Patrons d'Architecture

R4.01 - Architecture Logicielle

Luc Klaine

Table des matières

Introduction
Définitions

- **L'Architecture Logicielle** :
  - Définit la **structure des modules** d'un système logiciel.
  - Inclut les **composants logiciels**, les **propriétés externes** visibles de ces composants et les **relations** entre ces composants.
  - Ne décrit pas **ce que doit réaliser le système**.
  - Décrit **comment il doit être conçu** pour à répondre aux spécifications.

- **Les Patrons d'Architecture** :
  - Donnent des **solutions générales** à des **problèmes d'architecture récurrents**.
  - Servent de **référence à l'organisation structurelle** des systèmes.
  - Permettent de **déterminer le framework** du logiciel.

Introduction
Motivations

- **Préoccupations** :
  - Le système est-il interactif ?
  - Nécessite-t-il de fréquents changements ?
  - Le logiciel est-il réparti sur le réseau ?
  - Comment les fonctionnalités sont-elles décomposées en composants ?
  - Quelles exigences non-fonctionnelles sont-elles importantes ?

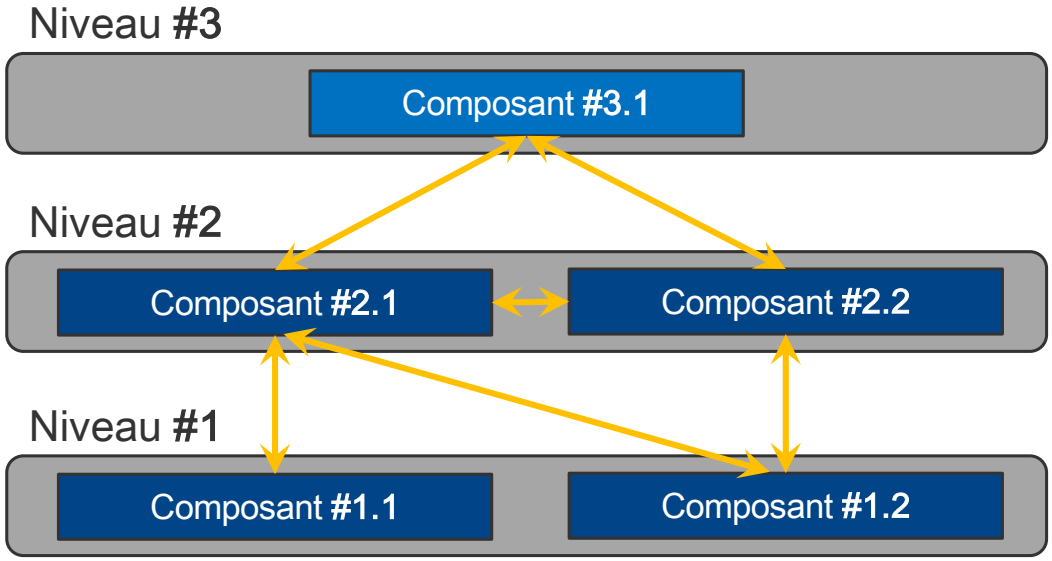
- **Enjeux** :
  - Faciliter le développement et les futurs changements **(développabilité)**.
  - Faciliter la maintenance (**maintenabilité)**.
  - Faciliter les tests **(testabilité)**.
  - Faciliter la réutilisation des composants **(réutilisabilité)**.

Table des matières

Styles architecturaux

1. Architecture en étages (n-tiers)

- **Type d'organisation** : verticale



- **Quand l'utiliser** ?
  - Lorsqu'il existe plusieurs niveaux d'abstraction dans les responsabilités

