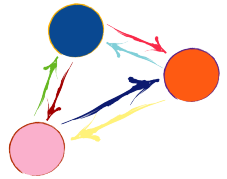
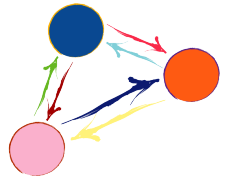


Interaction Homme-Machine

Repères fondamentaux en architecture logicielle

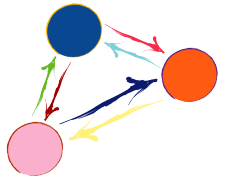
Gaëlle Calvary
Professeur en Informatique

Institut polytechnique de Grenoble
Laboratoire d'Informatique de Grenoble

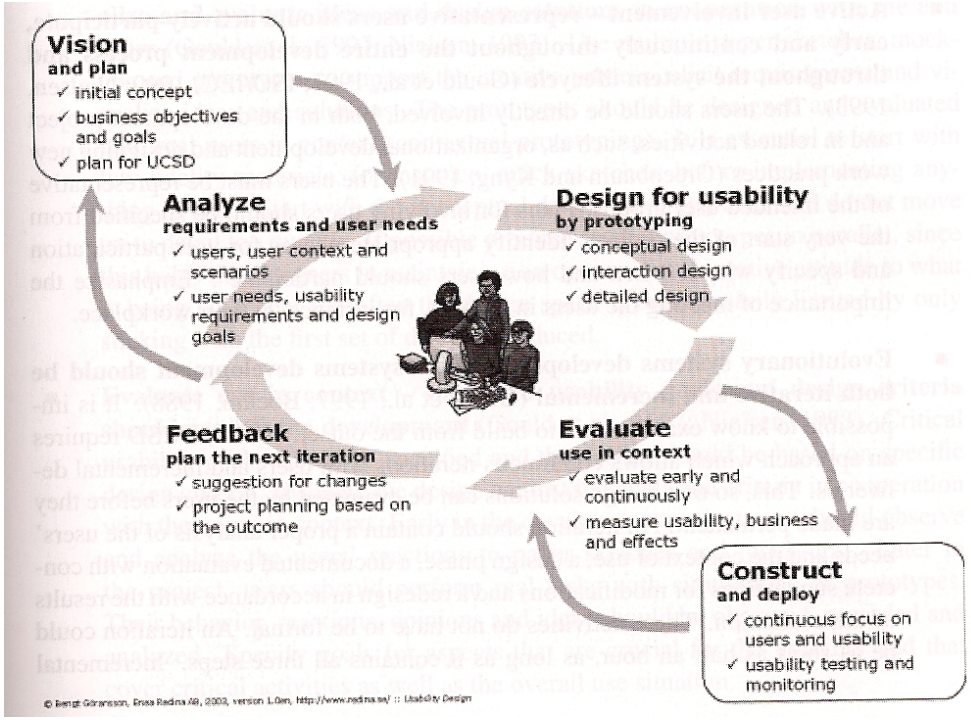


Principes généraux

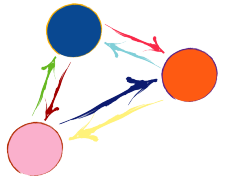
Principes généraux



#1. Savoir de quoi on parle ! Prototype à finalité d'évaluation versus produit final ?



Principes généraux



#1. Savoir de quoi on parle ! Prototype à finalité d'évaluation versus produit final ?

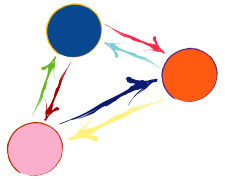
#2. Savoir choisir le langage et l'environnement de programmation appropriés pour l'objectif ciblé : prototype versus produit final. Dans les deux cas, pouvoir itérer !

#3. Savoir qu'itérer implique modifiabilité du logiciel !

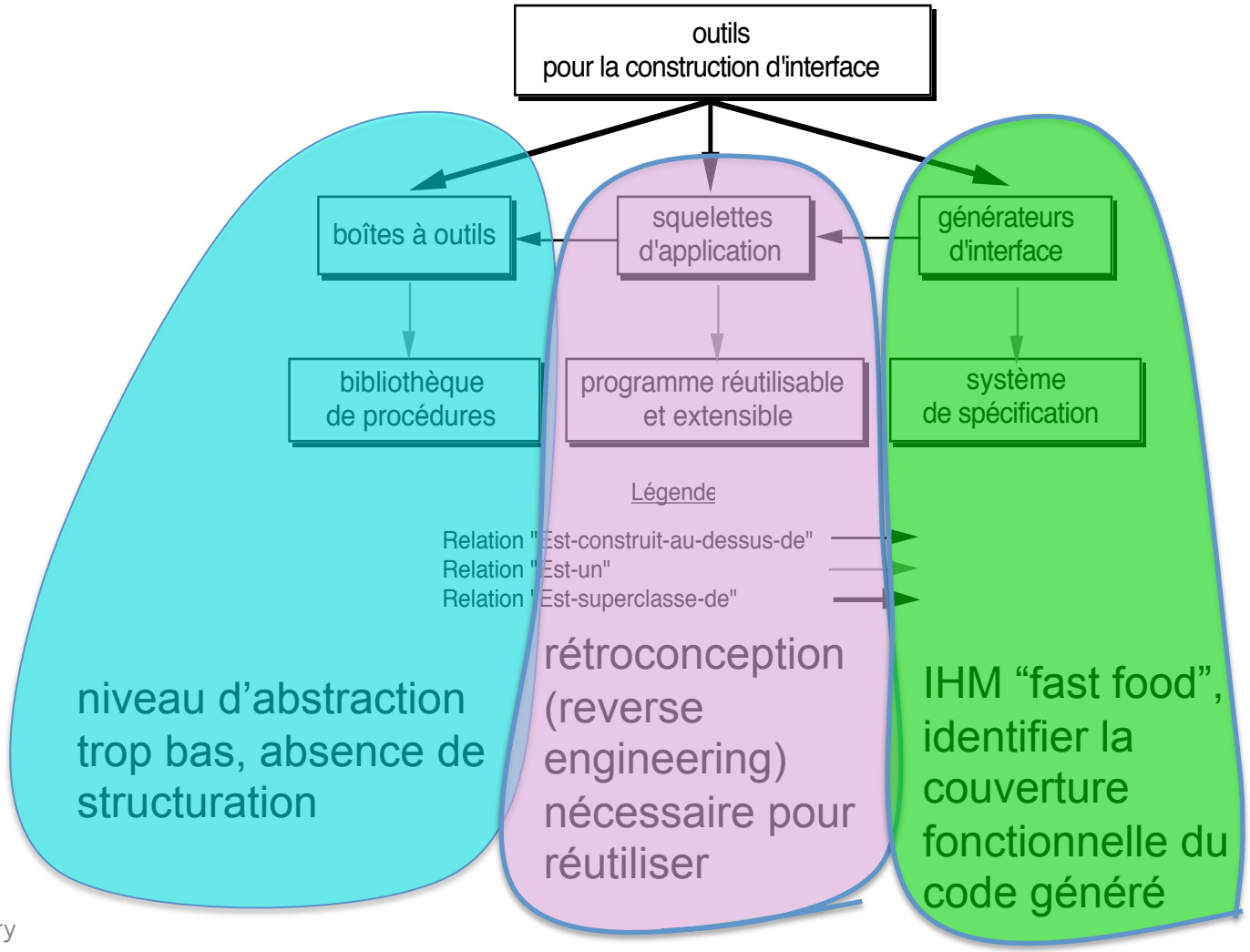
#4. Savoir que l'artisanat est acceptable pour des prototypes, systèmes prospectifs ou systèmes simples mais pas plus !

#5. Savoir que les outils sont utiles mais imparfaits : ne seront pas garants de l'architecture logicielle !

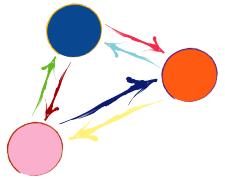
Principes généraux



- Panorama des outils



Principes généraux



#1. Savoir de quoi on parle ! Prototype à finalité d'évaluation versus produit final ?

#2. Savoir choisir le langage et l'environnement de programmation appropriés pour l'objectif ciblé : prototype versus produit final. Dans les deux cas, pouvoir itérer !

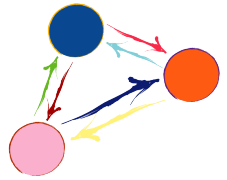
#3. Savoir qu'itérer implique modifiabilité du logiciel !

#4. Savoir que l'artisanat est acceptable pour des prototypes, systèmes prospectifs ou systèmes simples mais pas plus !

#5. Savoir que les outils sont utiles mais imparfaits : ne seront pas garants de l'architecture logicielle !

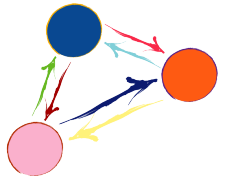
#6. Savoir se référer à des modèles d'architecture logicielle comme cadres de pensée ! Facilitera la modifiabilité donc l'évolution !

