



# Chapitre 4

## Patrons d'architecture

### 2I1AC3 : Génie logiciel et Patrons de conception

Régis Clouard, ENSICAEN - GREYC

« L'informatique n'est pas plus la science des ordinateurs  
que l'astronomie n'est celle des télescopes. »  
Michael R. Fellows et Ian Parberry

## ■ Patron d'architecture

- Se réfère à l'organisation structurelle de l'application.
- D'un niveau d'abstraction supérieur au patron de conception.

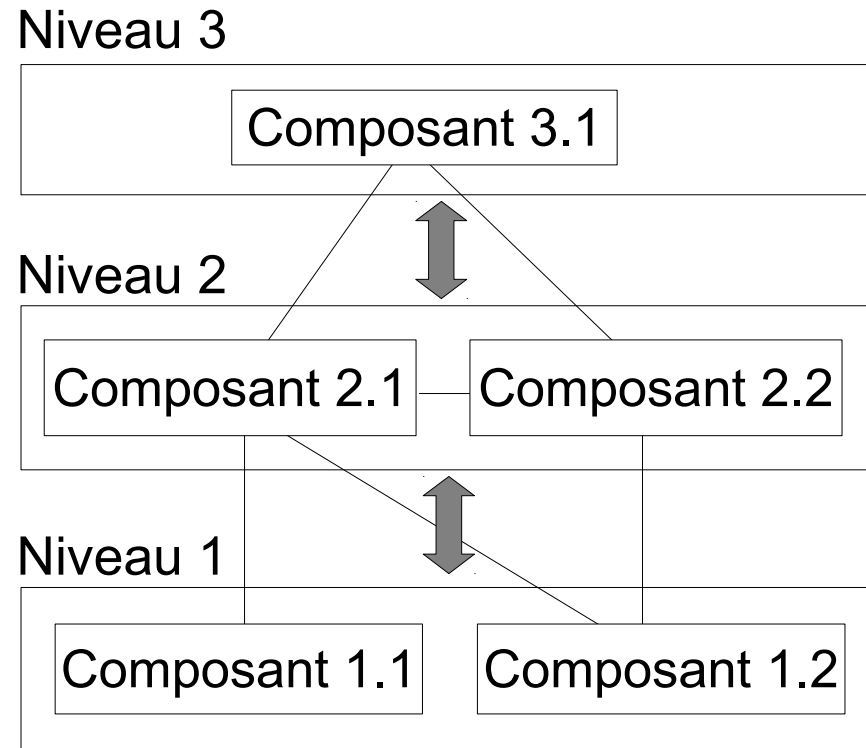
## ■ Préoccupations

- La limitation des performances du matériel.
- La haute disponibilité (ie. rapport entre le temps de fonctionnement réel d'un système et celui espéré).
- La minimisation des risques d'entreprise.

## ■ Enjeux

- Développabilité
- Testabilité
- Réutilisabilité

## ■ Organisation verticale



## ■ Quand

- Plusieurs niveaux d'abstraction dans les responsabilités
  - ▶ Les couches supérieures : interactions avec les utilisateurs.
  - ▶ Les couches basses : répondre aux requêtes.

■ Exemple : *modèle OSI*

## ■ Recommandation

- Un niveau est formé de composants réutilisables dans les mêmes conditions.
- Les relations d'un niveau à un autre sont décrites par des interfaces.
- Aucun composant ne peut s'étaler sur deux niveaux.
- Les échanges sont limités entre deux couches consécutives.

## ■ Avantages

- Niveaux réutilisables et interchangeable.
- Dépendances uniquement locales entre deux niveaux consécutifs.
- Les développeurs et utilisateurs de chaque niveau peuvent ignorer les autres niveaux. Tout passe par les interfaces.

# 04

## ex. Trois étages (*Three-tier*)

5

### Chapitre

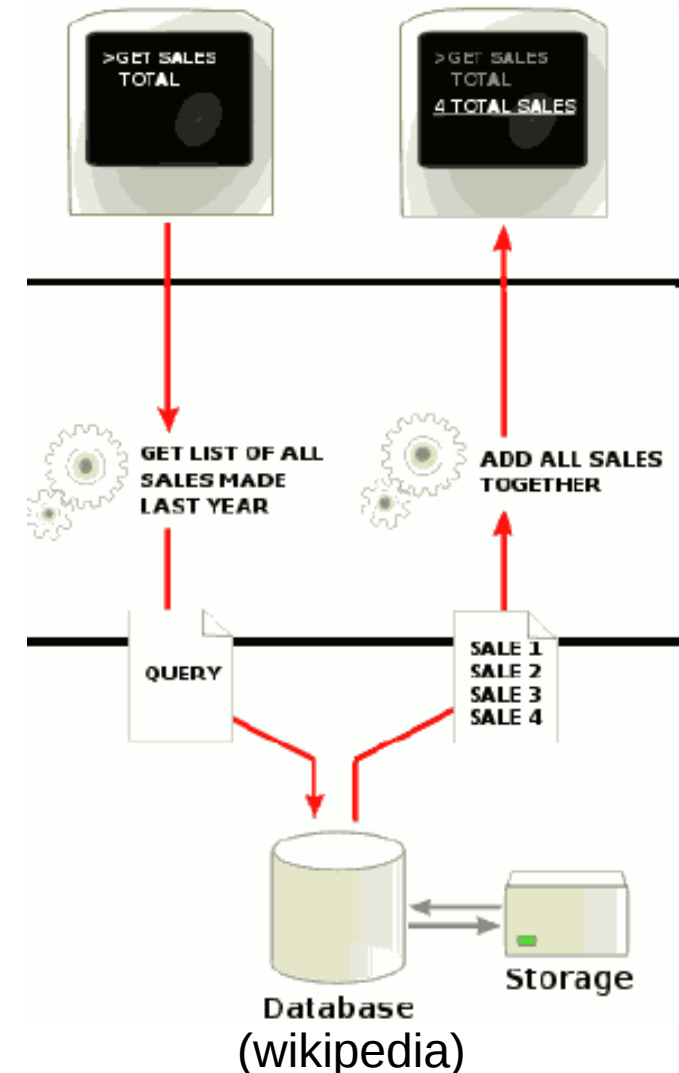
#### ■ Organisation verticale (cas particulier du n-tier)