Styles architecturaux

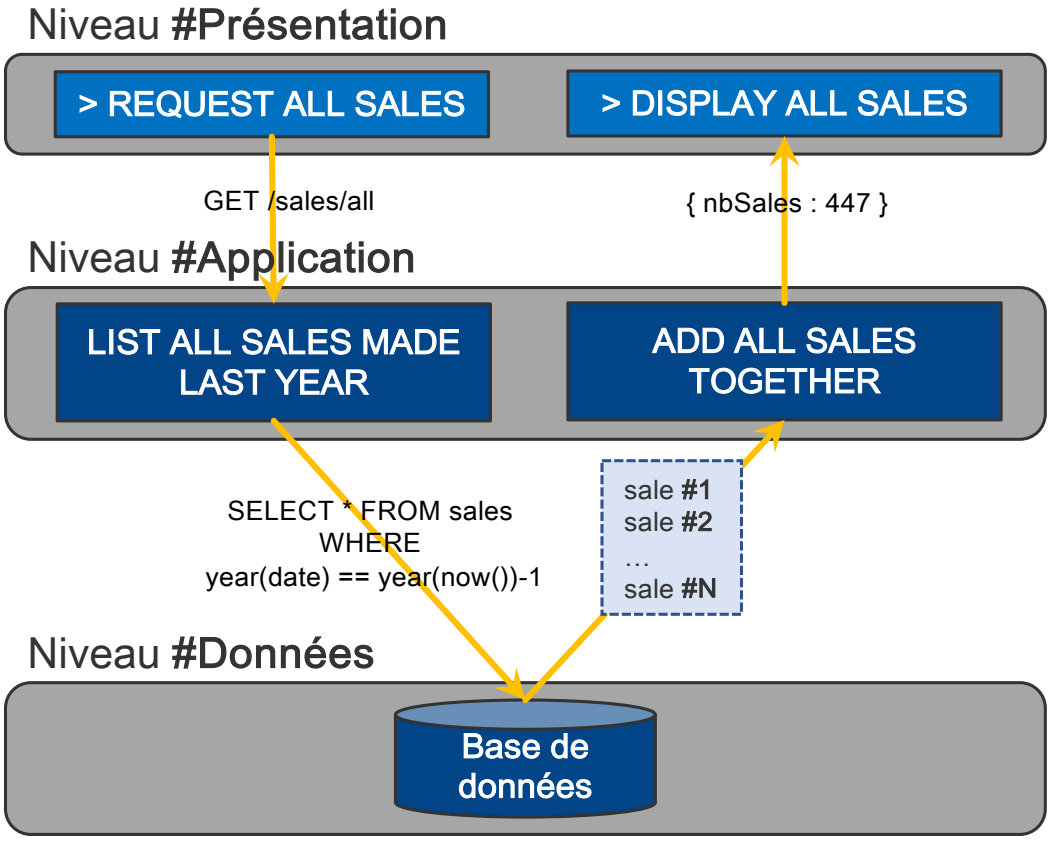
1. Architecture en étages (n-tiers)

- **Principes** :
  - Chaque étage est formé de **composants réutilisables dans les mêmes conditions**.
  - Les **échanges** entre les étages sont décrits par des **interfaces**.
  - Les étages supérieurs interagissent avec les utilisateurs (**requêtes**).
  - Les étages inférieurs opèrent sur les données (**réponses**).
  - Les composants ne doivent s'étaler que sur un étage.
  - Les échanges sont bornés entre deux étages consécutifs.
- **Avantages** :
  - **Réutilisabilité et interchangeabilité des étages**.
  - **Gestion par interface** : les développeurs et les utilisateurs de chaque niveau peuvent ignorer les autres étages.
- **Exemples** :
  - Architecture **Client/Serveur** présentant une interface utilisateur.

Styles architecturaux

1. Architecture en étages (n-tiers)

- **Détail d'une architecture à trois étages** : site de e-commerce



Interface utilisateur :
Réagit aux événements utilisateur et effectue le rendu des données

Traitement métier des données :
Coordonne les décisions, les évaluations logiques et effectue les calculs suivant les règles métiers

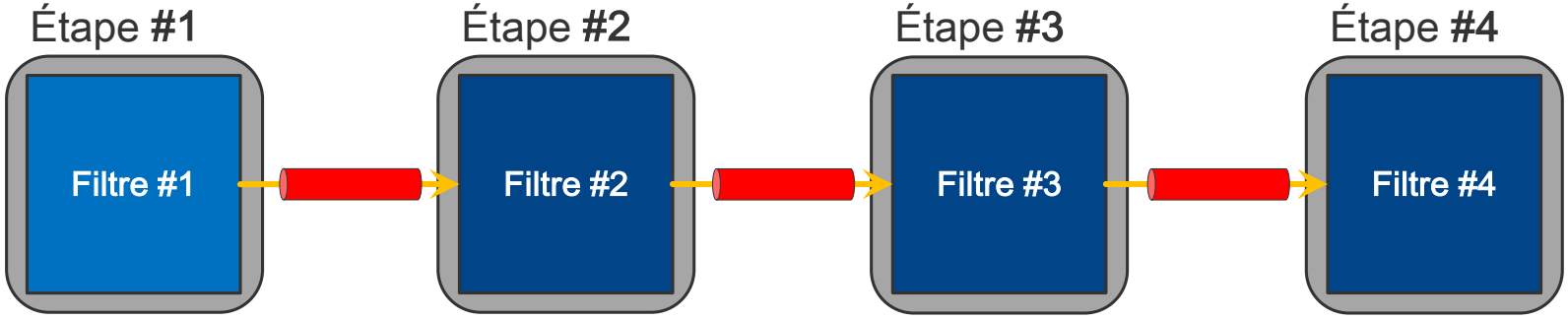
Accès aux données persistantes :
Stocke et extrait les données de la base de données

Table des matières

Styles architecturaux

2. Tubes et filtres

- **Type d'organisation** : horizontale



- **Quand l'utiliser** ?
  - Lorsque les fonctionnalités doivent traiter les données en série.

