



04

Chapitre

Patrons d'architecture

2I1AC3 : Génie logiciel et Patrons de conception

Régis Clouard, ENSICAEN - GREYC

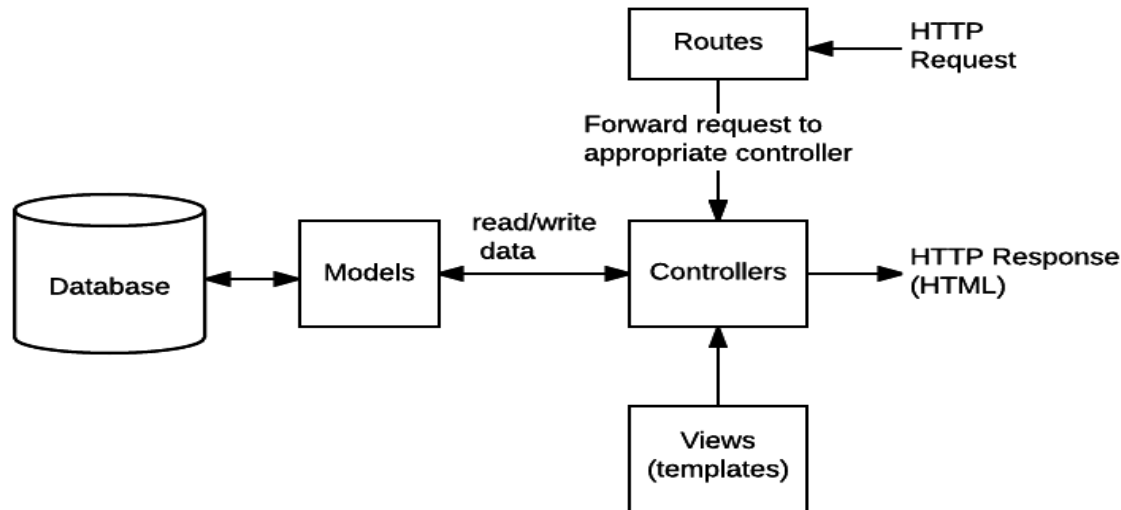
« L'informatique n'est pas plus la science des ordinateurs
que l'astronomie n'est celle des télescopes. »

Michael R. Fellows et Ian Parberry

Définition

2

- Patron d'architecture
 - Il se réfère à l'organisation structurelle de l'application
 - Il est d'un niveau d'abstraction supérieur au patron de conception
- L'architecture est la vision globale de l'organisation des composants du logiciel
 - Elle définit le *framework* du logiciel
- Exemple d'une architecture type pour Node.js



- Les enjeux sont les mêmes que pour la conception orientée objet
- Accroître :
 - Développabilité
 - Maintenabilité
 - Testabilité
 - Réutilisabilité

Préoccupations pour le choix d'une architecture

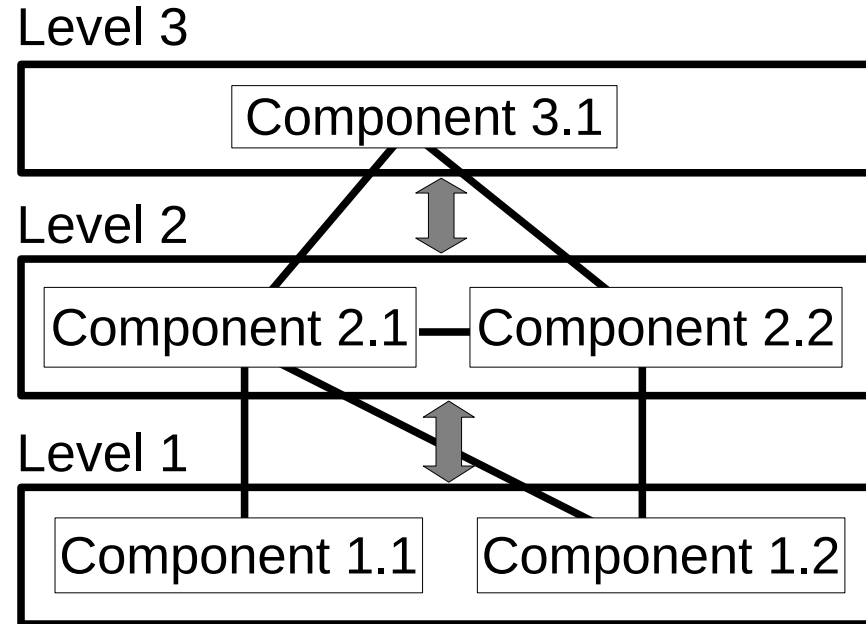
4

- L'architecte logiciel doit se poser les questions suivantes :
 - Le système est-il interactif ?
 - Nécessite-t-il de fréquents changements ?
 - Le système est-il réparti sur le réseau ?
 - Comment les fonctionnalités sont-elles décomposées entre les composants ?
 - Quelles exigences non fonctionnelles sont importantes ?
 - ▶ Limitation des performances du matériel.
 - ▶ Haute disponibilité : tolérance aux pannes.
 - ▶ Risques d'entreprise (économique, financier, santé, compétitivité, etc).
 - ▶ Sécurité.
 - ▶ ...
- Note : Dans un rapport de projet, ces questions doivent être adressées dans la description des objectifs.

1. Architecture en étages (*n-tier architecture*)

5

■ Organisation verticale



■ Quand l'utiliser ?

- Plusieurs niveaux d'abstraction dans les responsabilités.
 - ▶ Les couches supérieures : interactions avec les utilisateurs (requête).
 - ▶ Les couches basses : opérations sur les données (réponse).

