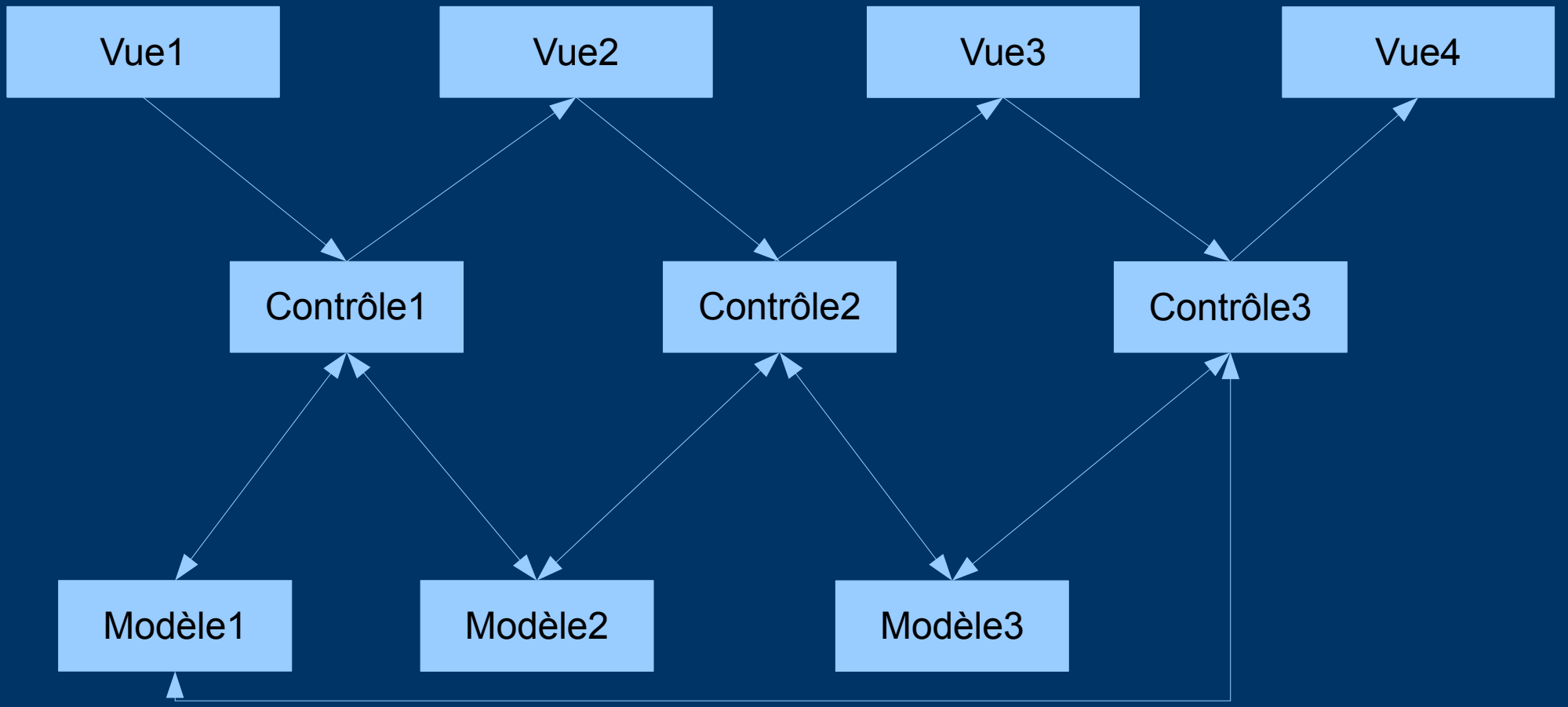
MVC et applications web

MVC1

- Modèle
  - Les données (Java Beans)
- Vue
  - La page web/JSP
- Contrôleur
  - Une servlet de contrôle qui
    - Traite l'info de la page source en modifiant les données si nécessaire
    - Affiche la nouvelle page

➔ La vue sert à configurer le contrôle

MVC1 : exemple



Du MVC1 au MVC2

- Inconvénient MVC1 :Multiplication du nombre de Servlets
  - Lourdeur de déclaration au niveau du Serveur d'Application
  - Mises à jour peu aisées
- Modèle MVC2
  - Une seule Servlet de contrôle servant d'aiguillage et à l'architecture simple :

```
switch(pageSource) {
    case page1 : pageSuite=traitement1();afficher(pageSuite);break;
    case page2 : pageSuite=traitement2();afficher(pageSuite);break;
    ...
}
```

  - En pratique : une classe par traitement

MVC2 : exemple