Styles architecturaux

2. Tubes et filtres

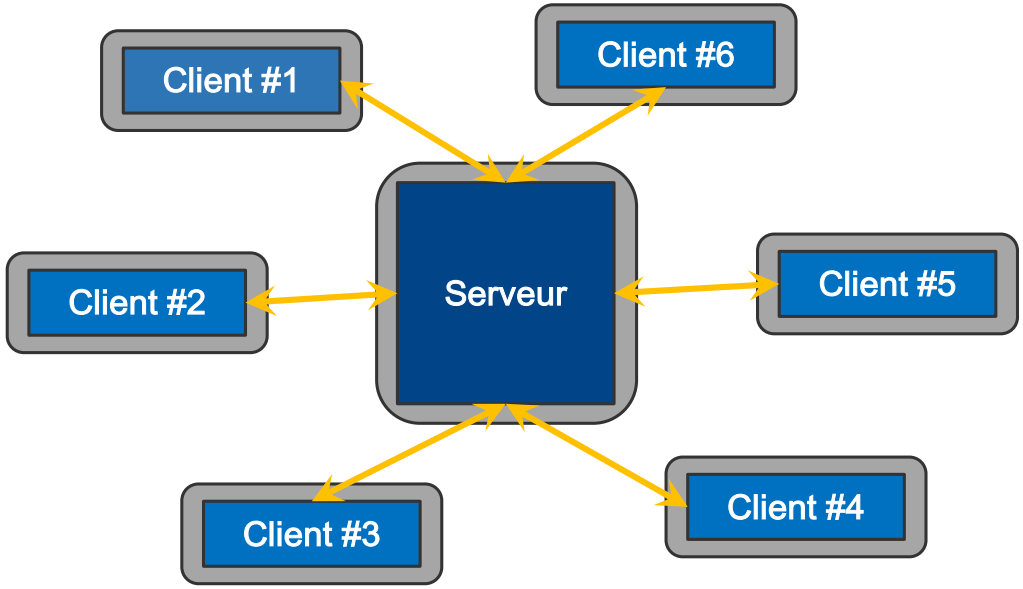
- **Principes** :
  - Chaque étape de traitement est encapsulée dans un **filtre**.
  - Les données transitent d'un filtre à l'autre par un **tube**.
  - Les tubes sont le plus souvent des **pipes** ou des **fichiers**.
  - Le comportement général du système est une succession de comportements particuliers.
- **Avantages** :
  - **Limitation à deux composants** des contraintes de communication.
  - **Remplacement facile des filtres**.
- **Exemples** :
  - Compilateur, Chaîne de traitement.

Table des matières

Styles architecturaux

3. Client/Serveur

- **Type d'organisation** : centralisée



- **Quand l'utiliser ?**
  - Lorsque le système interagit à la demande avec des services localisés.

Styles architecturaux

3. Client/Serveur

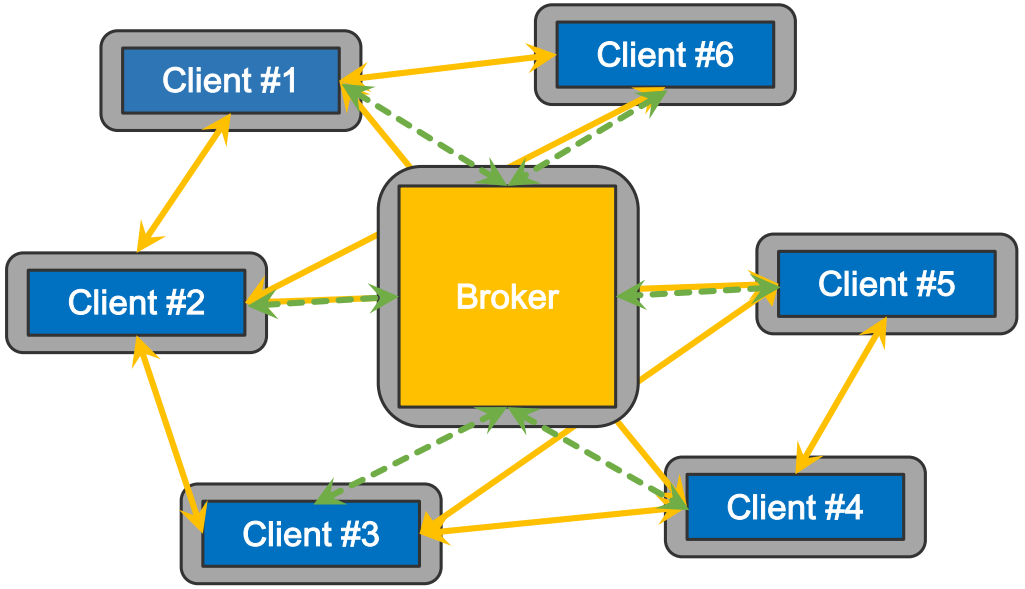
- Avantages :
  - **Indépendance des composants** pour le fonctionnement et le développement (**côté client** et **côté serveur**).
  - **Centralisation et sécurisation** des accès aux données.
- Inconvénients :
  - **Faible tolérance aux erreurs**.
  - **Forte sensibilité à la montée en charge**.
  - Importance de la disponibilité du serveur.
  - **Testabilité difficile**.
- Variantes :
  - **Clients lourds / Clients légers**.
  - **Multiplication des serveurs** : passage par un **broker** qui reçoit les requêtes des clients et effectue les redirections au bon serveur.