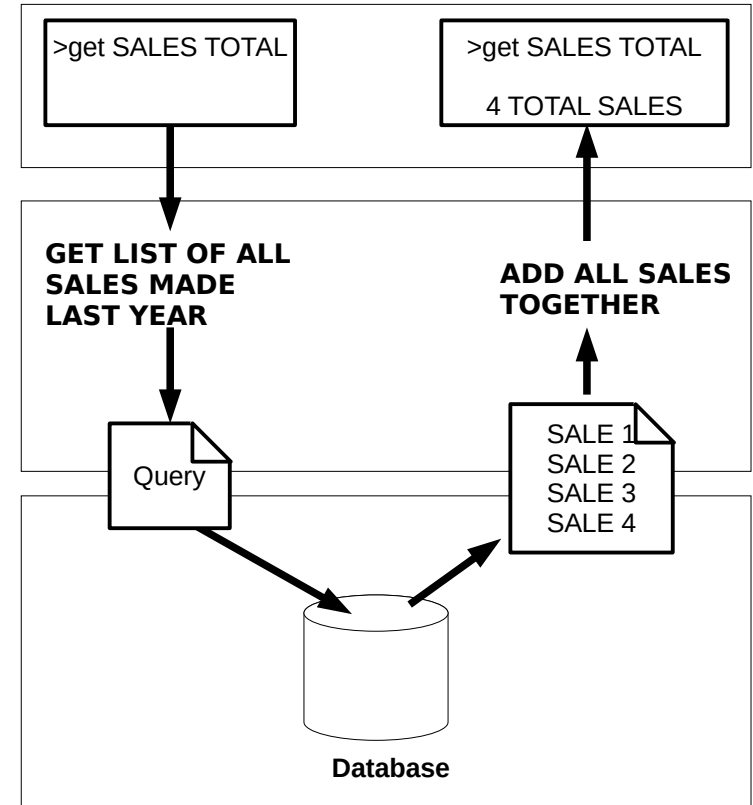
■ Exemple : *modèle OSI d'ISO : 7 étages*

- Recommandation
 - Un étage est formé par un groupe de classes qui sont **réutilisables dans les mêmes conditions**.
 - Les relations d'un étage à un autre sont décrites par des **interfaces**. Le but est de créer des pare-feux entre étages.
 - Aucun composant ne peut s'étaler sur deux étages.
 - Les échanges sont limités entre deux étages consécutives.
- Avantages
 - Étages réutilisables et interchangeables.
 - Dépendances uniquement locales entre deux étages consécutifs.
 - Les développeurs et utilisateurs de chaque niveau peuvent ignorer les autres étages.

Exemple architecture à trois étages

7

- Organisation verticale (cas particulier du n-tier)
 - Niveau **présentation** (interface utilisateur)
 - ▶ Visualise les données.
 - Niveau **application** (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 l'utiliser ?
 - Architecture client-serveur présentant une interface utilisateur.
- Exemple : *sites Web de commerce électronique*



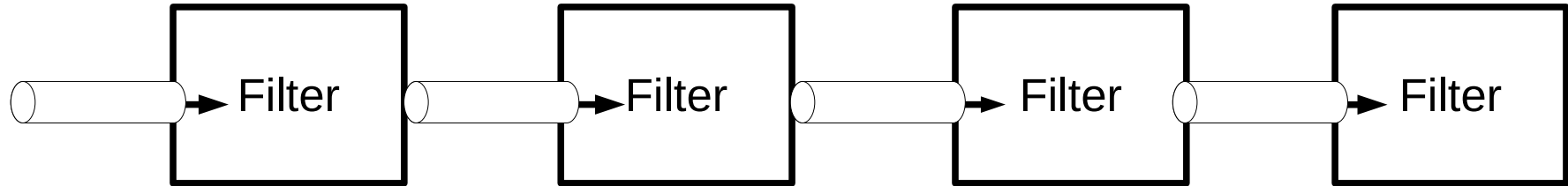
Exemple d'une architecture à 2 étages : Réflexion (Reflection)

- Sémantique bien précise :
 - Niveau **logique** : présentation des données aux utilisateurs
 - Niveau **physique** : implémentation des données.
- Quand l'utiliser ?
 - Recherche d'efficacité quand la représentation des données diffère de la présentation qui en est faite à l'utilisateur.
- Exemple : *Système de gestion de bases de données*

2. Tubes et Filtres (*Pipes & Filters*)

9

- Organisation horizontale



- Quand l'utiliser ?

- 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.
 - ▶ Chaque filtre est au même niveau d'abstraction que les autres.

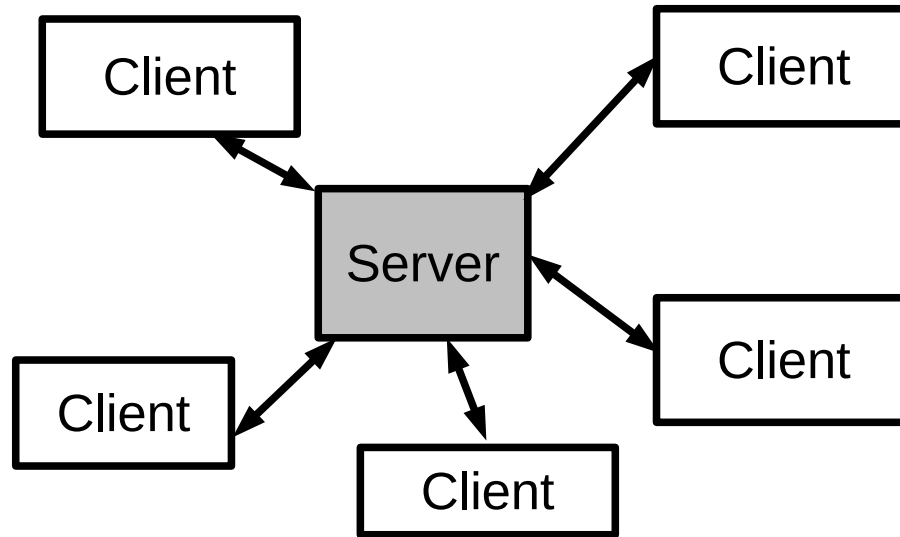
- Exemples : *Compilateur, Chaîne de traitement d'images*

- Recommandation
 - Les tubes sont classiquement des « pipes » de communication entre processus ou des fichiers.
- Caractéristiques
 - Il n'y a plus de relation hiérarchique.
 - Le comportement général du système est une succession de comportements individuels.
- Avantages
 - Les contraintes de communication sont limitées à deux composants.
 - Facilité pour remplacer un filtre.
 - Facilité pour implémenter la concurrence entre filtres (parallélisme).