Votre savoir-faire est essentiel !

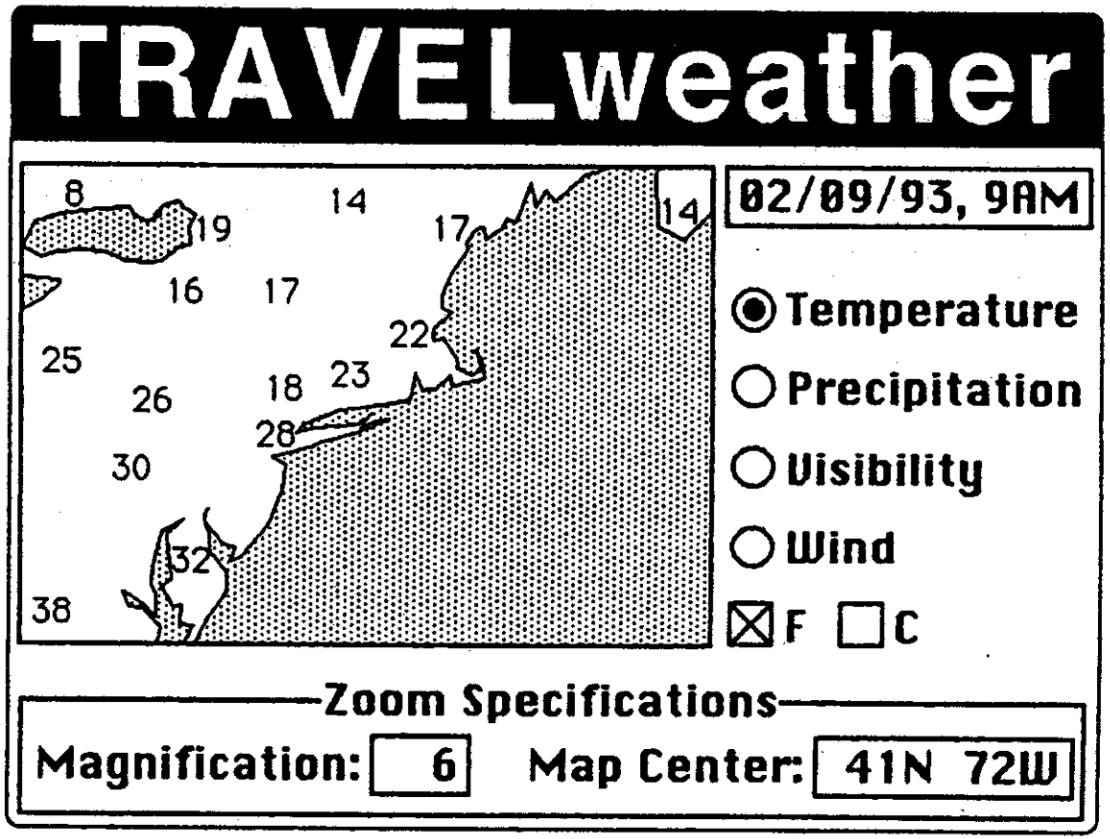


Un exemple pour s’échauffer

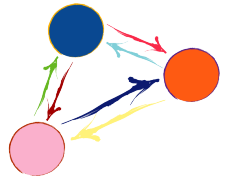
Cas d'étude



- Si je vous dis "Architecture logicielle", que me répondez-vous ?

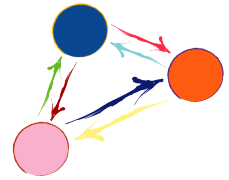


- Si je vous dis « Quelle architecture logicielle », que répondez-vous ?



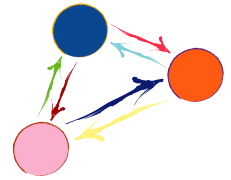
Notions fondamentales

Utilité



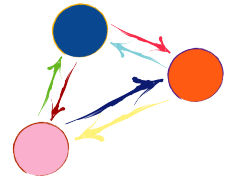
- Finalité d'une architecture
  - Communication (précision et non ambiguïté de la description)
  - Evolution, rétro conception d'un système existant
  - Evaluation (selon des critères qualité)

Définition



- Plusieurs définitions
  - Absence de définition consensuelle
  - Définition du comité IEEE 1471 (2000)
  - Définition de Bass et al.(1998)

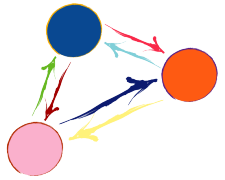
Définition de IEEE 1471



- “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”

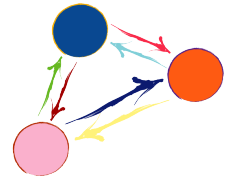
- Autrement dit
  - Une architecture est le résultat d’un processus contraint par l’environnement
  - L’environnement : participants (culture en qualité logicielle, outils, requis commercial…)
  - “Fondamental” dénote les aspects du système qui sont importants pour un participant donné, impliqué dans une étape donnée du processus de développement

Définition de IEEE 1471



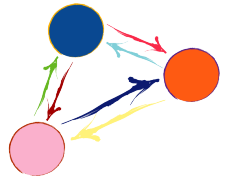
- Distinction entre architecture et description d'architecture
  - Une architecture est un concept : elle existe, bien que non observable
  - Une description d'architecture : représentation de ce concept pour une finalité donnée. C'est une entité concrète

Définition de Bass et al.

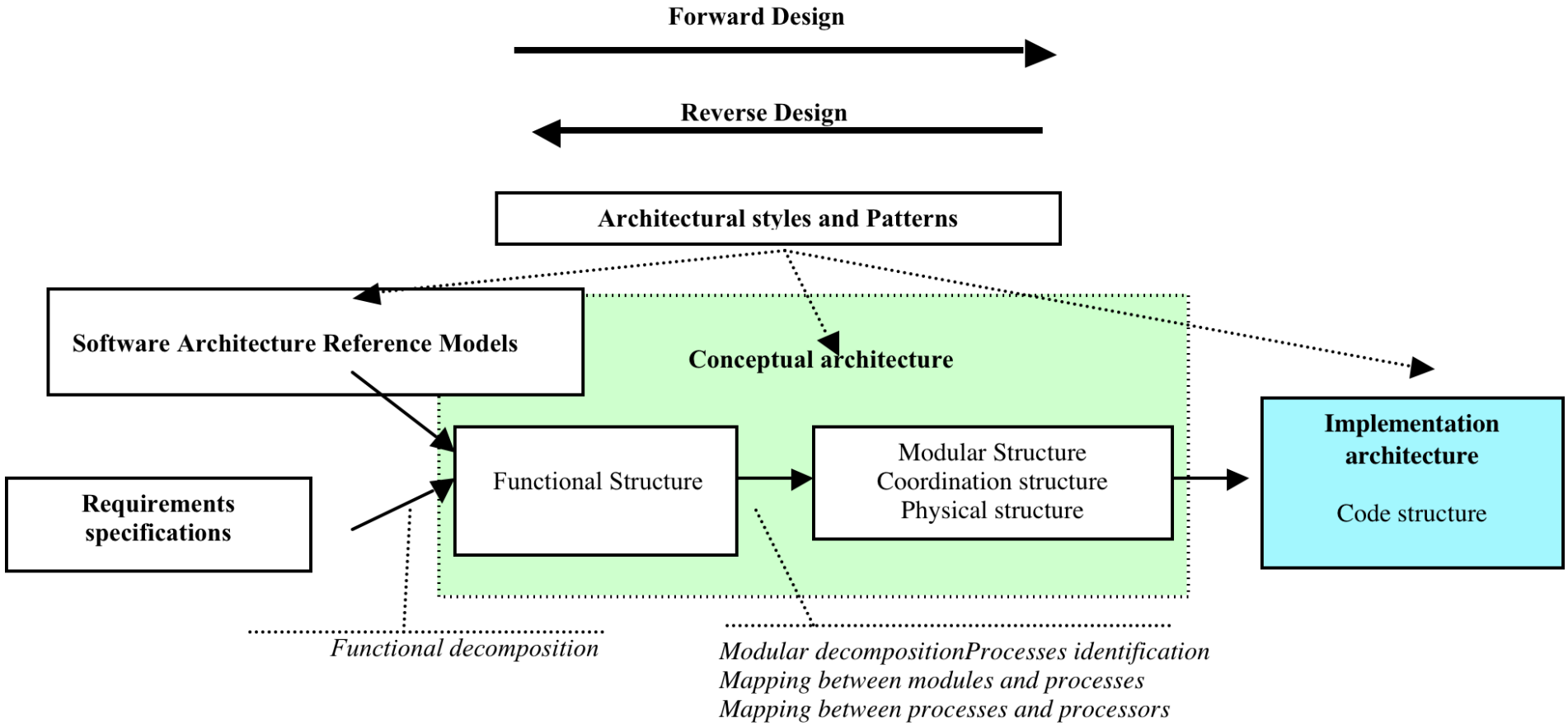


- “A software architecture is a set of structures which comprise software components, the externally visible properties of these components and the relationships among them”
- Autrement dit :
  - Plusieurs points de vue sur une architecture (cf. architecture civile)
  - Un point de vue : une structure, sa représentation pour une finalité donnée
  - Propriétés d’un composant : description du comportement attendu/hypothèses sur le comportement attendu (e.g., services fournis ou requis, performance)
  - Propriétés observables de l’extérieur : un composant est
    - une unité d’abstraction : “a unit of independent production, acquisition, and deployment that interacts to form a functioning system” [Szyperski 97]
    - 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.)