- Niveau **présentation** (interface utilisateur)
  - ▶ Visualise les données.
- Niveau **logique** (règles métier)
  - ▶ Coordonne les décisions et évaluations logiques, et effectue les calculs.
- Niveau **données** (ORM)
  - ▶ Les données sont stockées et extraites de bases de données.

#### ■ Quand

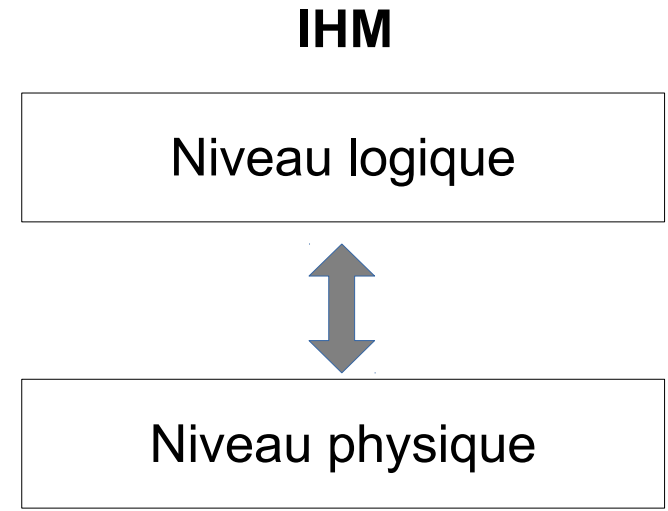
- Architecture client-serveur présentant une interface utilisateur.

#### ■ Exemple : *sites Web de commerce électronique*



## ■ Organisation verticale

- Niveau logique
  - ▶ Présentation des données aux utilisateurs.
- Niveau physique
  - ▶ Implémentation des données.



Matériel

## ■ Quand

- Pour des raisons d'efficacité, la représentation des données diffère de la logique métier.
- Il n'y a pas ici de notion de hiérarchisation, mais une recherche d'efficacité.

## ■ Exemple : *Système de gestion de bases de données*

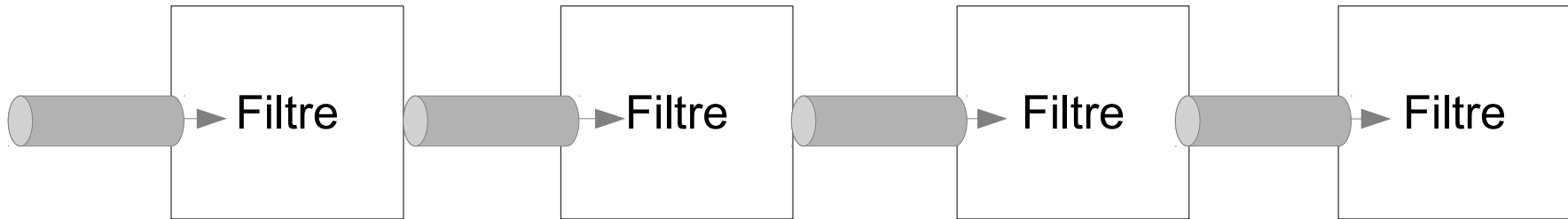
### ■ Recommandation

- Nécessite une forte correspondance entre les niveaux.
- La correspondance passe par des interfaces.

### ■ Avantages

- Présente aux utilisateurs une interface conforme à la logique métier.
- Utilise la meilleure structuration des données.
- Cache la complexité de la représentation physique aux utilisateurs.

## ■ Organisation horizontale



## ■ Quand

- Les fonctionnalités procèdent par traitement des données en flux.
  - ▶ Les données passent d'un filtre à un autre par des tubes.
  - ▶ Chaque étape de traitement est encapsulée dans un composant filtre.

■ Exemples : *Compilateur, Chaîne de traitement d'images*



## ■ Recommandation

- La communication entre filtre passe par des interfaces.