Table des matières

Styles architecturaux

4. Pair-à-pair (P2P)

- **Type d'organisation** : décentralisée



- **Quand l'utiliser ?**
  - Lorsque la sensibilité à la charge ou la disponibilité du serveur devient critique.

Styles architecturaux

4. Pair-à-pair (P2P)

- **Principes** :
  - **Multiplication des clients** : passage par un **broker** qui se charge du routage.
  - **Division en segment des données volumineuse** : la liste des segments est alors distribuée aux clients concernés.
- **Avantages** :
  - **Grande tolérance aux erreurs**.
  - **Rapidité de communication**.
  - **Parallélisme possible**.
  - Plus une donnée est demandée plus elle est disponible.
- **Exemples** :
  - **BitTorrent, WebRTC**.

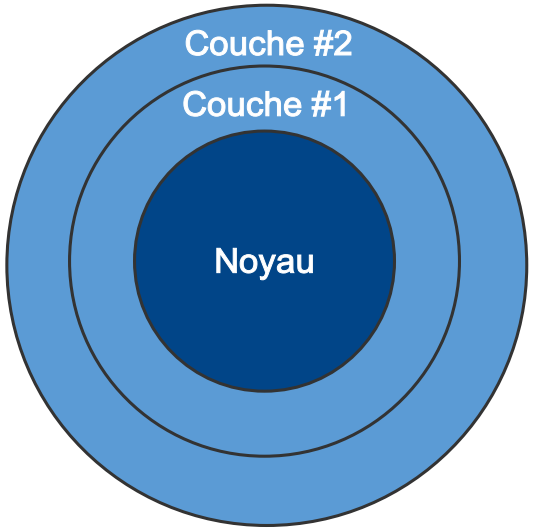
Table des matières

Styles architecturaux

5. Micro-noyau

- **Type d'organisation** : en couche

Services externes :
S'appuient sur les couches inférieures



Noyau fonctionnel :
Détient les services fondamentaux

- **Quand l'utiliser** ?
  - Lorsque le système doit être adaptable au remplacement ou au changement de l'environnement d'accueil.

Styles architecturaux

5. Micro-noyau

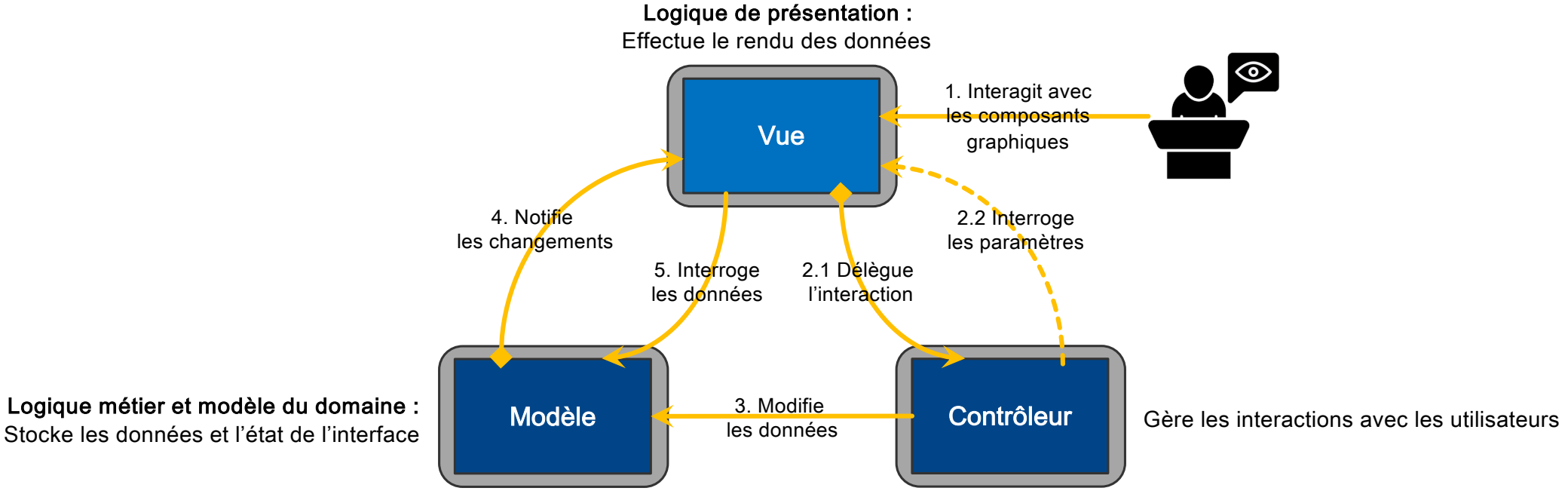
- **Principes** :
  - **Minimiser la taille du noyau**.
  - Déporter au maximum le code vers les couches externes.
- **Avantages** :
  - **Extensibilité**.
  - **Portabilité**.
- **Exemples** :
  - **Systèmes d'exploitation**.