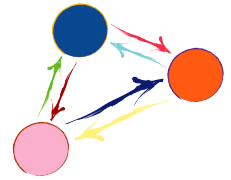
Processus de conception : étapes



- Processus à la fois ascendant et descendant mais des étapes parfaitement identifiées

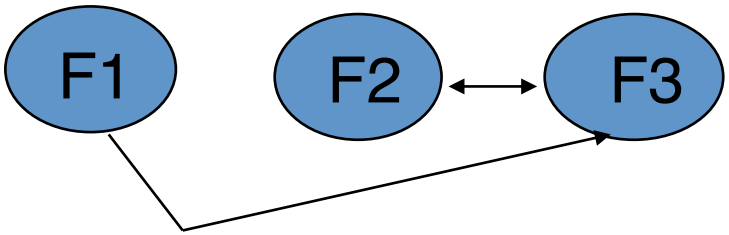


Processus de conception : étapes

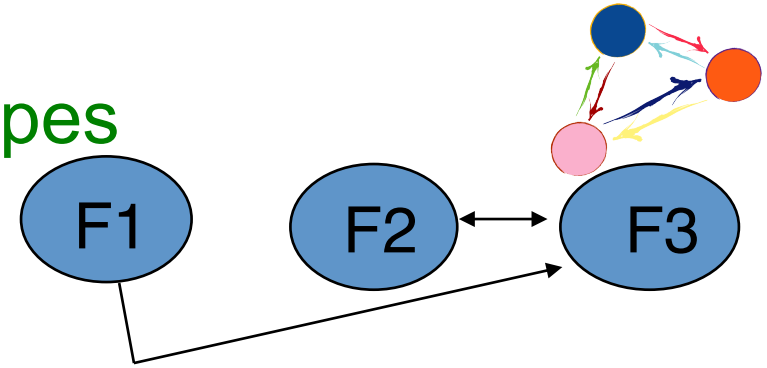


1. Décomposition fonctionnelle

   - Requis fonctionnels -> unités plus simples
   - Relations (“échange des données avec”)



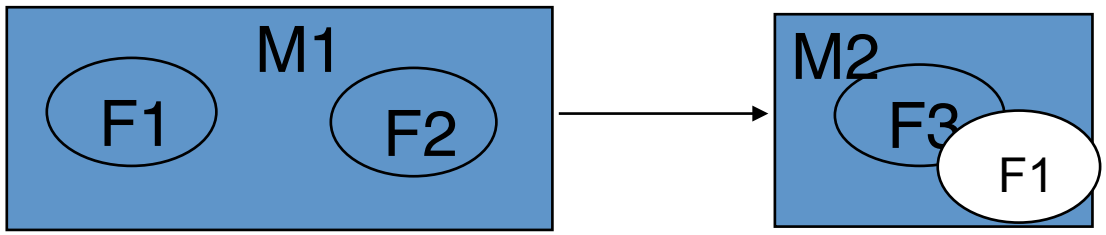
Processus de conception : étapes



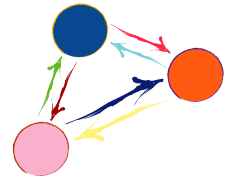
1. Décomposition fonctionnelle

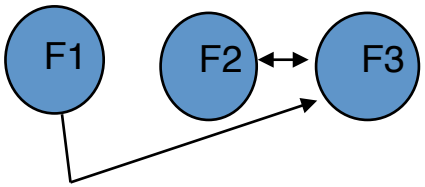
2. Décomposition modulaire : vue statique du système

   - Composants : modules
   - Relations : “est un sous-module de”
   - Allocation des fonctions aux modules = def. couverture fonctionnelle de chaque module (1 fonction/aspect peut être couvert(e) par plusieurs modules, 1 module peut couvrir plusieurs fonctions)

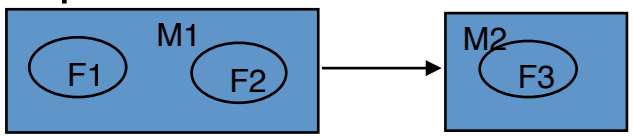


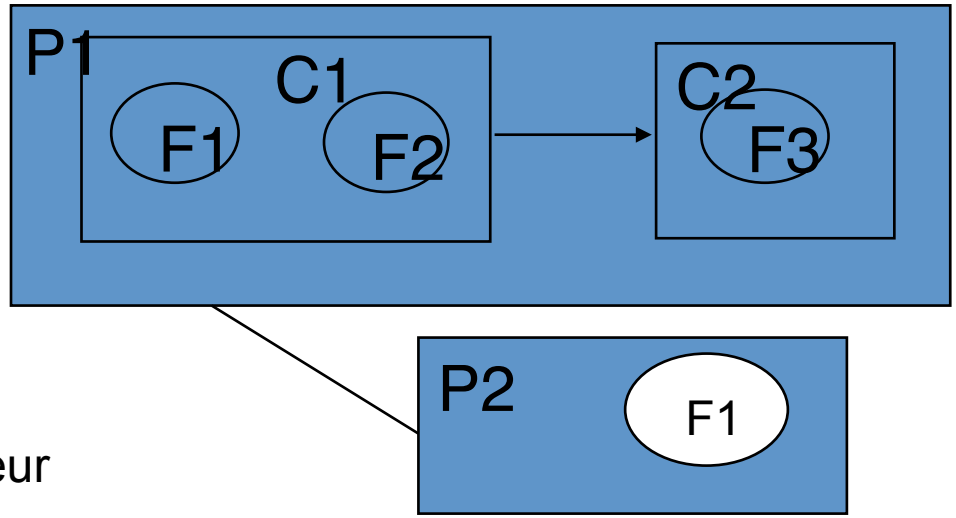
Processus de conception : étapes





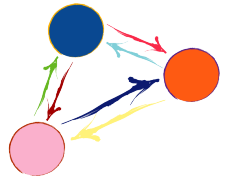
1. Structure fonctionnelle : requis fonctionnel
   - Composant : fonction
2. Structure modulaire : point de vue statique
   - Composant : module
3. Structure de coordination : point de vue dynamique
   - Composant : processus, thread
   - Relation : synchronisation, contrôle d'accès
   - Allocation des modules aux processus





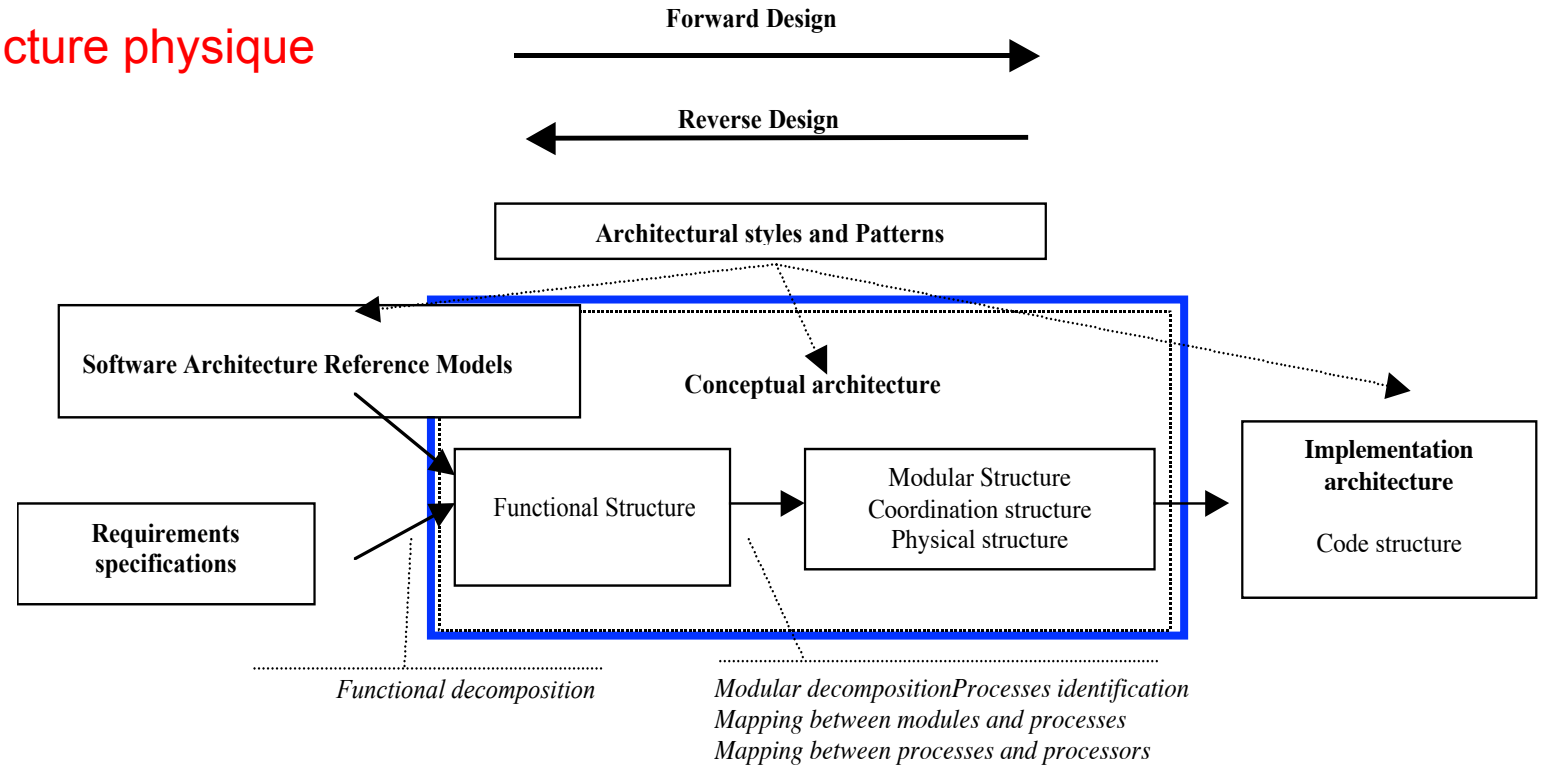
4. Structure physique
   - Composant : processeur
   - Allocation processus-processeur