3. Clients - Serveur

11

- Organisation centralisée



- Quand l'utiliser ?

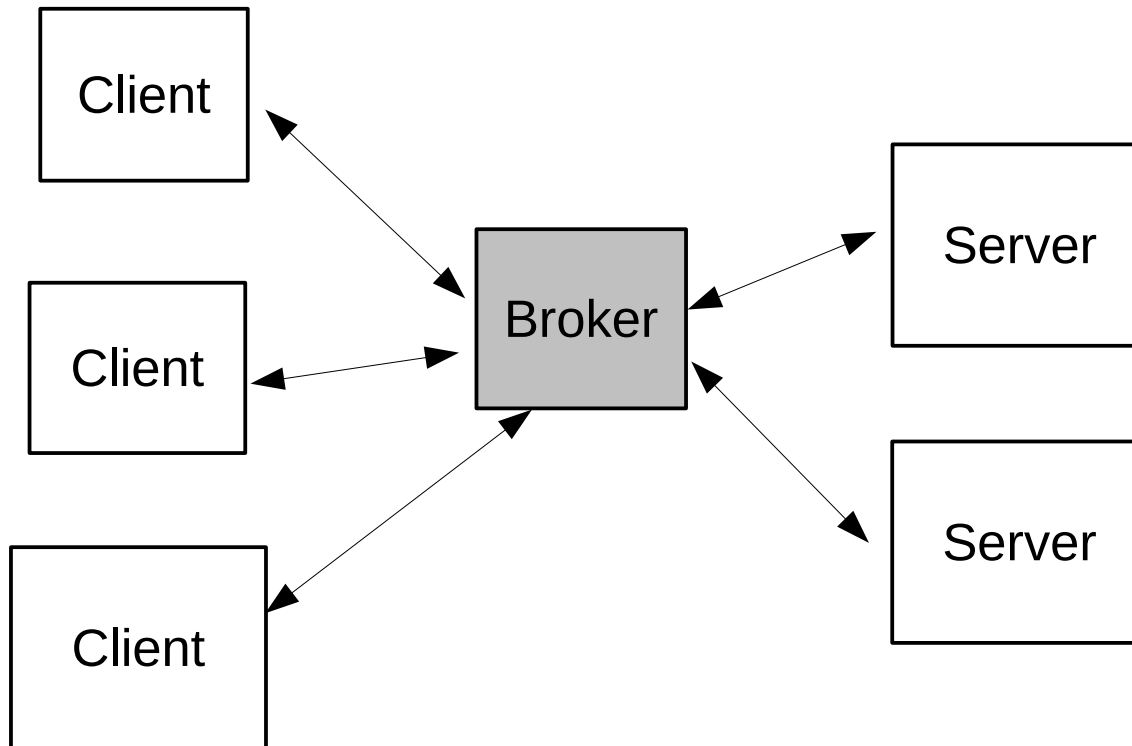
- Système distribué qui interagit par invocation de services localisés.
 - ▶ Le serveur est un fournisseur de services et de ressources. Il est typiquement composé d'une base de données.
 - ▶ Le client interagit avec l'utilisateur et formule des requêtes au serveur.

- Exemple : *Guichet Automatique Bancaire*

Version multi-serveurs : Le patron Broker

12

- Multi-serveurs : un Broker est responsable de la communication. Il fait l'intermédiaire pour localiser le serveur qui possède le service. Les clients envoient les requêtes au broker qui les redirige vers le bon serveur.

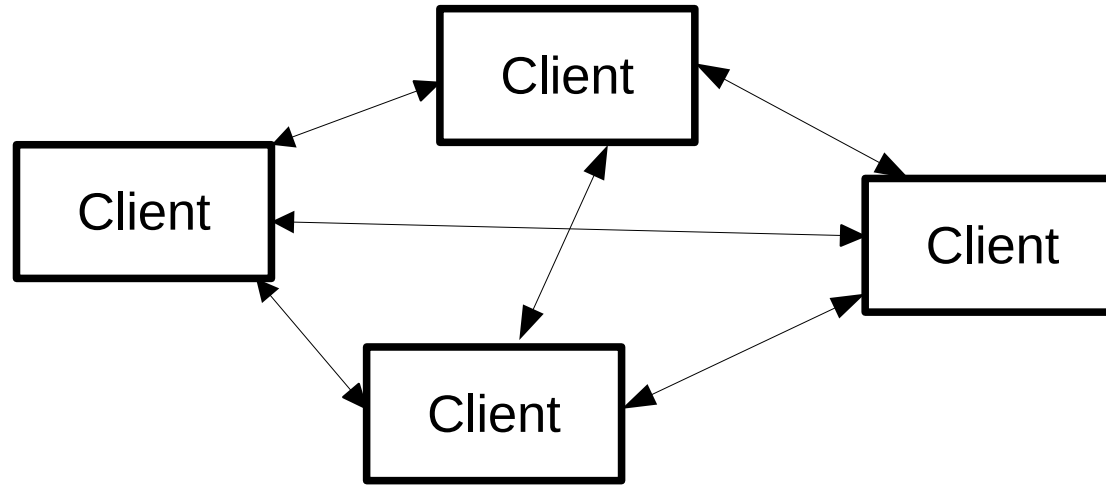


- Variations
 - Clients lourds / Clients légers. Le client le plus léger ne fait que l'affichage des résultats de requête retournés par le serveur.
 - Le serveur et les clients sont généralement distribués sur des nœuds différents et la communication utilise le réseau mais tout peut très bien être localisé sur un même nœud et la communication utilise la mémoire.
- Avantages
 - Indépendance des composants pour le fonctionnement et le développement.
 - ▶ Côté serveur / Côté client.
 - Centraliser et sécuriser les accès aux données.
- Inconvénients
 - Faible tolérance aux erreurs. Tout repose sur la disponibilité du serveur.
 - Sensible à la montée en charge.
 - Difficile à tester.

4. Pair-à-Pair (*Peer to peer*)

14

- Organisation décentralisée



- Quand l'utiliser ?