Table des matières

Styles architecturaux

6. Modèle-Vue-Contrôleur (MVC)

- **Type d'organisation** : en boucle



- **Quand l'utiliser ?**
  - Pour les interfaces graphiques multi-fenêtrées (GUI).

Styles architecturaux

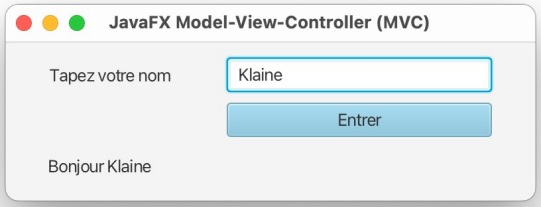
6. Modèle-Vue-Contrôleur (MVC)

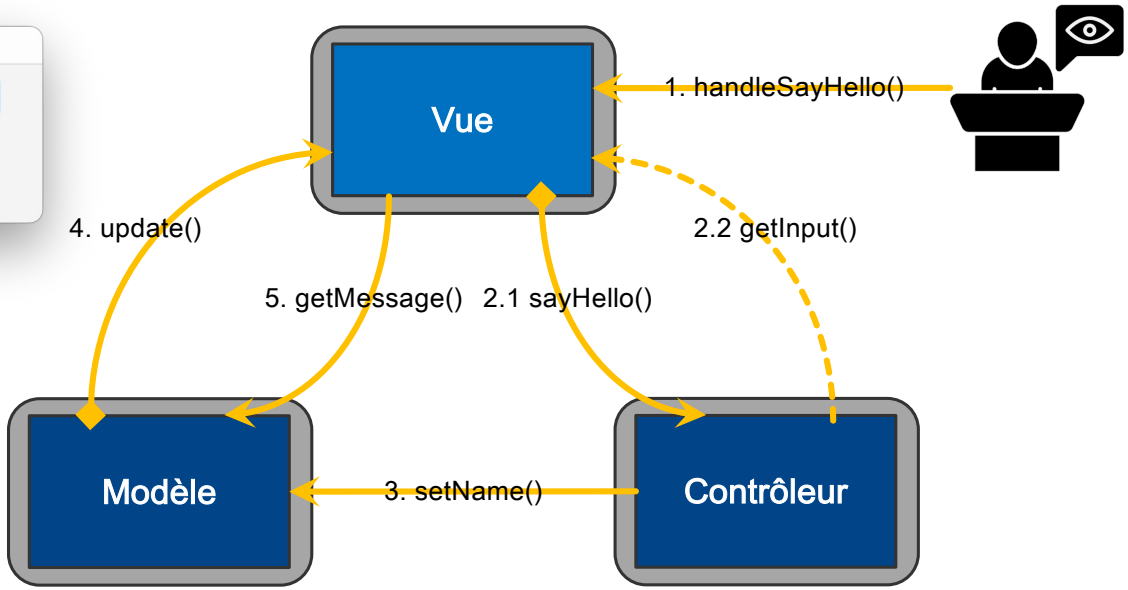
- Avantages :
  - **Découplage vue/modèle** : peuvent évoluer indépendamment.
  - **Multiplication des vues sur un même modèle**.
  - **Ajout facile de vues.**
- Inconvénients :
  - **Interaction vue/modèle :**
    - La vue a la responsabilité de l'affichage et de la récupération des données
  - **Mauvaise séparation des responsabilités :**
    - L'état courant de l'interface peut être détenu aussi bien par la vue, le modèle que le contrôleur.
  - **Testabilité difficile**.
- Conclusion :
  - **Patron peu utilisé en pratique pour les GUI** :
    - La plupart des Frameworks soi-disant MVC sont en réalité basés sur une architecture MVP.