Processus de conception : étapes

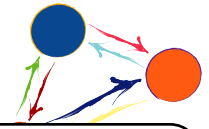


- Une architecture conceptuelle inclut :
  - Structure fonctionnelle
  - Structure modulaire
  - Structure de coordination
  - Structure physique

Peuvent être couvertes par des outils

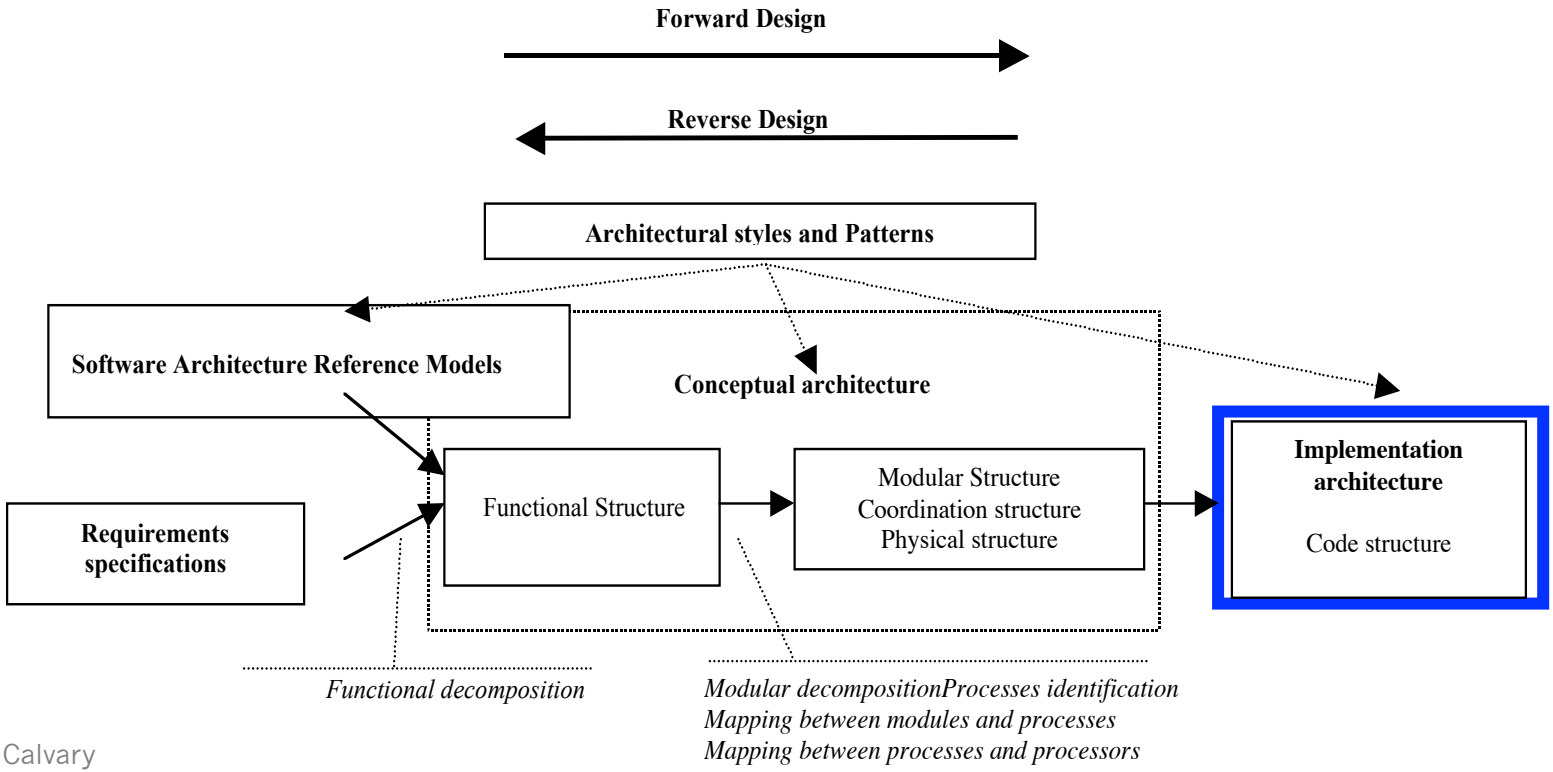


Processus de conception : étapes

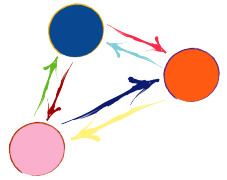


Peut être couverte par des outils

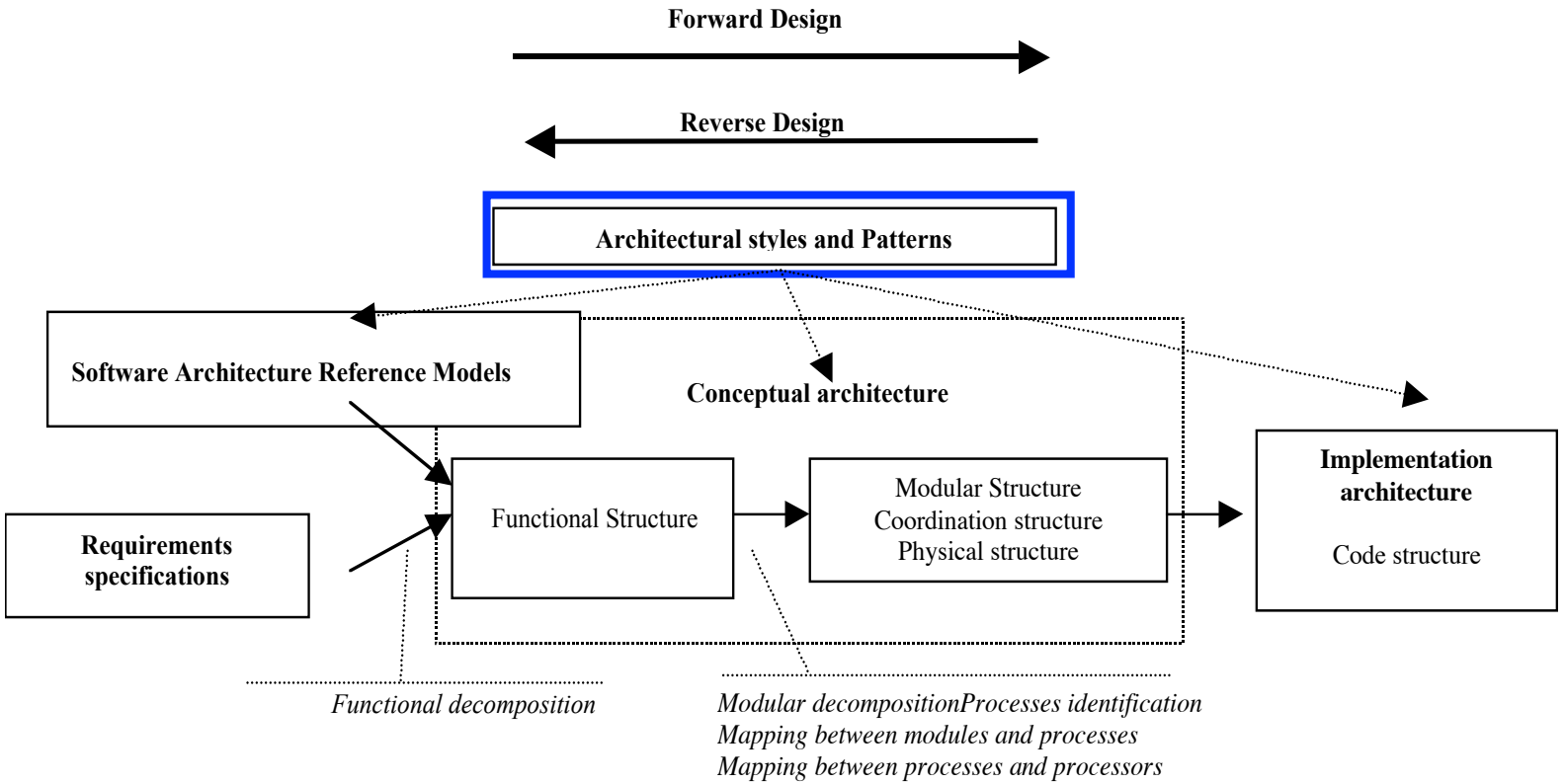
- Architecture implémentationnelle
  - Mise en correspondance des structures de l'architecture conceptuelle
    - Ex: Structure modulaire -> packages, procédures, classes, objets
  - Si besoin, affinement des connecteurs en {composants, connecteurs}