- Constat : avec une architecture client-serveur, plus une donnée est demandée moins elle est disponible sur le serveur.
- L'architecture pair-à-pair inverse la tendance : chaque client qui a récupéré la donnée devient un serveur.
- Si nécessaire, un « broker » se charge du routage.

- Exemple : *BitTorrent*, *Git*

- Recommandations
 - Si la donnée est volumineuse, elle est divisée en segments. Un nœud du réseau distribue alors une liste de ces segments.
- Avantages
 - Grande tolérance aux fautes.
 - Rapidité de communication.
 - Autorise le parallélisme.
 - Plus une donnée est demandée plus elle est disponible.

5. Micro-noyau (Microkernel)

16

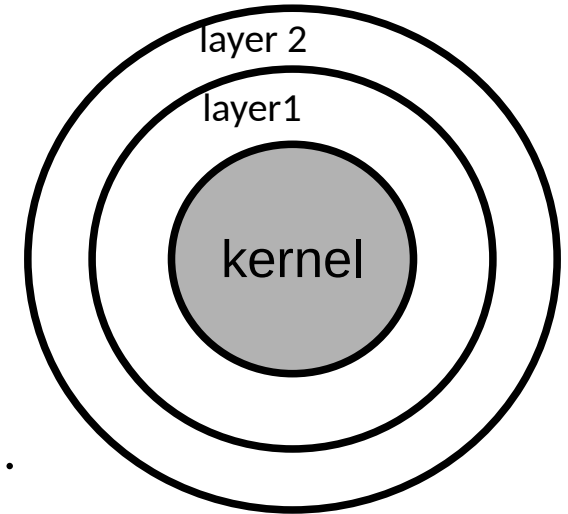
■ Organisation en couches

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

■ Quand l'utiliser ?

- Rendre les services indépendants du remplacement du noyau (p. ex. architecture matériel, OS) en plaçant la plus grande partie des services à l'extérieur de ce noyau.

■ Exemples : *Windows NT, Machine virtuelle Java (JVM)*



- Recommandation
 - Minimiser la taille du noyau et déporter la plupart du code vers les couches externes.
- Avantages
 - Extensibilité.
 - Portabilité : seul le noyau a besoin d'être changé.

6. Modèle-Vue-Contrôleur (MVC)

18

■ Organisation en boucle (version originelle)

• Modèle

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

• Vue

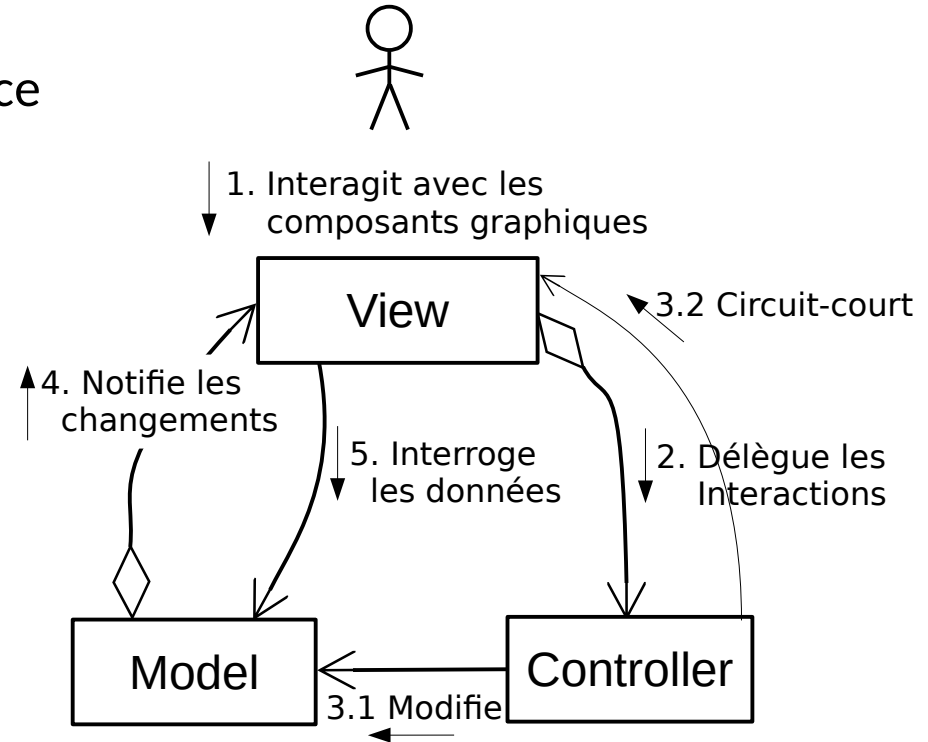
- ▶ Affiche les informations aux utilisateurs
- ▶ Détient la logique de présentation

• Contrôleur

- ▶ Gère les interactions avec les utilisateurs

■ Quand l'utiliser ?

- Une interface graphique multi-fenêtrée (GUI)



Les logiques dans une interface graphique

19

- Logique métier (cf. modèle du domaine et règles métier)
 - Code de l'application qui crée, stocke et modifie les données et qui leur donne un sens.
 - Elle est spécifique du domaine d'application.
 - Elle ne se préoccupe pas de la façon de présenter ses données.
- Logique de présentation
 - Code relatif à la façon de réagir aux interactions avec l'utilisateur et de présenter les informations.