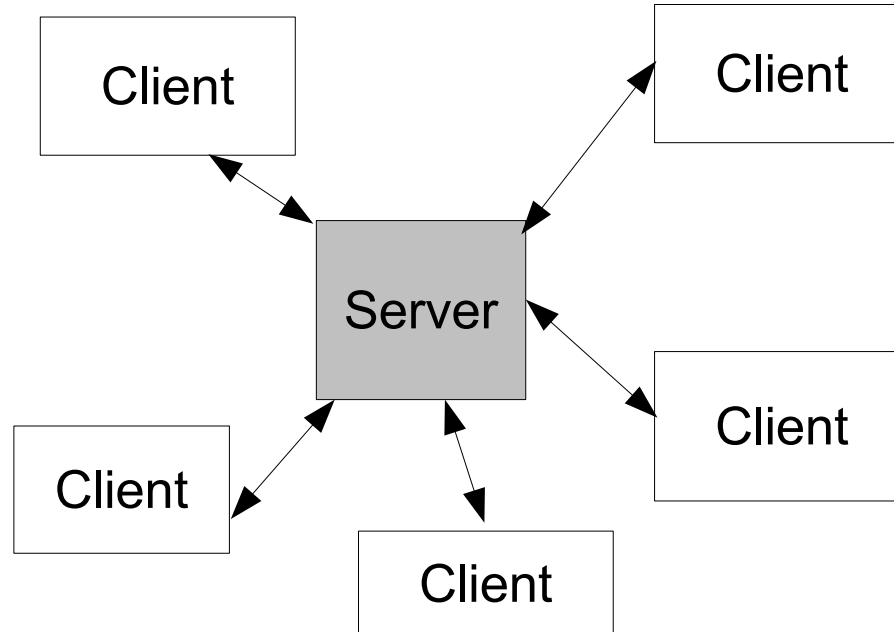
## ■ Caractéristiques

- Plus de relation hiérarchique.
- Responsabilités bien identifiées.

## ■ Avantages

- Les contraintes de communication sont limitées à deux composants.
- Il est facile de remplacer un filtre.
- Facile d'implanter la concurrence entre filtres.

## ■ Organisation centralisée



## ■ Quand

- Systèmes distribués qui interagissent par invocation de services localisés.
  - ▶ Le serveur propage les requêtes.
  - ▶ Le serveur relaye les résultats et les erreurs.

■ Exemple : *Guichet Automatique Bancaire*

## ■ Variation

- Clients lourds / Clients légers.
- Multi-serveurs : un Broker fait l'intermédiaire pour localiser le serveur qui possède le service.

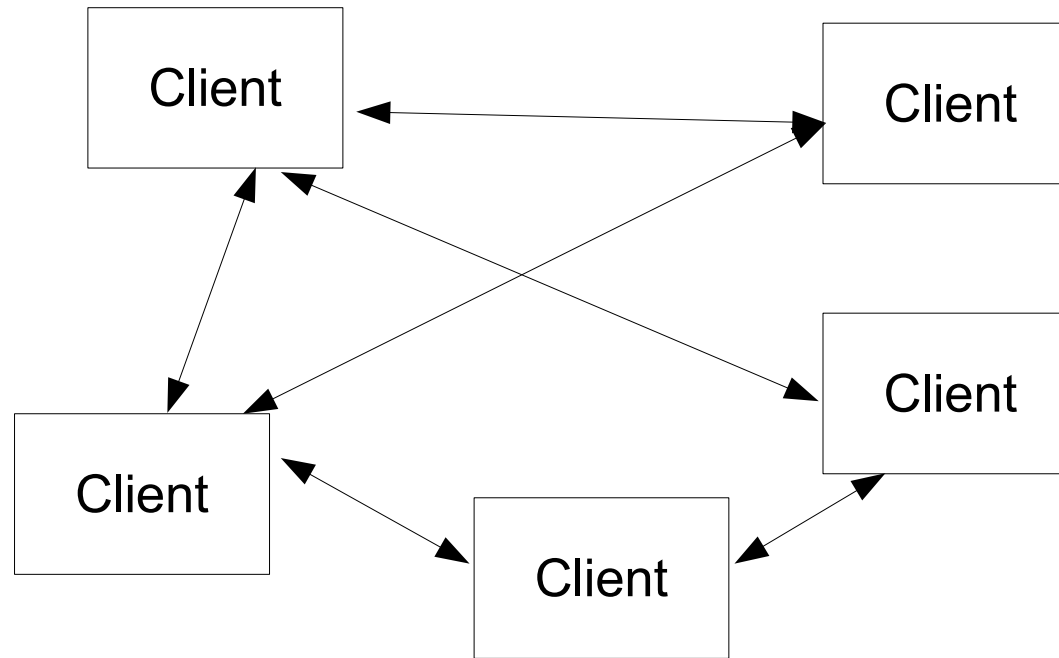
## ■ Avantages

- Indépendance des composants pour le fonctionnement et le développement.
  - ▶ Côté serveur / Côté client.
- Localisation transparente.

## ■ Inconvénients

- Faible tolérance aux erreurs.
- Sensible à la montée en charge.
- Difficile à tester.

## ■ Organisation décentralisée



## ■ Quand

- Constat : plus une donnée est demandée moins elle est disponible avec une architecture client-serveur.
- Inverser la tendance : chaque client qui a récupéré la donnée devient un serveur.