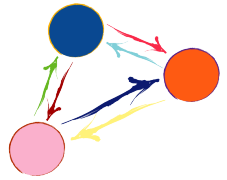
Processus de conception : étapes



- Style d'architecture
  - une notion orthogonale : le choix d'un style peut intervenir dans toutes les étapes du processus de conception d'architecture



Processus de conception : styles

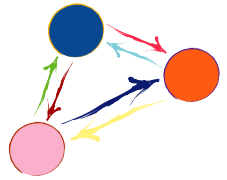


- Exemples
  - *Pipes and filters* (pipe-line)
  - Machines abstraites en couche
  - Client-serveur

- Un style comprend
  - un vocabulaire d'éléments
  - des contraintes de configuration de ces éléments
  - une sémantique pour interpréter une configuration
  - véhicule des propriétés

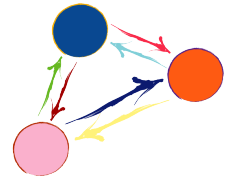
- Un style n'est pas une architecture

Processus de conception : patron

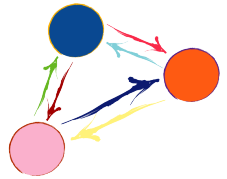


- Une microstructure architecturale en réponse à un micro problème de conception architecturale
- Un micro modèle de référence
- Patrons de Gamma et al.
  - Un nom
  - Description du problème couvert par le pattern
  - Structures conceptuelles qui répondent au problème (structure fonctionnelle et diagrammes qui montrent les interactions entre les composants)
  - Heuristiques spécifiant quand et comment appliquer le patron, les compromis
  - Un exemple d'implémentation en C++
  - Style orienté objet

Conclusion : points essentiels

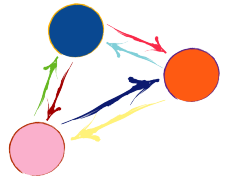


- Une architecture
  - est une chose abstraite
  - revêt de multiples points de vue : descriptions de structures
  - s'évalue au regard de critères définis par avance : elle n'est donc jamais intrinsèquement bonne ou mauvaise → approche par scénarios