Exemple de logiques

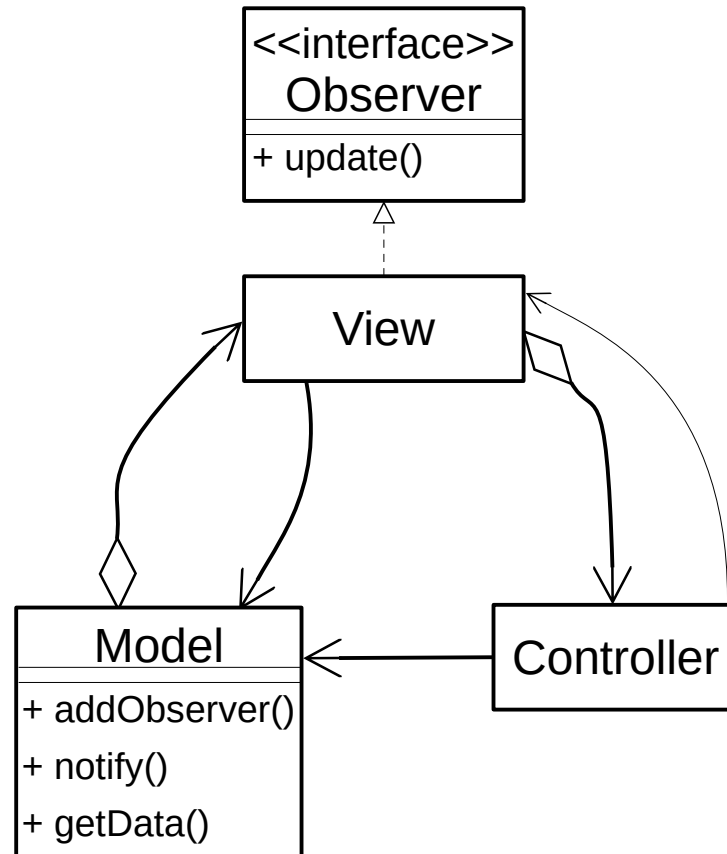
20

- Soit une application qui gère l'évolution d'une température ambiante.
 - Une règle indique que si la valeur de la température dépasse un certain seuil alors elle doit être affichée avec la couleur rouge sinon noire.
- Nous divisons cette règle en trois parties :
 - 1) Si la valeur dépasse un certain seuil, elle est trop élevée
 - ▶ **Logique métier**
 - 2) Si elle est trop élevée, elle devrait être affichée de manière spéciale
 - ▶ **Logique de présentation**
 - 3) Pour marquer une valeur spéciale, afficher en rouge sinon en noir
 - ▶ **Code du rendu visuel sans logique**

Implémentation

21

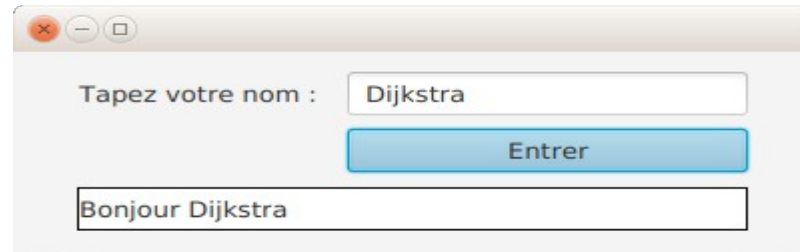
- Découpler les vues du modèle : Observateur
 - Ajouter un protocole de souscription / notification pour les vues au modèle.