Styles architecturaux

6. Modèle-Vue-Contrôleur (MVC)

- **Exemple** : écran de connexion





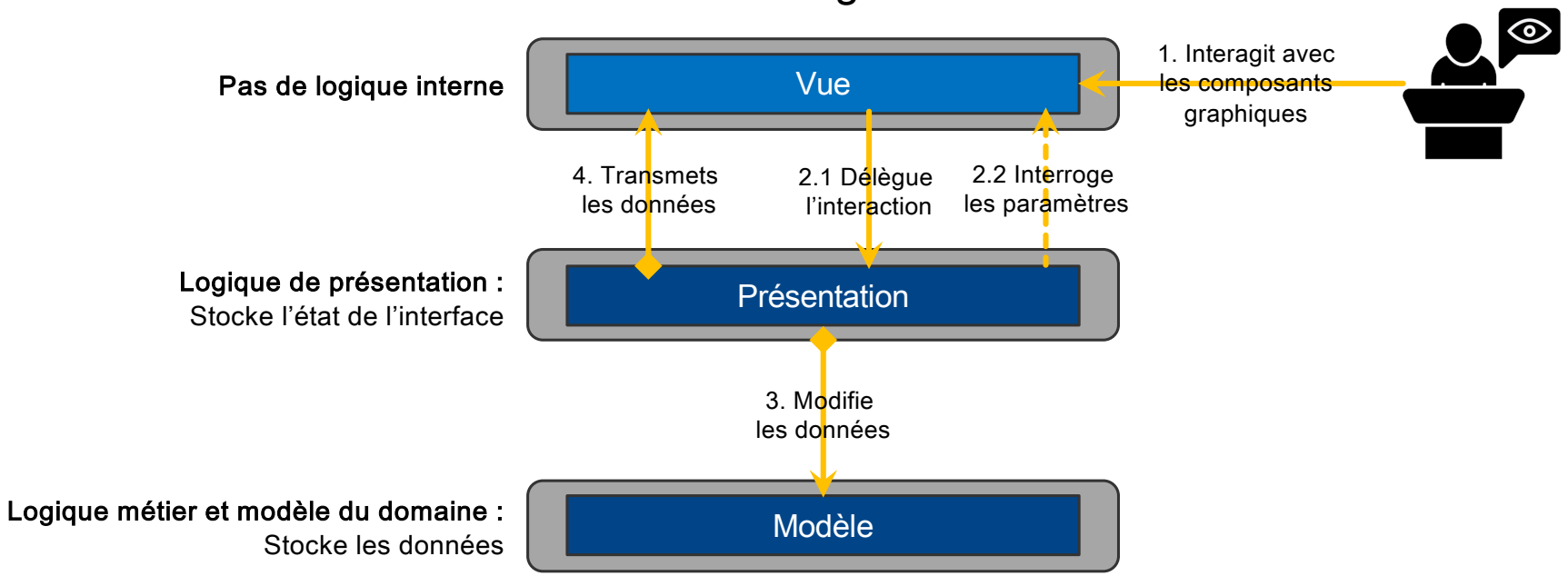
- **Exercice** : implémentation

Table des matières

Styles architecturaux

7. Modèle-Vue-Présentation (MVP)

- **Type d'organisation** : verticale
  - Dérivée de l'architecture MVC en étage



Styles architecturaux

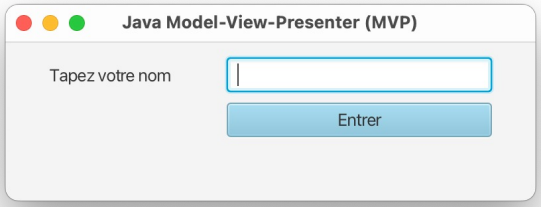
7. Modèle-Vue-Présentation (MVP)

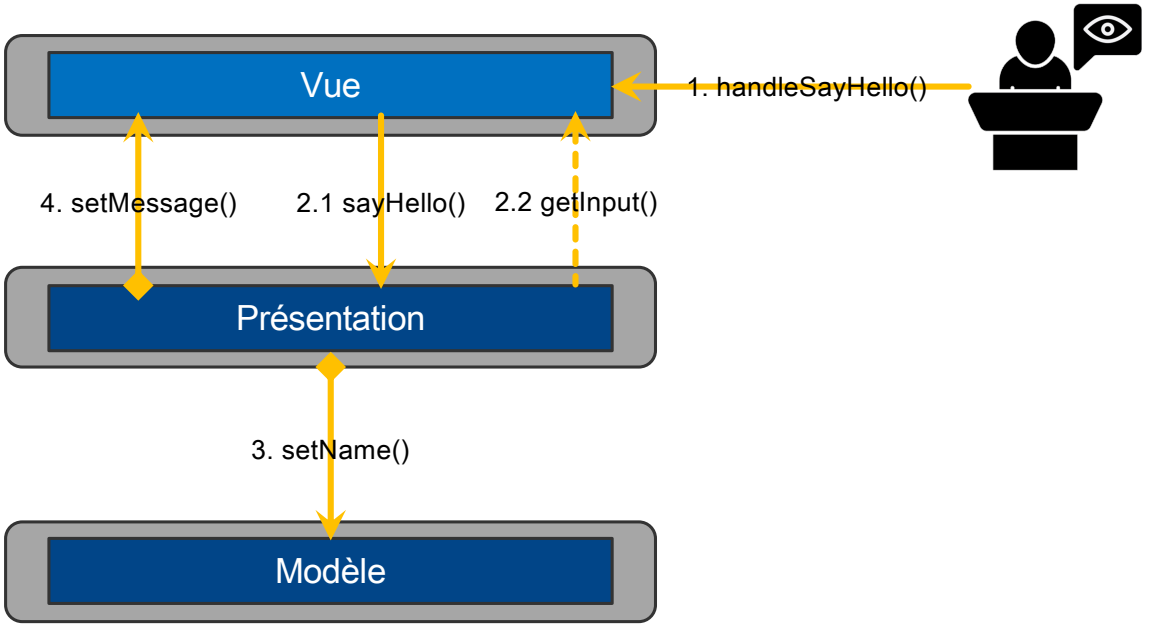
- Avantages :
  - Suppression de l'interaction vue/modèle :
    - Interaction faite par le biais de la présentation qui organise les données à afficher dans la vue.
  - Testabilité facile :
    - Sauf la vue qui ne détient pas de logique.
  - Modèle indépendant de la vue et de la présentation.
- Inconvénients :
  - Code redondant dans la présentation.