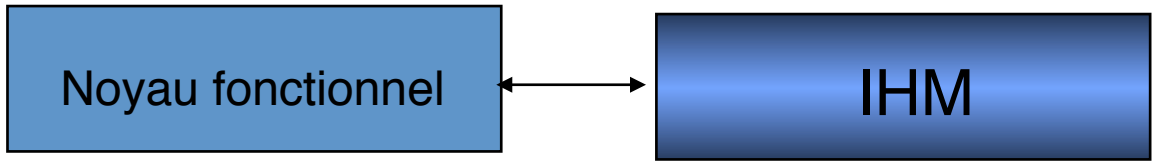


Trois classes de modèles de référence : les modèles à couches

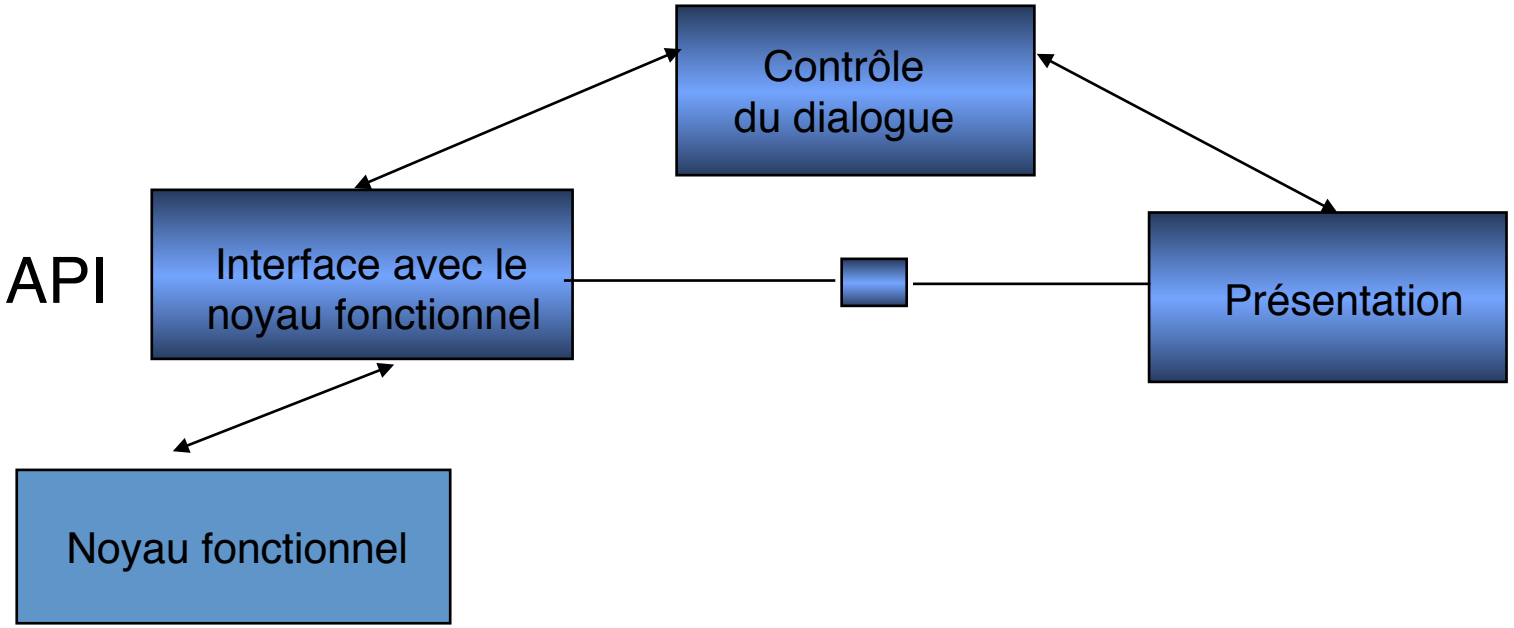
Modèle de référence Seeheim



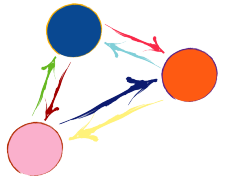
- Fondement



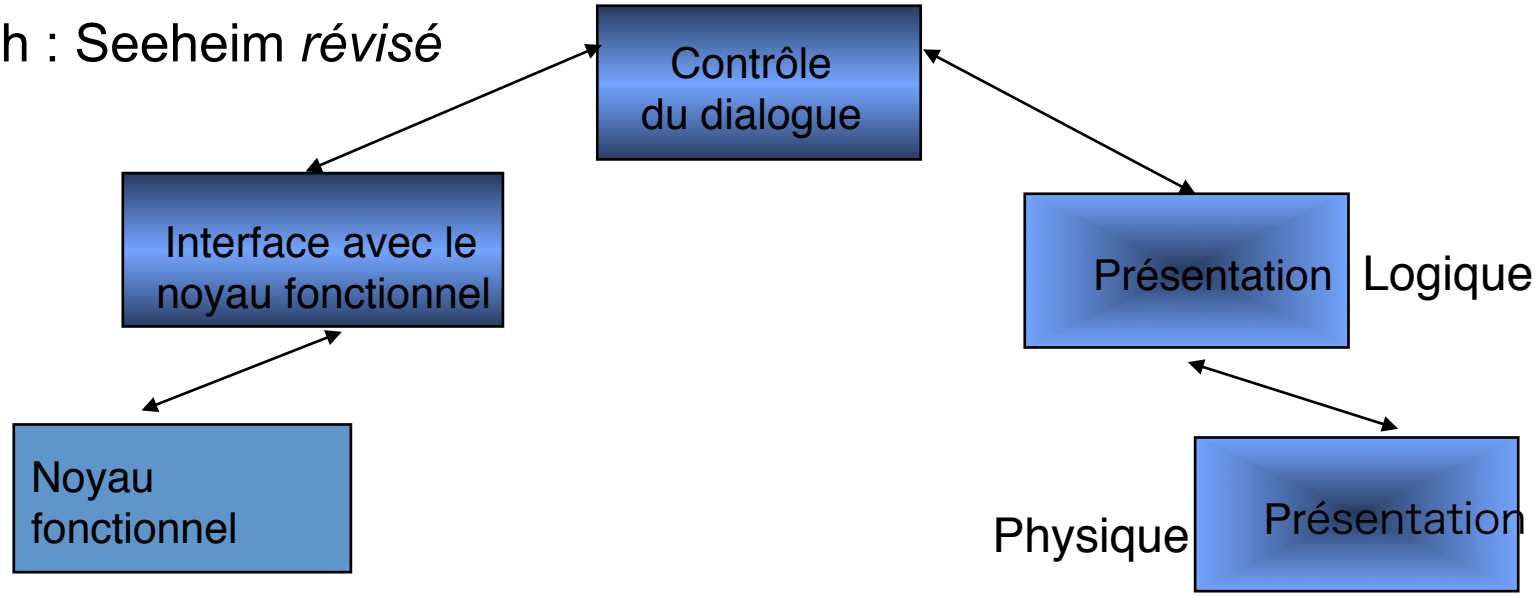
- Seeheim : modèle séminal



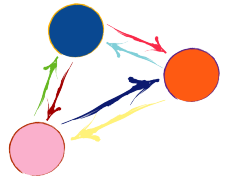
Modèle de référence Arch



- Arch : Seeheim *révisé*

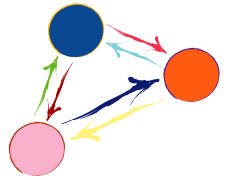


- Métamodèle Slinky : migration fonctionnelle



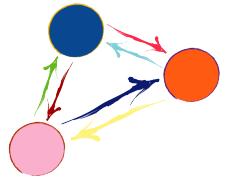
Trois classes de modèles de référence : les modèles à agents

Principes



- 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)

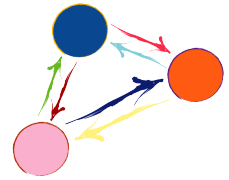
Motivations



- 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

MVC

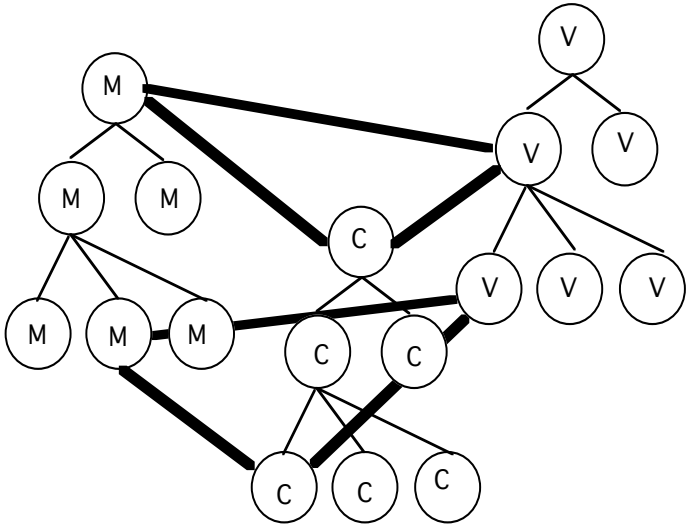


MVC (Smalltalk)

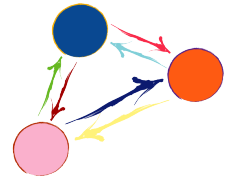
- Model : la compétence abstraite de l'agent (son NF)
- View : le rendu perceptible de l'agent (son comportement en sortie)
- Controller : son comportement en entrée

Aspects réalisation

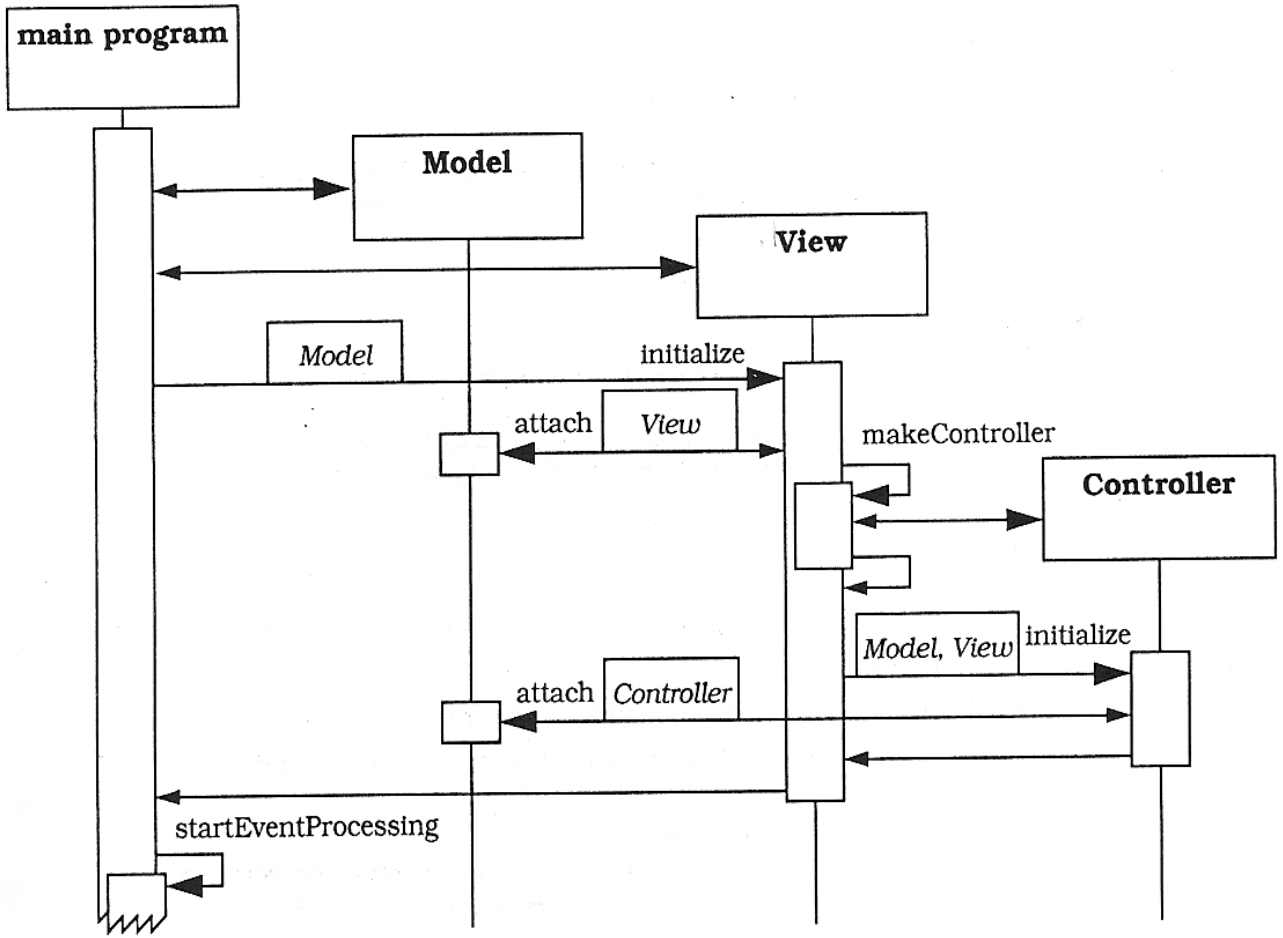
- Un agent : 3 objets Smalltalk (1 par facette)
- Hiérarchie de Models, de Views, de Controllers