## ■ Exemple : BitTorrent

## ■ Recommandations

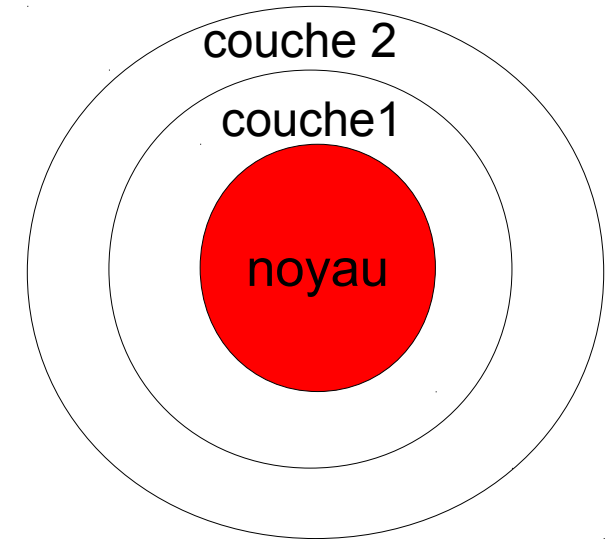
- Si la données est volumineuse, elle est divisée en segments.
- Un client / serveur ne distribue que des segments.

## ■ Avantages

- Tolérance aux fautes.
- Rapidité de communication.
- Autorise le parallélisme.
- Plus une donnée est demandée plus elle est disponible.

## ■ Organisation en couches

- Noyau fonctionnel
  - ▶ Encapsule tous les services fondamentaux.
- Services externes
  - ▶ Utilisent les couches inférieures comme interface.



## ■ Quand

- Le système doit être adaptable au remplacement ou au changement de l'environnement d'accueil (p. ex. architecture matériel, OS).

## ■ Exemples : *Windows NT, JVM*

## ■ Avantages

- Extensibilité.
- Portabilité : seul le noyau a besoin d'être changé.

## ■ Organisation en boucle

### ● **Modèle : backend**

- ▶ Stocke les données et l'état de l'interface.
- ▶ Détient le modèle du domaine et la logique métier.

### ● **Vue : front end**

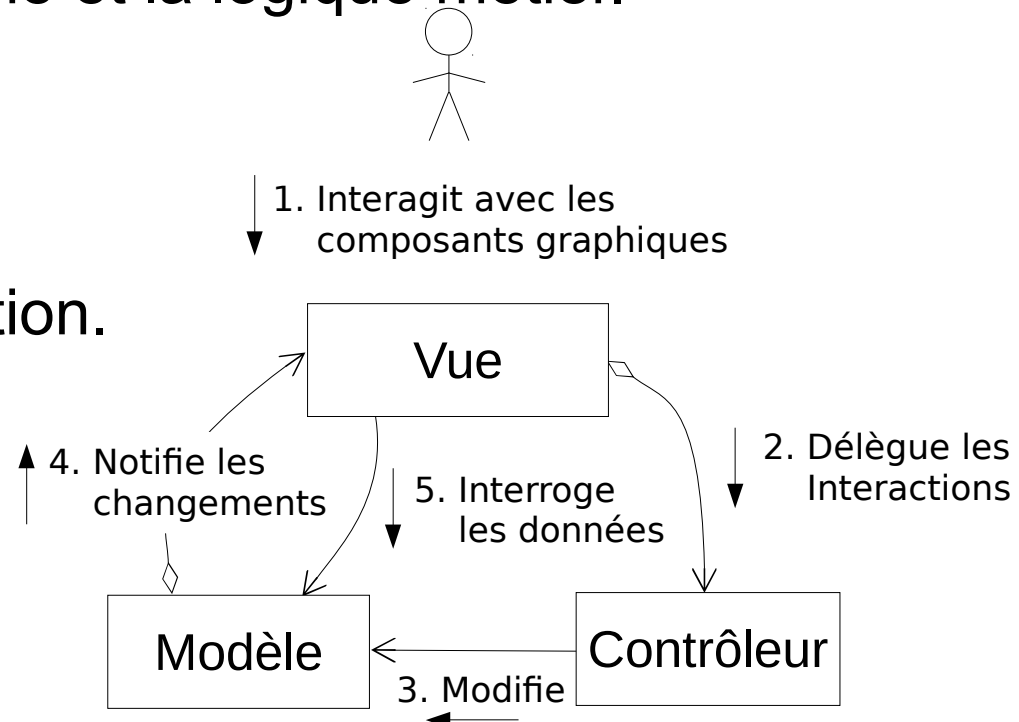
- ▶ Affiche les informations aux utilisateurs.
- ▶ Détient logique de présentation.

### ● **Contrôleur : front end**

- ▶ Gère les interactions avec les utilisateurs.