Styles architecturaux

7. Modèle-Vue-Présentation (MVP)

- **Exemple** : écran de connexion





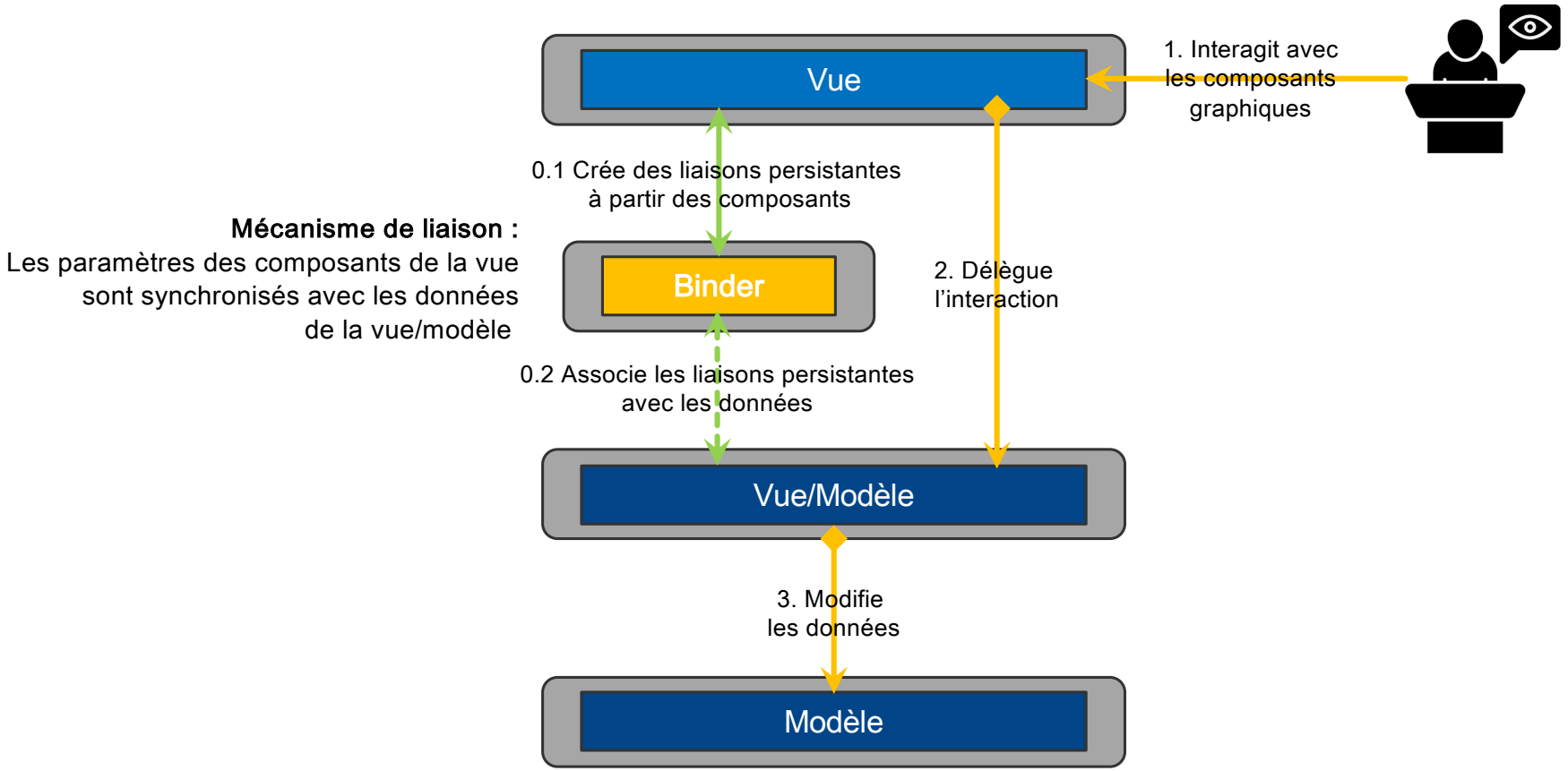
- **Exercice** : implémentation

Table des matières

Styles architecturaux

8. Modèle-Vue-Vue/Modèle (MVVM)

- **Type d'organisation** : verticale
  - Dérivée de l'architecture MVP avec un mécanisme de liaison