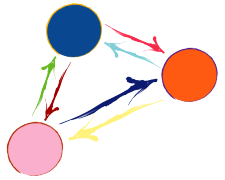
MVC



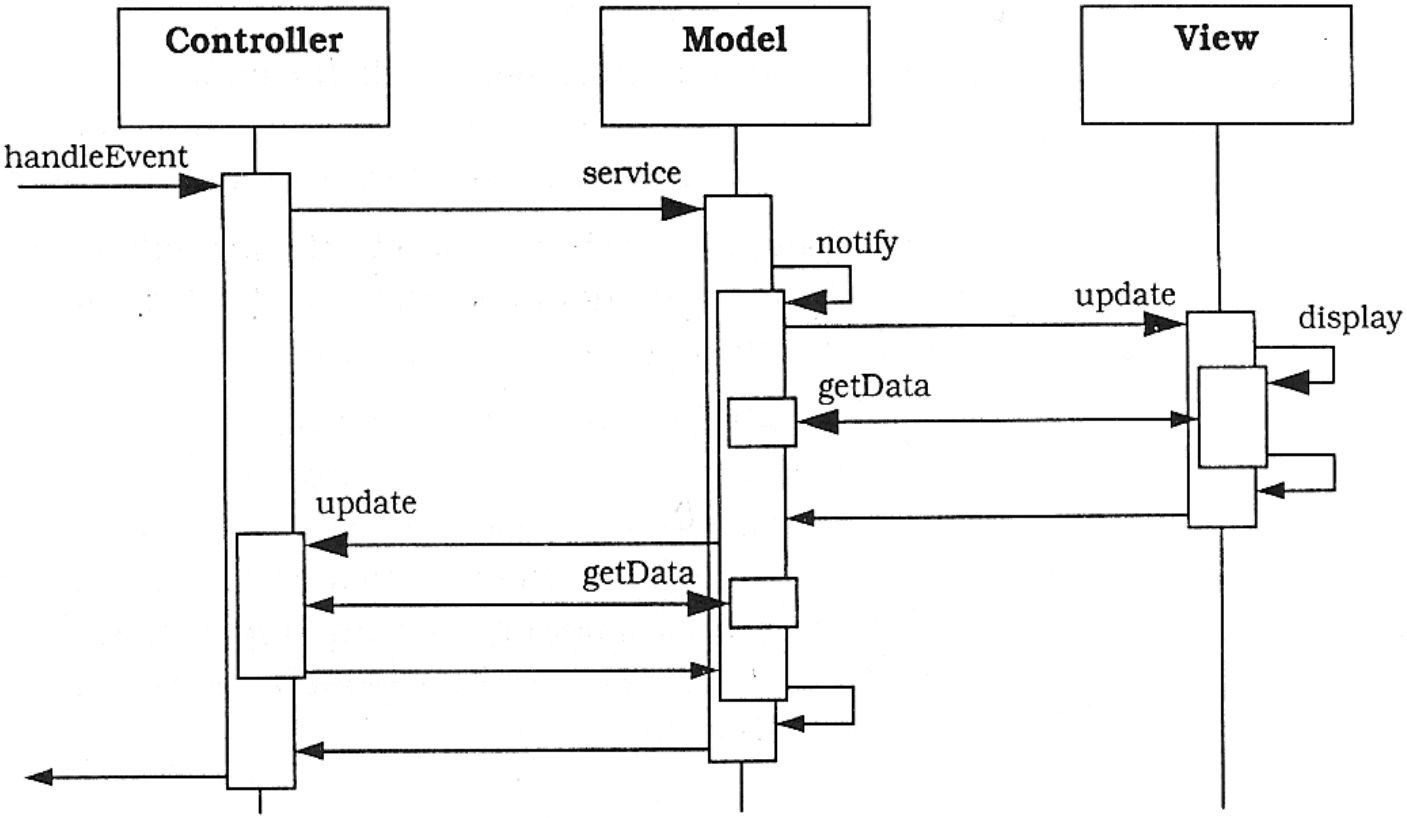
How the facets are created



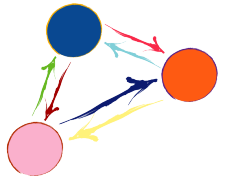
MVC



How the facets communicate



PAC

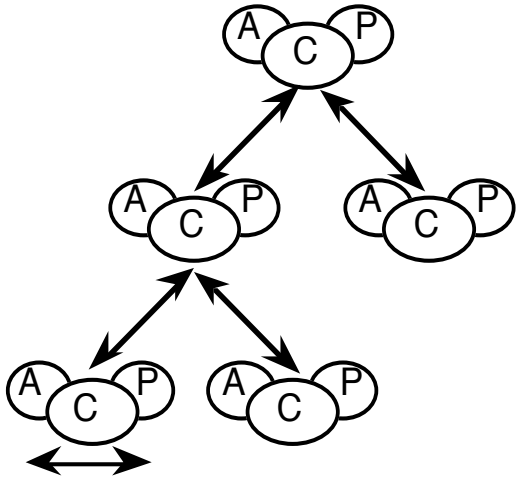


Principes

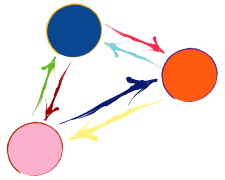
- Abstraction : le M de MVC
- Présentation : le V+C de MVC, le V de ALV
- C : expression des dépendances entre A et P (le L de ALV) échanges avec les autres agents

Aspects réalisation

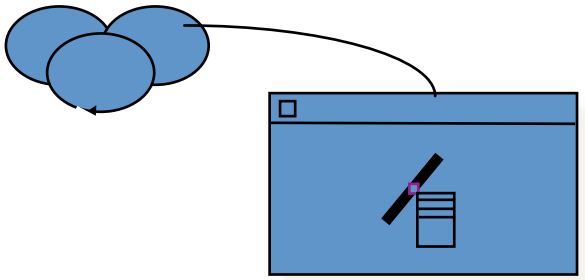
- Aucune recommandation
- dépend de la plate-forme d'accueil
- 1 agent = 1 module C, 1 objet, 1 objet par facette (comme MVC ou ALV)



PAC : heuristiques pour identifier les agents

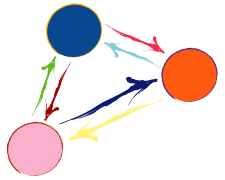


R1 : Si espace de travail (fenêtre, panel)-> un agent



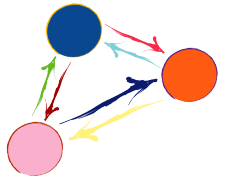
Un espace de travail -> un agent

PAC : heuristiques pour identifier les agents

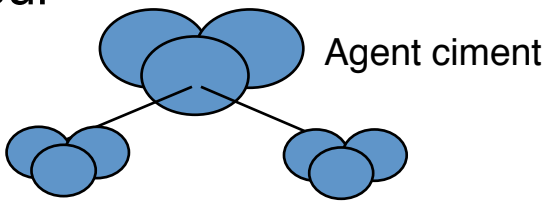


- R2: Si représentation d'objet du domaine complexe -> un agent PAC (par ex., un interacteur inexistant dans la boîte à outils)
  - Le réchaud, le thermomètre
  - Une palette d'outils

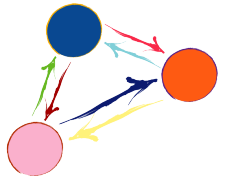
PAC : heuristiques pour identifier les agents



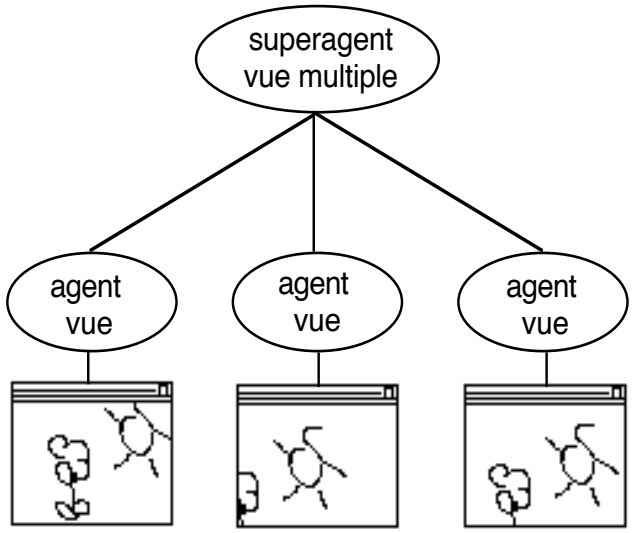
- R3 : Si "analyse syntaxique distribuée" alors agent Ciment syntaxique qui fusionne les actions distribuées en un niveau d'abstraction supérieur



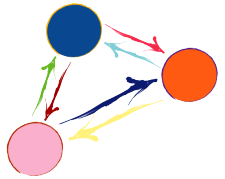
PAC : heuristiques pour identifier les agents