## ■ Quand

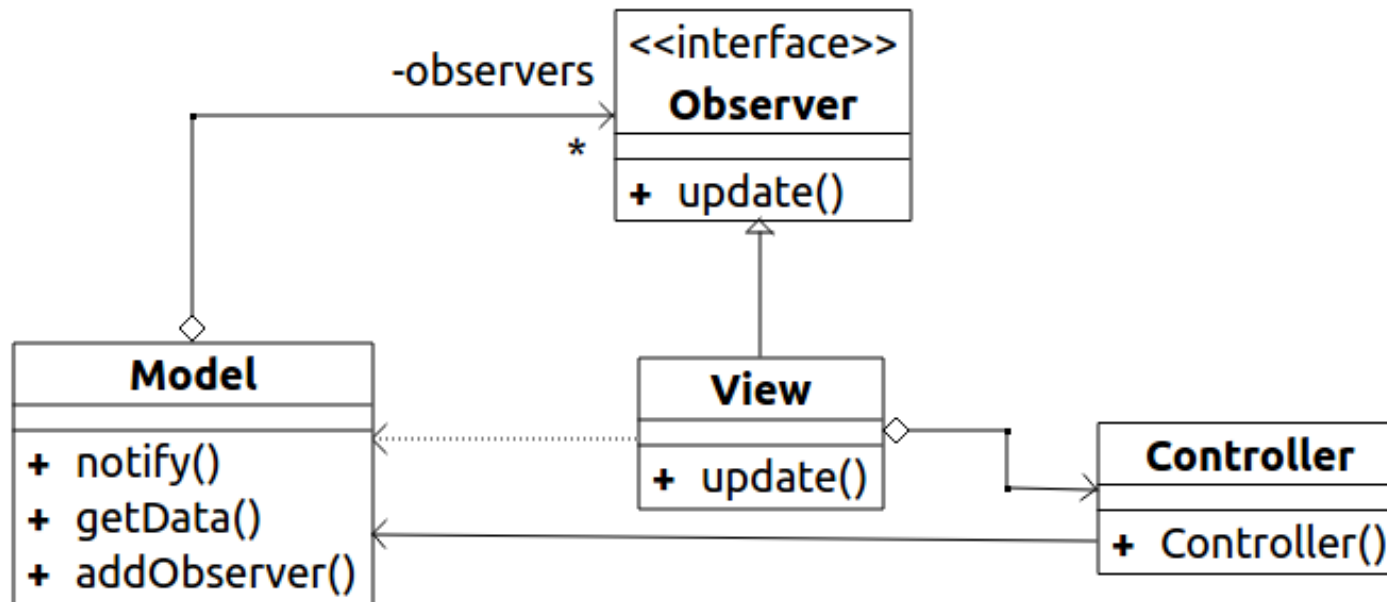
- Une interface graphique multi-fenêtrée (GUI).





## ■ Découpler les vues du modèle : Observateur

- Ajouter un protocole de souscription / notification pour les vues au modèle.



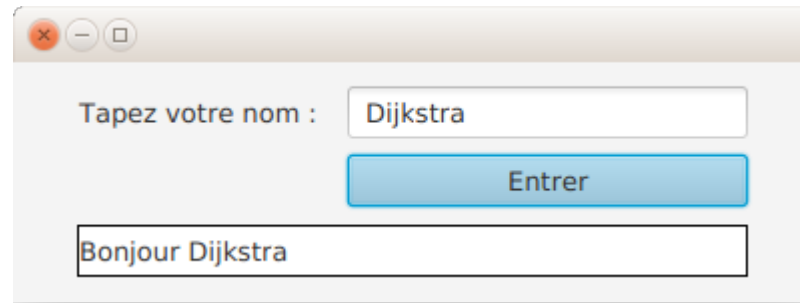
# 04

## Exemple d'un écran de connexion

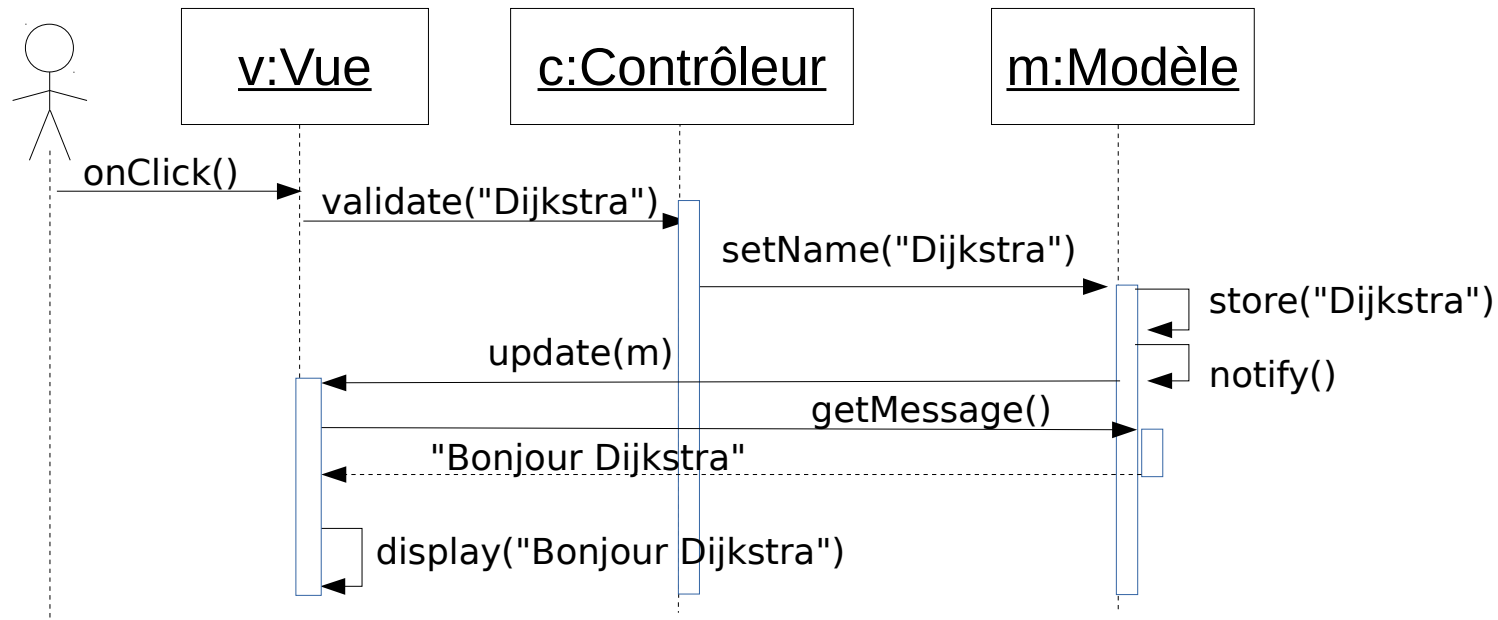
18

### Chapitre

- L'utilisateur renseigne le nom puis appuie sur le bouton « Entrer » et l'écran affiche un message de bienvenue.