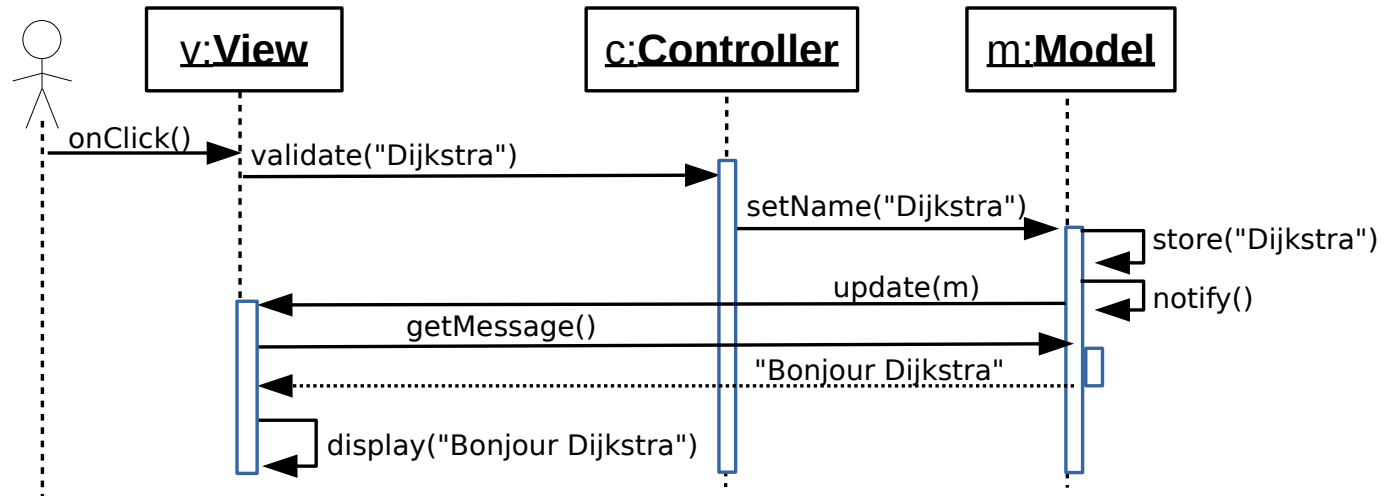
Exemple d'un écran de connexion

22

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



- Diagramme de séquence

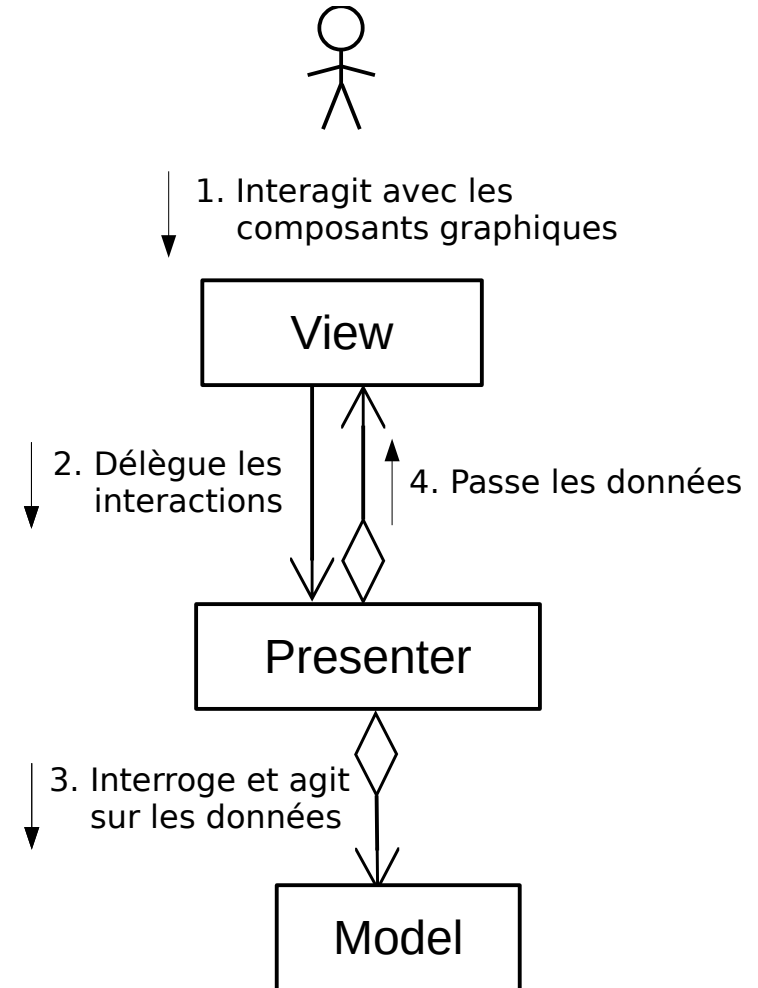


- Avantages
 - Découplage entre les vues et le modèle, qui peuvent évoluer indépendamment.
 - Plusieurs vues sur le même modèle.
 - Une vue peut être ajoutée facilement.
- Inconvénients
 - Mauvaise séparation des responsabilités
 - ▶ La vue a 2 responsabilités : affichage et récupération des données.
 - ▶ Qui détient l'état courant de l'interface : contrôleur, modèle ou vue ?
 - Difficilement testable (inter-dépendance des unités).
- Conclusion
 - Ce patron est, en réalité, peu utilisé
 - On lui préfère des patrons dérivés (MVP et MVVC)

Modèle-Vue-Présentation (MVP)

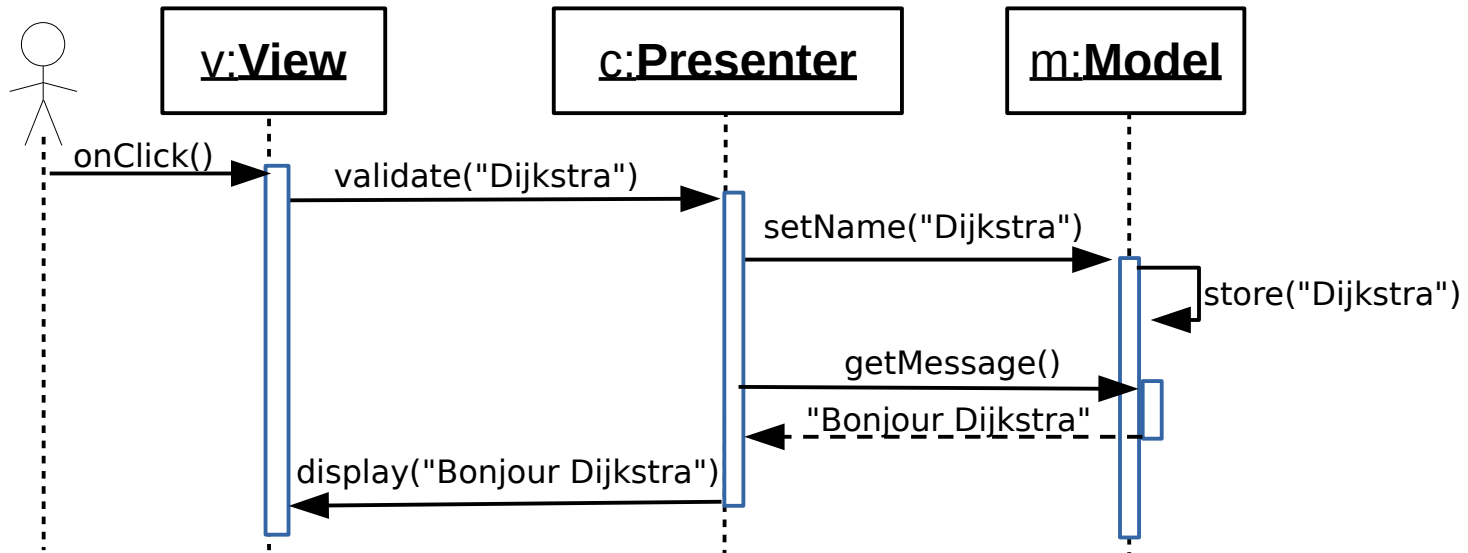
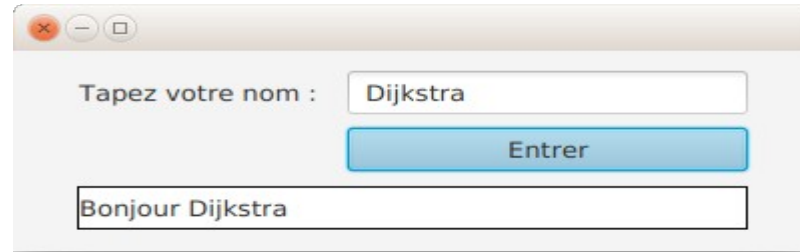
24

- Variante du modèle MVC mais s'appuyant une architecture en 3 étages (3-tier).
 - Intentions :
 - ▶ Réduire le couplage Vue – Modèle
 - ▶ Améliorer la répartition des responsabilités
 - ▶ **Modèle**
 - Stocke les données (modèle du domaine)
 - Logique métier (règles métier)
 - ▶ **Vue**
 - Composants graphiques
 - Passive : ne contient aucune logique interne
 - ▶ **Présentation**
 - État de l'interface
 - Logique de présentation



Exemple d'un écran de connexion

25



- Vue : seule partie dépendante de la bibliothèque graphique (en théorie)
 - Pas d'inclusion de classes du modèle.
 - Les échanges avec la présentation se font à partir de classes génériques.
- Présentation
 - Pas d'inclusion de classes graphiques (en théorie).
 - Inclusion de classes du modèle.
- Modèle : indépendante des deux autres parties.
 - Pas d'inclusion de classes des autres parties.

