Quatrième partie IV

# IHM et architectures logicielles

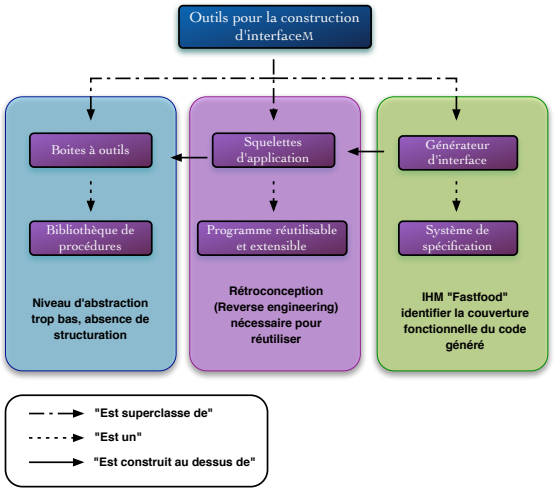
## Plan du cours



## Introduction

- Constat
  - Conception d'IHM : difficile, donc nécessairement itérative
  - Itérativité implique modifiabilité du logiciel
- Savoir-faire artisanal
  - acceptable pour maquettes, systèmes prospectifs
- Complexité et taille croissantes des IHM
- Outils de développement des IHM
  - utiles mais imparfaits

## Introduction



## Introduction

- Conséquence : besoin de cadre de pensée, i.e., de modèles d'architecture
- Finalité d'une architecture
  - Communication (précision et non ambiguïté de la description)
  - Rétro-conception d'un système existant
  - Evaluation (selon des critères qualité)

## Introduction

1. Architecture logicielle : Fondements
2. Modèles de référence : Seeheim, Arch
3. Modèles de référence à agents : MVC, PAC
4. Modèle de référence hybride : PAC-Amodeus

## Architecture logicielle fondements

- Définition de la notion d'architecture
  - Absence de définition consensuelle
- Deux définitions se dégagent
  - Définition du comité IEEE 1471 (2000)
  - Définition de Bass et al.(1998)

## Architecture logicielle fondements

- Définition
  - The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution
- Autrement dit
  - Une architecture est le résultat d'un processus contraint par l'environnement
  - L'environnement : participants (culture en qualité logicielle, outils, requis commercial, etc.)
  - Fondamental dénote les aspects du système qui sont importants pour un participant donné, impliqué dans une étape donnée du processus de développement
- Distinction entre architecture et description d'architecture
  - Une architecture est un concept : elle existe, bien que non observable
  - Une description d'architecture : représentation de ce concept pour une finalité donnée. C'est une entité concrète.

## Architecture logicielle fondements

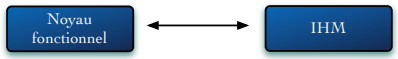
- Définition
  - A software architecture is a set of structures which comprise software components, the externally visible properties of these components and the relationships among them.
- Autrement dit
  - Plusieurs points de vue sur une architecture (cf. Architecture civile)
    - Un point de vue : une structure, sa représentaton pour une finalité donnée
  - Propriétés d'un composant : description du comportement attendu/hypothèses sur le comportement attendu
    - Services fournis ou requis, Performance, Protocole de communication, contrats
  - Propriétés observables de l'extérieur : un composant est
    - une unité d'abstraction
    - un service, un module, une bibliothèque, un processus, une procédure, un objet, un agent, etc., sont des composants
  - Relations → connexion → connecteurs (appel procédural, RMI, socket, etc.)