- Diagramme de collaboration



## ■ Avantages

- Découplage entre les vues et le modèle, qui peuvent évoluer indépendamment.
- Plusieurs vues sur le même modèle.
- Une vue peut être ajoutée facilement.

## ■ Inconvénients

- Mauvaise séparation des responsabilités
  - ▶ La vue a 2 responsabilités : affichage et récupération des données.
  - ▶ Qui détient l'état de l'interface : contrôleur, modèle ou vue ?
- Difficilement testable (inter-dépendance des unités).

## ■ Conclusion

- Ce patron est peu utilisé en pratique pour les GUI.
- On lui préfère des variantes (Model2) ou des patrons dérivés (MVP et MVVC).

## ■ Variante du modèle MVC

- Réduit le couplage Vue - Modèle

▶ **Modèle**

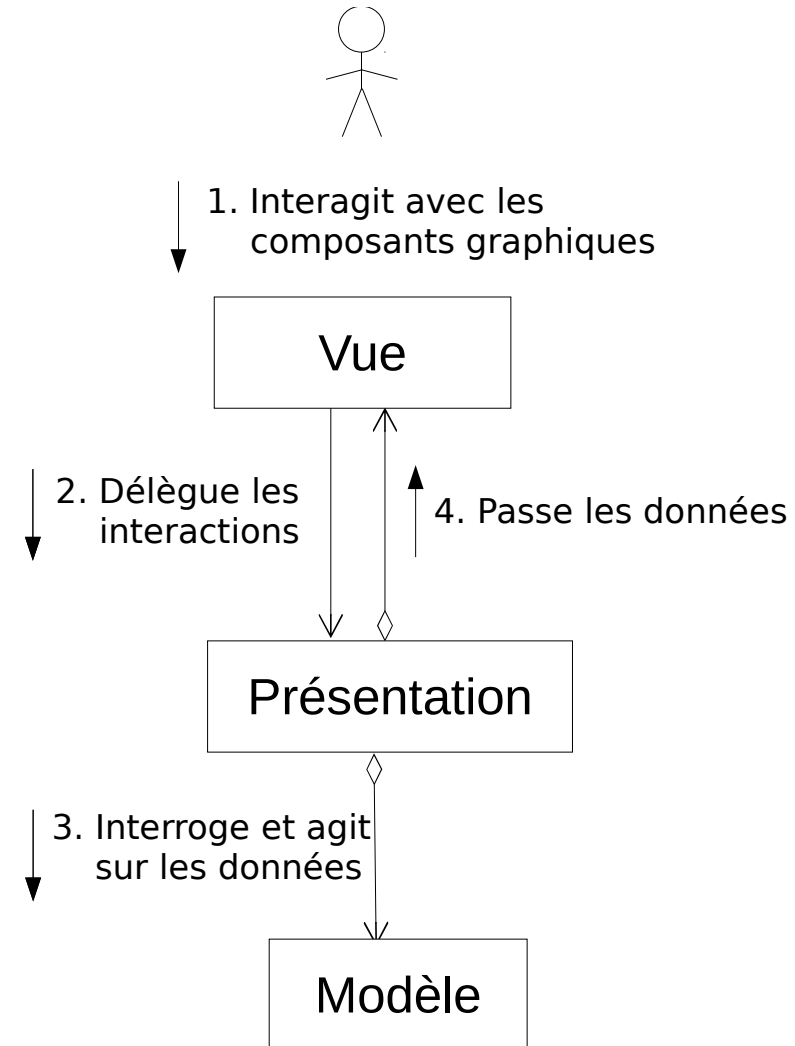
- Stocke les données.
- Modèle du domaine et logique métier.

▶ **Vue**

- Passive : ne contient aucune logique interne.

▶ **Présentation**

- État de l'interface.
- Logique de présentation.