- Avantages
 - Élimine l'interaction entre la vue et le modèle.
 - L'interaction est faite 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.
 - Le modèle est indépendant des autres composants.
- Inconvénients
 - Beaucoup de code redondant (*boilerplate code*) dans la présentation pour mettre à jour les données dans la vue, du genre :
 - ▶ `displayText1(text), displayText2(text)`
 - ▶ `setButton1Enabled(true), setButton2Enabled(true)`

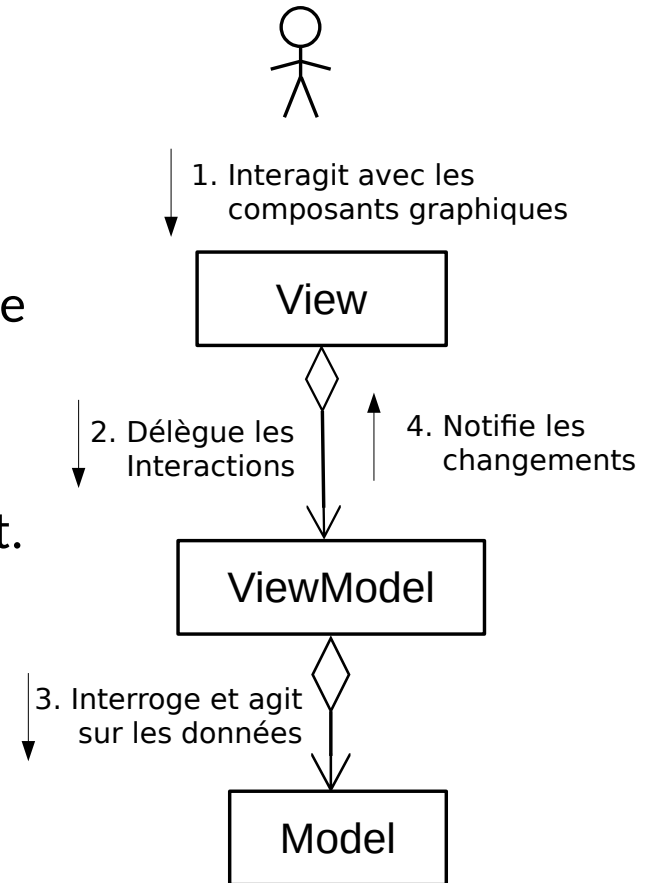
- Aujourd'hui, le nom MVC ne désigne plus une architecture en boucle, mais une architecture 3-tier semblable à MVP.
 - L'architecture en boucle est, en effet, inefficace.
- La différence réside dans la relation : qui crée qui
 - MVP : c'est la vue qui crée le présentateur (UI)
 - MVC2 : c'est le contrôleur qui crée la vue (Node.js, Symfony)

Modèle-Vue-ViewModel (MVVM)

29

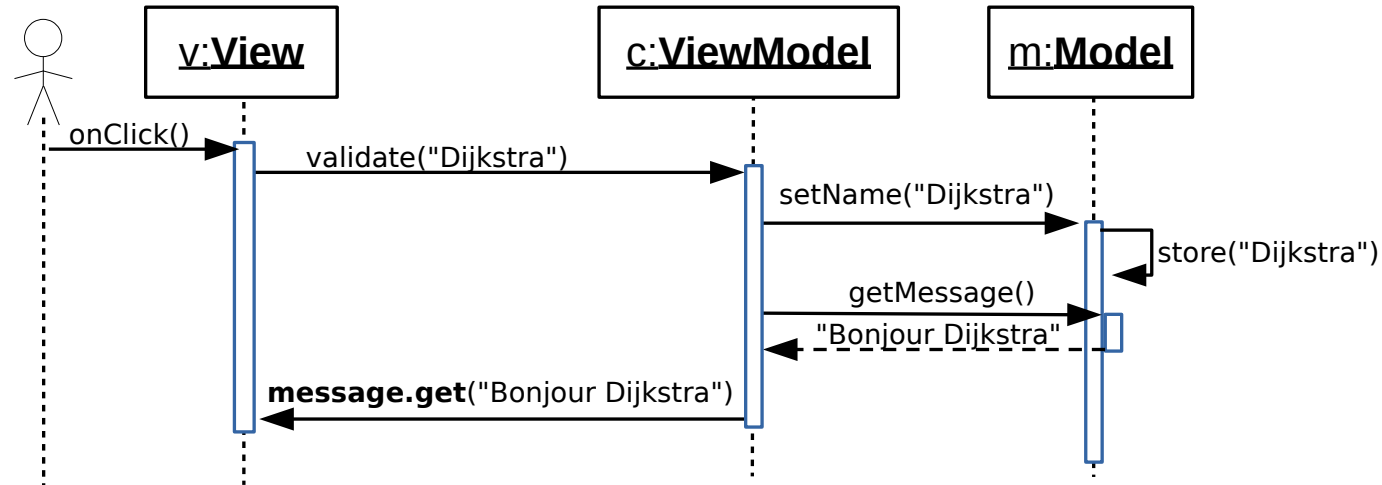
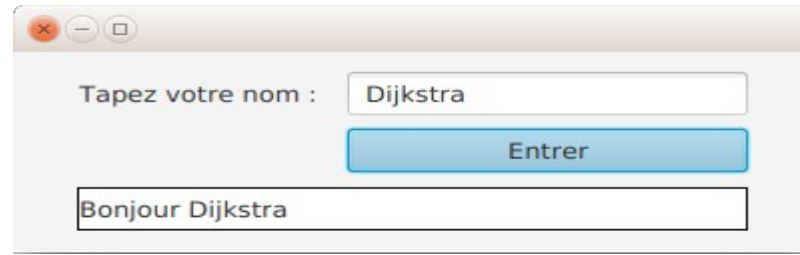
- Variante du modèle MVP
 - Intention : Supprimer le code redondant dans la présentation.
- Unique différence avec MVP
 - La mise à jour de la vue est faite par un mécanisme de liaison (*binder*) entre les composants de la vue et les attributs de vue modèle.
 - Quand un attribut de la présentation est modifié, le composant de la vue est mis à jour automatiquement.
 - Le binder est réalisé avec le patron Observateur de MVC qui est réintroduit mais au niveau de chaque attribut.
- Exemple de « binding » en JavaFX :