## Modèle de référence

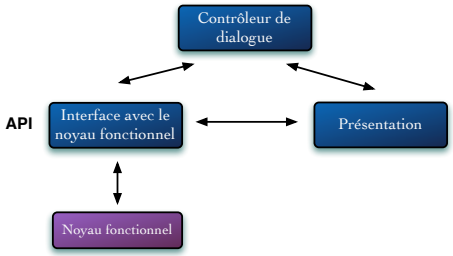
- Le modèle de Seeheim est un patron d'architecture logicielle introduit en 1983 pour structurer l'interface homme-machine dans un logiciel interacti
- Les composants du patron
  - La présentation est la couche qui gère les entrées et les sorties
  - La gestion du dialogue gère le séquencement des entrées et des sorties, par exemple l'enchaînement des écrans dans une interface graphique
  - Le modèle d'interfaçage de l'application est la couche qui sert à relier les fonctions et données du noyau fonctionnel aux données et actions de l'IHM
- Le modèle de Seeheim est un patron de conception :
  - abstrait : il ne précise pas comment réaliser les différentes parties et leurs interconnexions en utilisant les constructions disponibles dans les langages de programmation.
  - de haut niveau : il s'applique à l'analyse des logiciels à un grain macroscopique, au niveau du module, de la bibliothèque ou du processus.

## Modèle de référence

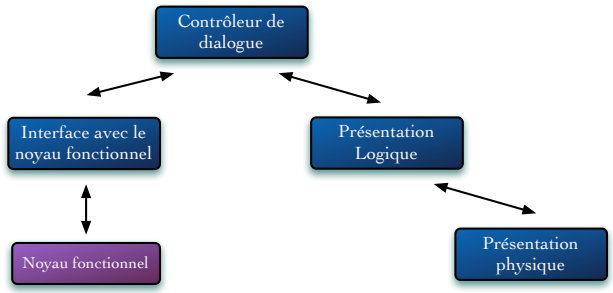
- Fondement




- Seeheim modèle séminal




## Modèle de référence



## Modèle de référence à agents

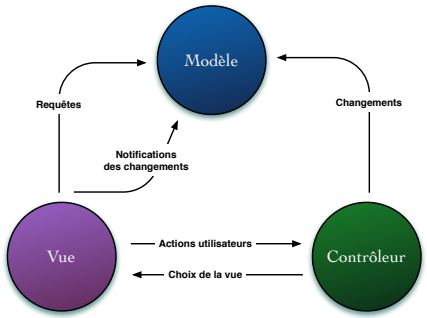
- Un système interactif = une collection d'unités de calcul "autonomes" et spécialisées (agents)
- Un agent
  - a un état
  - a une expertise
  - est capable d'émettre et de réagir à des événements
- Un agent en contact direct avec l'utilisateur = un interacteur
- Agents réactifs et agents cognitifs (IA)

## Modèle de référence à agents

- Modularité et parallélisme
  - conception itérative (modifiabilité)
  - dialogue à plusieurs fils
- Correspondance avec l'approche à objets et à composants
  - catégorie d'agents (réactifs) → classe
  - événement → méthode
  - encapsulation : l'agent (l'objet) est seul à modifier directement son état
  - mécanisme de sous-classe → modifiabilité

## Modèle de référence à agents

- Principe
  - Le patron MVC est issu de travaux de Trygve Reenskaug en 1978-79
  - Son but est de proposer une solution générale aux problèmes d'utilisateurs manipulant des données volumineuses et complexes
- Le modèle est composé :
  1. du modèle
  2. de la vue
  3. du contrôleur




## Modèle de référence à agents

- Le modèle représente le cœur de l'application
  - traitements des données, interactions avec la base de données, etc.
- Le modèle décrit les données manipulées par l'application
- Le modèle regroupe la gestion de ces données et est responsable de leur intégrité
- Le modèle comporte des méthodes standards pour mettre à jour ces données
  - insertion, suppression, changement de valeur
- Les résultats renvoyés par le modèle ne s'occupent pas de la présentation
- Le modèle ne contient aucun lien direct vers le contrôleur ou la vue
- La communication avec la vue s'effectue au travers du patron Observateur
- Le modèle peut autoriser plusieurs vues partielles des données

## Modèle de référence à agents

- Définition
  - Ce avec quoi l'utilisateur interagit se nomme précisément la vue
- Le rôle de la vue est de
  - présenter les résultats renvoyés par le modèle
  - recevoir toute action de l'utilisateur
    - hover, clic de souris, sélection d'un bouton radio, cochage d'une case, entrée de texte, de mouvements, de voix, etc.
- Les événements sont envoyés au contrôleur
- La vue n'effectue pas de traitement , elle se contente d'afficher les résultats des traitements effectués par le modèle et d'interagir avec l'utilisateur
- Plusieurs vues peuvent afficher des informations partielles ou non d'un même modèle