Styles architecturaux

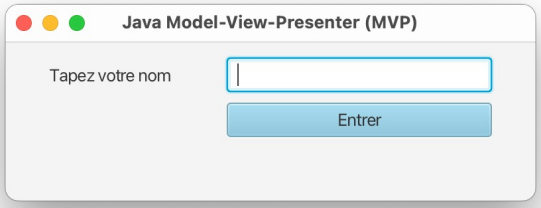
8. Modèle-Vue-Vue/Modèle (MVVM)

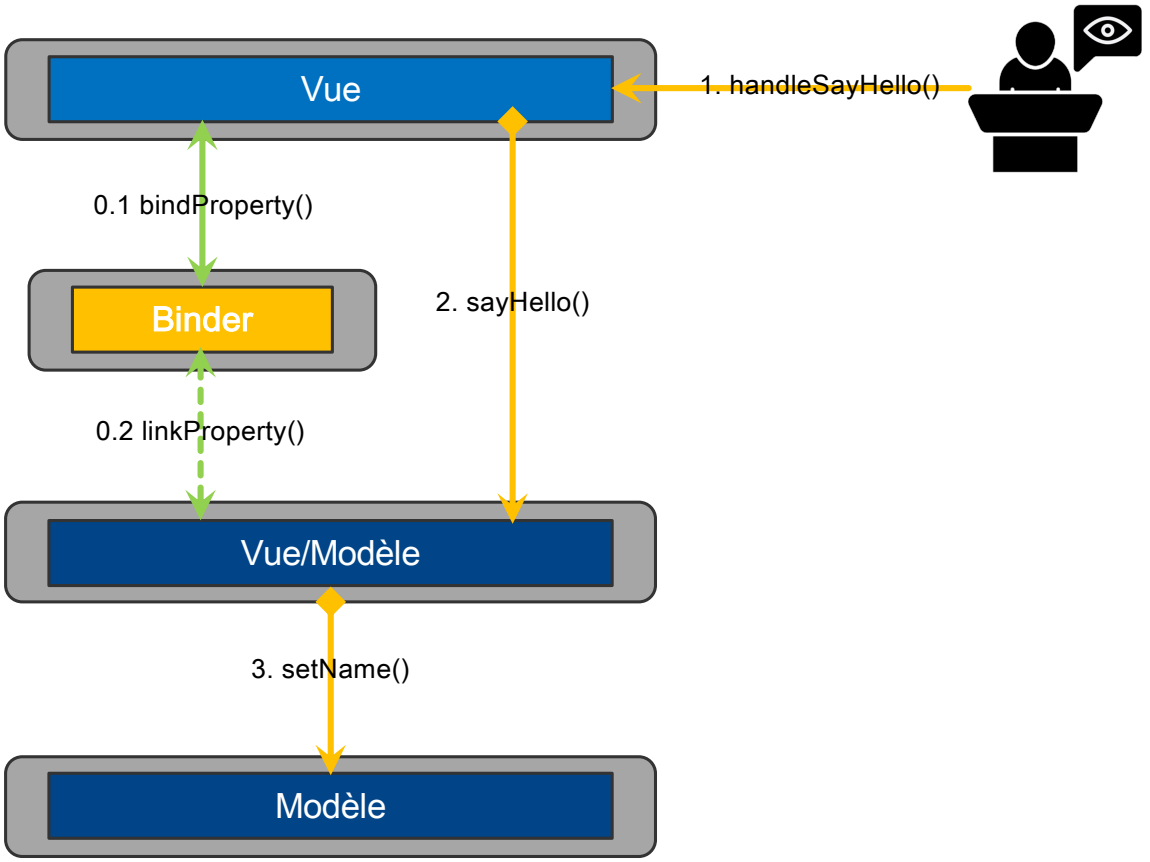
- **Avantages** :
  - **Les mêmes que pour l'architecture MVP.**
  - **Suppression du code redondant dans la vue modèle.**
- **Inconvénients** :
  - **Généralisation difficile du modèle pour les grandes applications**.
  - Le mécanisme de liaison peut être **très consommateur de ressources**.
  - À n'utiliser que **si la bibliothèque graphique ou le framework le permet**.

Styles architecturaux

8. Modèle-Vue-Vue/Modèle (MVVM)

- **Exemple** : écran de connexion





- **Exercice** : implémentation

Table des matières

Conclusion

Les bonnes pratiques de l'architecture logicielle

- L'architecture générale doit être choisie le plus tôt possible.
- Elle peut combiner plusieurs architectures.
- Elle doit permettre de différer les décisions majeures.
- Elle doit rester relativement stable même si elle doit pouvoir évoluer pour s'adapter aux particularités à venir.
- Elle doit s'appuyer sur des interfaces utilisés comme des pares-feux entre les composants.

Patrons d'Architecture

R4.01 - Architecture Logicielle

Luc Klaine

Avez-vous des questions ?

luc.klaine@univ-rennes.fr