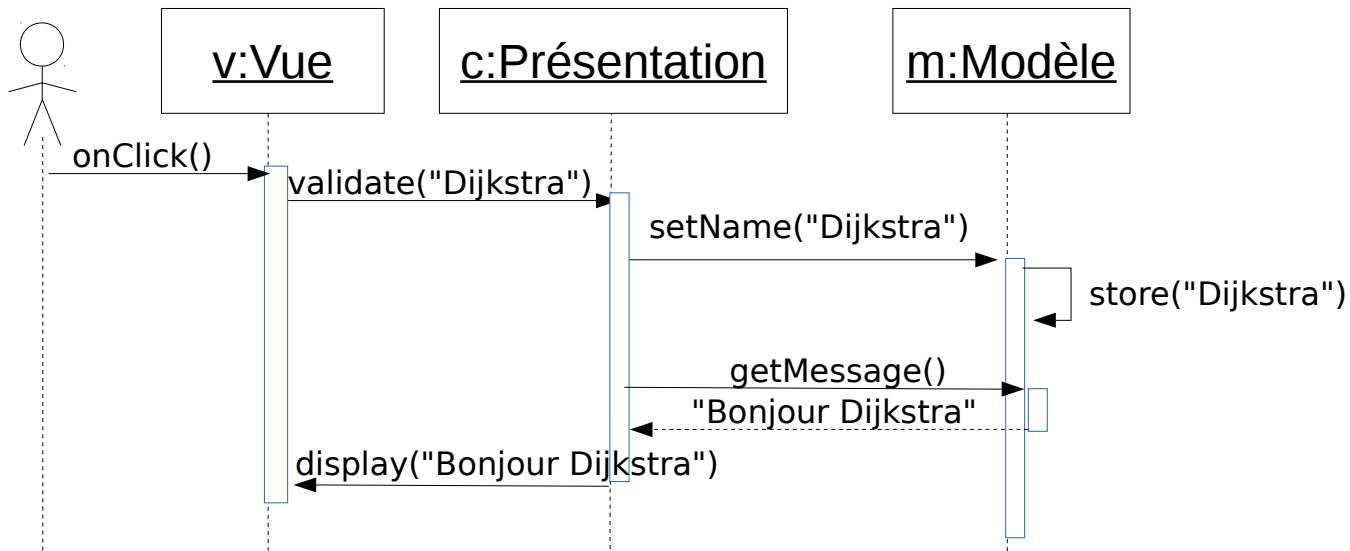
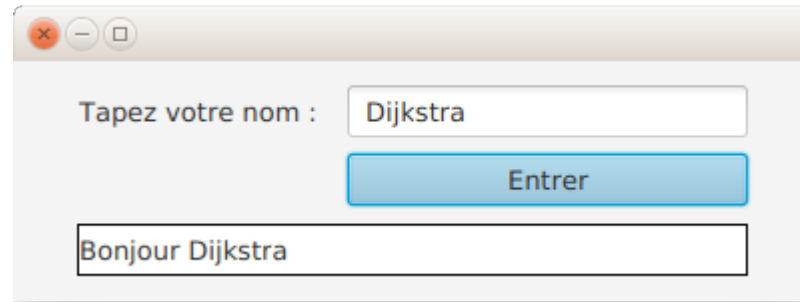


# 04

## Chapitre

# Exemple d'un écran de connexion

21



## ■ Avantages

- Élimine l'interaction entre la vue et le modèle parce qu'elle sera effectuée par le biais de la présentation, qui organise les données à afficher dans la vue.
- Tout est testable sauf la vue mais elle est dépourvue de logique.
- Modèle est indépendant des autres composants.

## ■ Inconvénients

- Beaucoup de code redondant (*boilerplate code*) pour mettre à jour les données dans la vue.

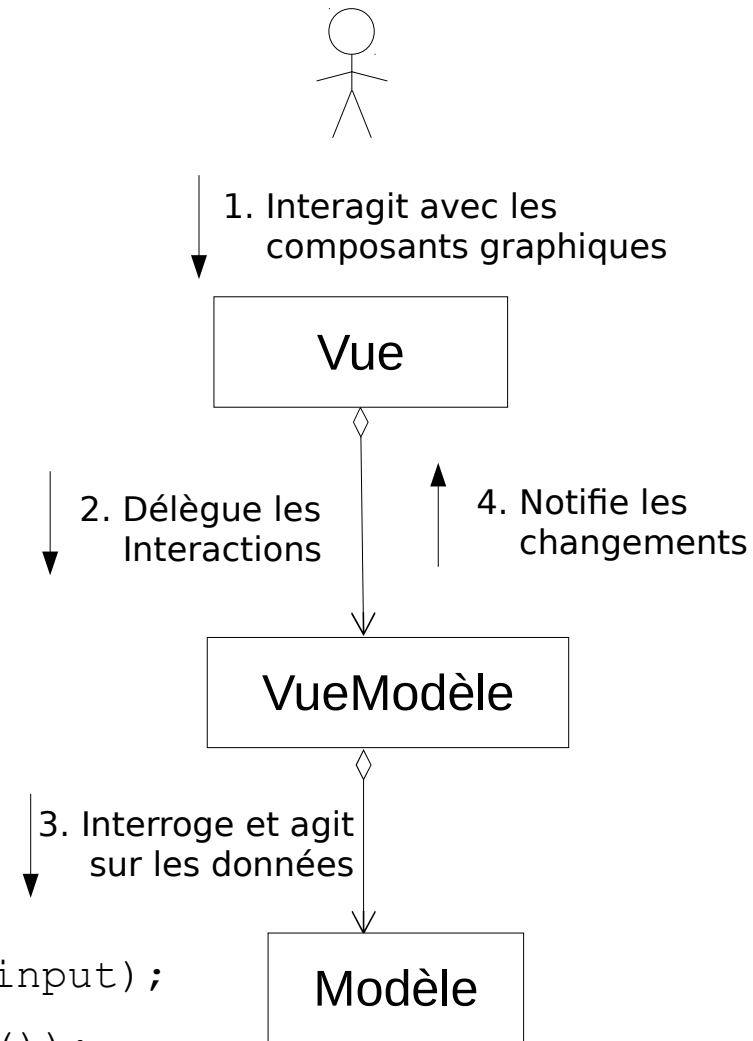
## ■ Variante du modèle MVP

- Supprimer le code redondant dans la présentation.