## Modèle de référence à agents

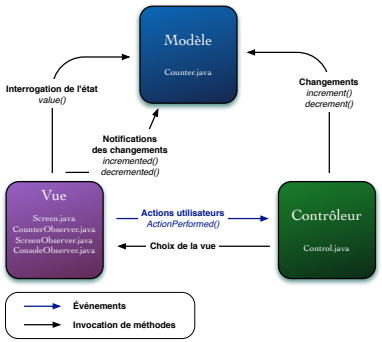
- Le contrôleur prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser
- Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer
- Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle , et ce dernier notifie la vue que les données ont changé pour qu'elle se mette à jour
- Le contrôleur n'effectue aucun traitement, ne modifie aucune donnée
  - Il analyse la requête du client et se contente d'appeler le modèle adéquat et de renvoyer la vue correspondant à la demande

## Modèle de référence à agents

- Avantages
  - Adaptée aux applications graphiques (non web)
  - Séparation des tâches :
    - Diminution de la complexité lors de la conception
    - Répartition suivant les développeurs
    - Maintenance et modifications facilitées
- Inconvénients
  - Moins bien adaptée aux applications web
  - Séparation des tâches :
    - Augmentation de la complexité lors de l'implémentation
    - Éventuel cloisement des développeurs

## Modèle de référence à agents

- Considérons un programme Java fondé sur l'architecture MVC avec deux vues du modéle d'un compteur
  - une vue "console"
  - une seconde ou' le compteur s'affiche dans une fenêtre



## Modèle de référence à agents

**La classe `Counter`**

```
public class CounterModel {

    private int value = 0;
    private CounterObserver observer ;

    public void setObserver(CounterObserver co) {
        observer = co ;
        co.value() ;
    }

    public void increment() {
        value++ ;
        observer.incremented() ;
    }

    public void decrement() {
        value-- ;
        observer.decremented() ;
    }

    public int value() {
        return value ;
    }
}
```

## Modèle de référence à agents

**La classe `Control` – Initialisation**

```
public class Control extends JPanel implements ActionListener {

    private JButton b1, b2, b3, b4 ;
    private CounterObserver screen, console ;
    private Counter counter ;
    private Worker worker ;

    public void initialize() {
        b1 = new JButton("Incremente") ;
        b1.setActionCommand("increment") ;
        b1.addActionListener(this) ;
        b1.setToolTipText("Incremente le compteur");

        // Ajouter les composants au conteneur courant
        add(b1) ;
        (...)
    }
}
```

## Modèle de référence à agents

**La classe `Control` – Gestion des événements**

```
public class Control extends JPanel implements ActionListener {

    private JButton b1, b2, b3, b4 ;
    private CounterObserver screen, console ;
    private Counter counter ;
    private Worker worker ;
    (...)

    public void actionPerformed(ActionEvent e) {
       if ("increment".equals(e.getActionCommand())) {
         counter.increment() ;
       } else if ("decrement".equals(e.getActionCommand())) {
         counter.decrement();
       } else if ("screen".equals(e.getActionCommand())) {
         counter.setObserver(screen) ;
       } else if ("console".equlas(e.getActionCommand())) {
         counter.setObserver(console) ;
       }
    }
    (...)
}
```

## Modèle de référence à agents

**La classe `CounterObserver`**

```
interface CounterObserver {
  public void value() ;
  public void incremented() ;
  public void decremented() ;
}
```

## Modèle de référence à agents

La classe `ConsoleObserver`

```
class ConsoleObserver implements CounterObserver {

  private counter ;

  ConsoleObserver(Counter c) {
    this.counter = c ;
  }

  public void value() {
    System.out.println("Counter Value = " + counter.value());
  }

  public void incremented() {
    System.out.println("Counter Incremented, New Value = " + counter.value());
  }

  public void decremented() ;
    System.out.println("Counter Decremented, New Value = " + counter.value());
  }
}
```