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



Exemple d'un écran de connexion

30



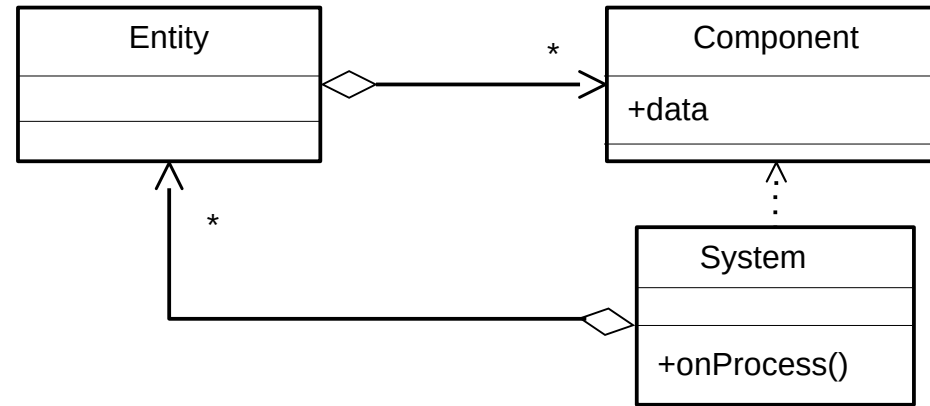
La méthode `message.get()` est appelée par le mécanisme de liaison (binder). Le label de Vue qui affiche le message est lié à l'attribut `message` de `VueModèle`.

- Avantages
 - Exactement les mêmes que pour le patron MVP.
 - Élimination du surcoût de code redondant (*boilerplate code*).
- Inconvénient
 - Généraliser le modèle de vue pour de grandes applications est difficile. La liaison de données peut aboutir à une consommation considérable de ressources (source : John Gossan créateur du patron MVVM).
 - À n'utiliser que si la bibliothèque graphique implémente nativement ces mécanismes de *binder*.
- Modèle de base pour .NET (avec XAML)

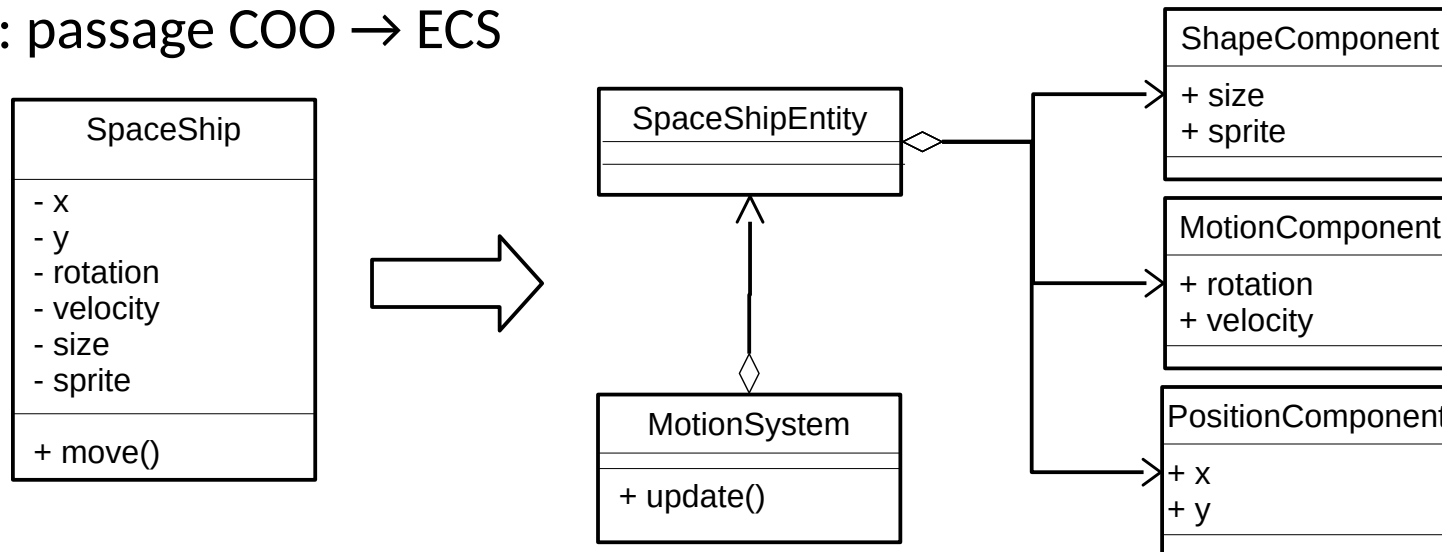
5. Entité - Composant - Système

32

- Nouveau paradigme : programmation orientée données



- Exemple : passage COO → ECS



- COO → ESCS : passer d'un tableau de structures à une structure de tableaux.
 - Entité : la référence à un objet du jeu (eg. combattant, mur). Les implémentations utilisent généralement un entier simple pour cela.
 - Composant : les données brutes pour un aspect de l'objet, et comment il interagit avec le monde (eg. la position). Les implémentations utilisent généralement des structures, des classes ou des tableaux associatifs.
 - Système : les algorithmes portant sur les composants. Chaque système fonctionne en continu (comme si chaque système avait son propre thread privé) et effectue des actions globales sur chaque entité qui possède un composant du même aspect que ce système.
- Organisation
 - Globalement un tableau où les lignes sont les identificateurs, les colonnes les systèmes et l'intersection est un composant.
- Quand l'utiliser ?
 - Vitesse d'accès aux données.
 - Limiter les associations entre les objets.
- Exemple : *moteur de jeux vidéos*

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