## ■ Unique différence avec MVP

- La mise à jour de la vue est faite par un mécanisme de liaison (*binder*) entre les composants de la vue et les attributs de vuemodèle.
- Le binder est réalisé avec le patron Observateur de MVC qui est réintroduit mais au niveau des attributs :

```
label.textProperty().bindBidirectional(viewModel.input);  
circle.scaleXProperty().bind(slider.valueProperty());
```

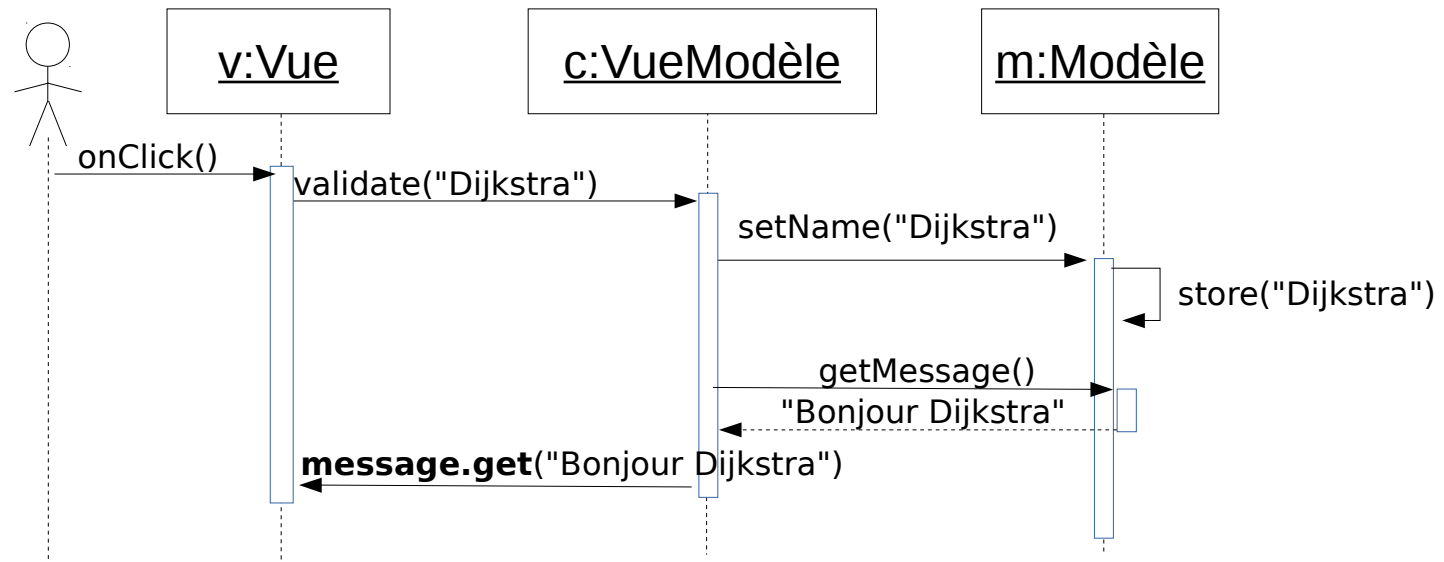
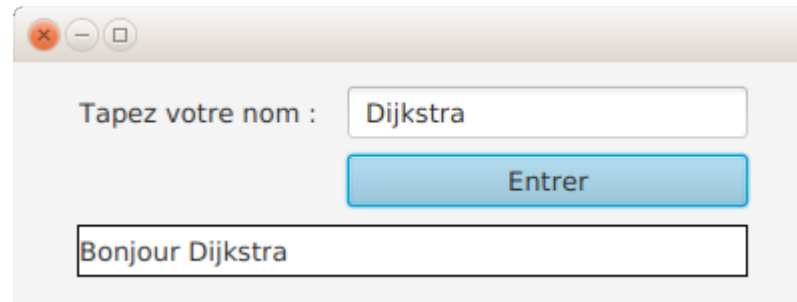


# 04

## Chapitre

# Exemple d'un écran de connexion

24



La méthode `message.get()` est appelée par le mécanisme de liaison (binder). Le label de Vue qui affiche le message est lié à l'attribut `message` de `VueModèle`.



## ■ Avantages

- Exactement les mêmes que pour le patron MVP.
- Élimination du surcoût de code redondant (*boilerplate code*).

## ■ Inconvénients

- Généraliser le modèle de vue pour de grande application est difficile.
- La liaison de données dans de grandes applications peut aboutir à une consommation considérable de ressources.

## ■ Modèle de base pour .NET (avec XAML)

- L'architecture générale doit être choisie très tôt.
  - Premiers temps de l'analyse.
- Elle doit être relativement stable.
  - Elle définit l'organisation du projet.
- Mais elle évoluera pour s'adapter aux particularités de l'application en cours de conception.
  - Une bonne architecture doit permettre de différer les décisions majeures le plus tard possible.
- Utiliser des interfaces comme des pare-feux entre les composants de l'architecture.
- Plusieurs architectures peuvent être combinées pour un même projet.