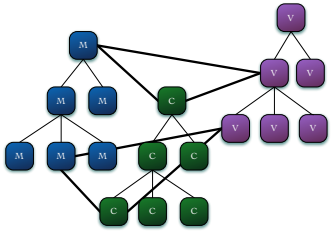
## Modèle de référence à agents

La classe `ConsoleObserver`

```
class ScreenObserver implements CounterObserver {

  private Screen screen ;
  pricate Counter counter ;

  ScreenObserver(Counter c) {
    this.counter = c ;
  }

  public void value() {
    screen.setText(Integer.toString(counter.value());
  }

  public void incremented() {
    screen.setText(Integer.toString(counter.value());
  }

  public void decremented() ;
    screen.setText(Integer.toString(counter.value());
  }
}
```

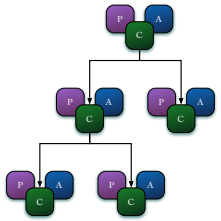
## Modèle de référence à agents

- L'architecture MVC vise à séparer le modèle de la vie du comportement
- L'implantation du modèle MVC utilise le patron de conception "observer"
  - Ce patron s'appuie sur la programmation événementielle
  - En java, il faut utiliser les *listeners*
  - Ce patron est la base de la modularité et permet de changer de vue
- Pour des applications plus complexe, il est nécessaire de construire une hiérarchie de modèles, de vues et de controleurs




## Modèle de référence à agents

- Principe
  - Présentation : le V + C de MVC
  - Abstraction : le M de MVC
  - Communication : échanges avec les autres agents
- Mise en œuvre
  - Aucune recommandation particulière
  - Dépend de la plateforme d'acceuil
  - 1 agent par concept = 1 P + 1 A + 1 C

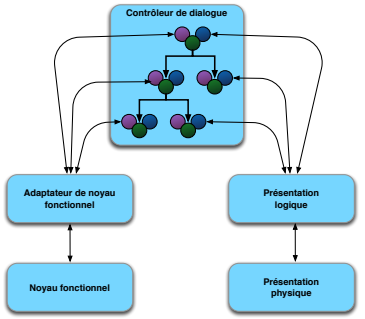


## Modèle de référence à agents

- Règle 1 : Un agent par espace de travail (fenêtre, panel)
- Règle 2 : Un agent PAC par objet complexe du domain, e.g., un interacteur inexistant dans la boite à outils)
- Règle 3 : Si “analyse syntaxique distribuée” alors agent Ciment syntaxique qui fusionne les actions distribuées en un niveau d’abstraction supérieur
- Règle 4 : Agent Vue multiple pour maintenir la cohérence entre plusieurs vues
- Règle 5 : Optimisation de la hiérarchie d’agents PAC
  - Remonter les fonctions du fils chez le père mais bien penser à l’évolution du logiciel avant de faire cela
- Règle 6 : Si les agents fils partagent la même abstraction, faire remonter ce A dans la partie A du parent - ou bien faire une pâquerette
- Règle 7 : Utiliser la délégation sémantique (méta-modèle Slinky) grâce aux A des agents qui peuvent contenir des informations et calcul relevant du noyau fonctionnel

## Modèle de référence hybride

- Motivations
  - Conserver la décomposition fonctionnelle de Arch (bon cadre de raisonnement)
  - Conserver la modularité et le parallélisme des modèles à agents
  - Permettre la délégation NF sémantique dans l’IHM via les facettes A des agents (performance)




## Modèle de référence hybride

- Liaison avec les objets implémentant les concepts du domaine
  - un agent lié participe à la chaîne de transformations du monde Noyau Fonctionnel vers le monde IHM concrete
  - un agent non lié est indépendant du domaine (améliorations conceptuelles)
- Les facettes d’un agent
  - un agent sans A (ou A minimaliste)
    - est une extension de la boîte à outils
    - est en liaison directe avec un objet-conept du domaine qui lui sert de compétence (A minimaliste)
  - un agent sans P
    - est une unité de calcul
    - maintient des relations entre agents (par exemple agent ciment syntaxique)

## Conclusion

- Il faut imérativement séparer le noyau fonctionnel de l’IHM
- Le modèle MVC est le plus répondu
  - Il n’est pas très adapté au application web
- les modèles à agent sont les plus modulables
  - Ils sont aussi les plus complexes à implanter
  - Le modèle PAC-Amodeus est un bon compromis