- R4 : Agent Vue multiple pour maintenir la cohérence entre plusieurs vues

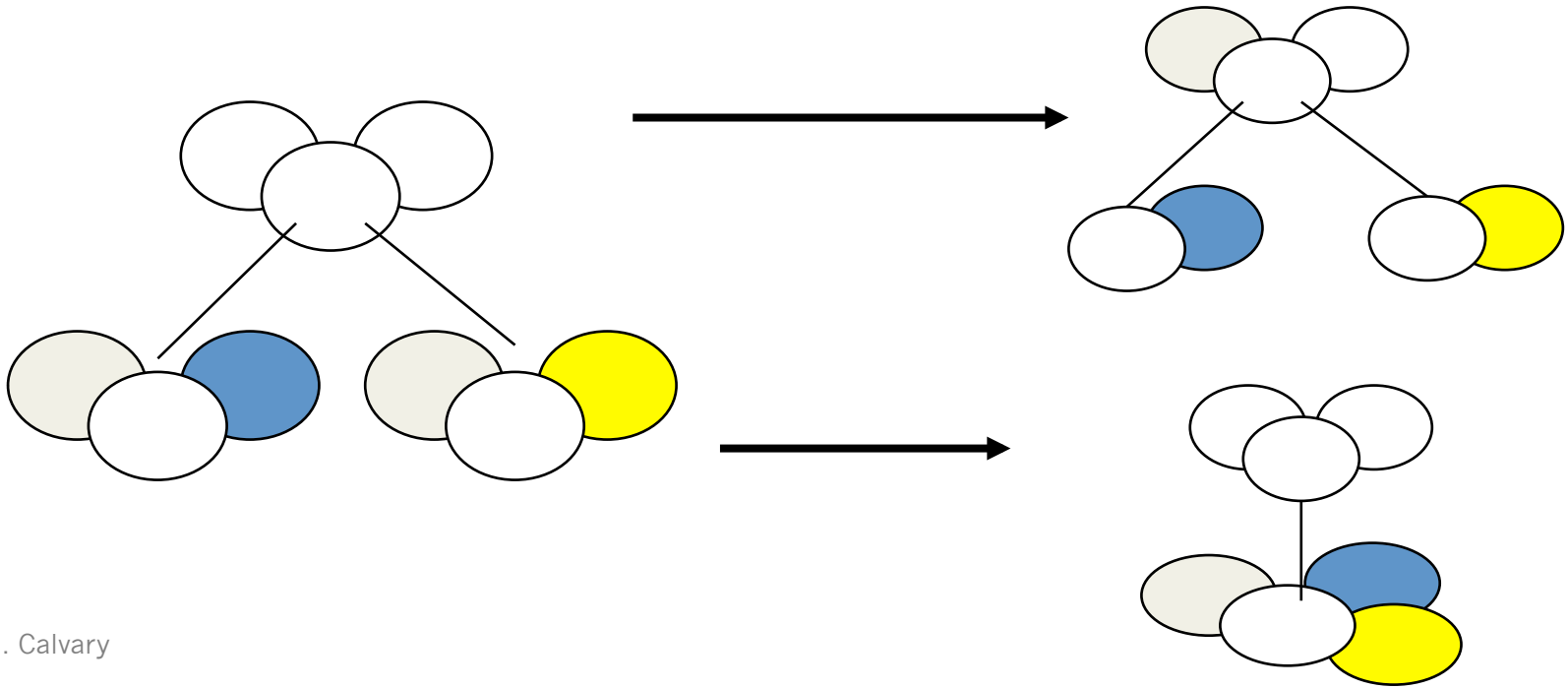


PAC : heuristiques pour identifier les agents

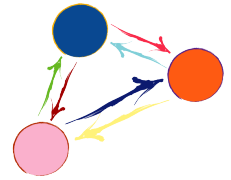


- R5: Optimisation de la hiérarchie d'agents PAC
  - Remonter les fonctions du fils chez le père mais bien penser à l'évolution du logiciel avant de faire cela

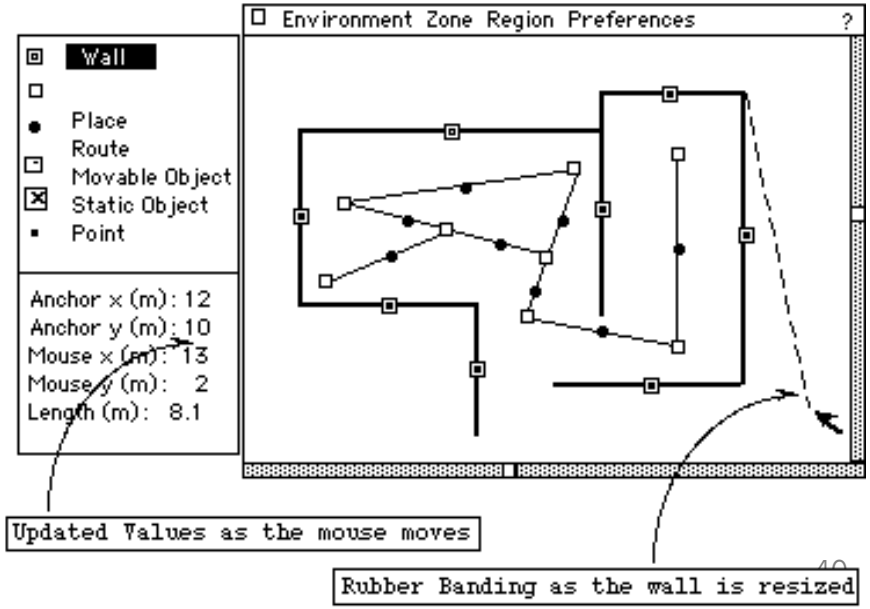
- R6: Si les agents fils partagent la même abstraction, faire remonter ce A dans la partie A du parent - ou bien faire une pâquerette



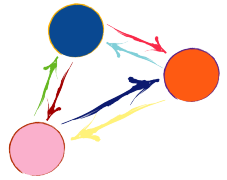
PAC : heuristiques pour identifier les agents



- R7 : Utiliser la délégation sémantique (méta-modèle Slinky) grâce aux A des agents qui peuvent contenir des informations et calcul relevant du NF
  - Dans l'exemple du robot mobile, le A de l'agent mur contient la modélisation de mur dans les termes du domaine (mètres, non pas pixels).
  - Evite les appels au NF pendant le tracé du mur



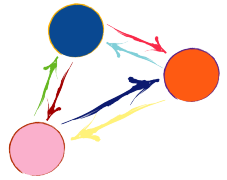
PAC : une implémentation Java



- C

```
package PacModel;

public class Control
{

  protected Abstraction  Abstr = null;
  public Abstraction getA() {return Abstr;}

  protected Presentation Pres  = null;
  public Presentation getP() {return Pres;}

  protected Control PacFather  = null;
```

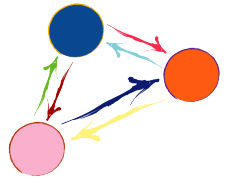
PAC : une implémentation Java



- A

```
package PacModel;

public class Abstraction
{
  // A ref on the control facet
  protected Control Ctrl = null;


  /**
  * Constructor for Abstraction facet. Register the control.
  * @param Crtl A ref on the control facet which manage this facet
  **/
  public Abstraction (Control Ctrl)
  {
     this.Ctrl = Ctrl;
  }
```

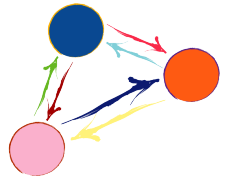
PAC : une implémentation Java



- P

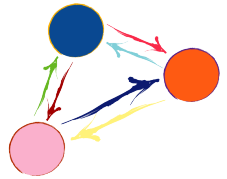
```
package PacModel;

public class Presentation
{
  // A Ref on my controler
  protected Control Ctrl = null;

  /*
  * Contructor. Register the reference on the control.
  * @param Ctrl The reference on the control attached with this facet
  */
  public Presentation (Control Ctrl)
  {
    this.Ctrl = Ctrl;
  }
```

PAC : une implémentation Java



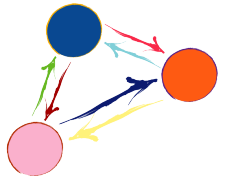
- A of MyApp

```
package MyApp;

import PacModel.*;

public class A_MyApp extends Abstraction
```

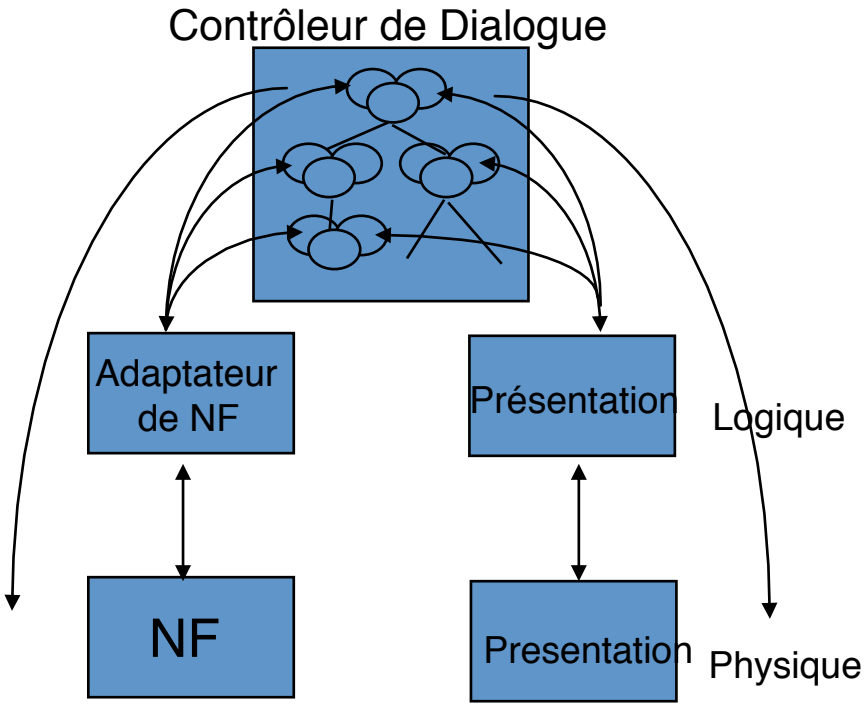


Trois classes de modèles de référence : les modèles hybrides

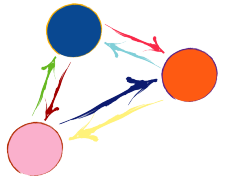
Motivations et exemple : PAC-AMODEUS



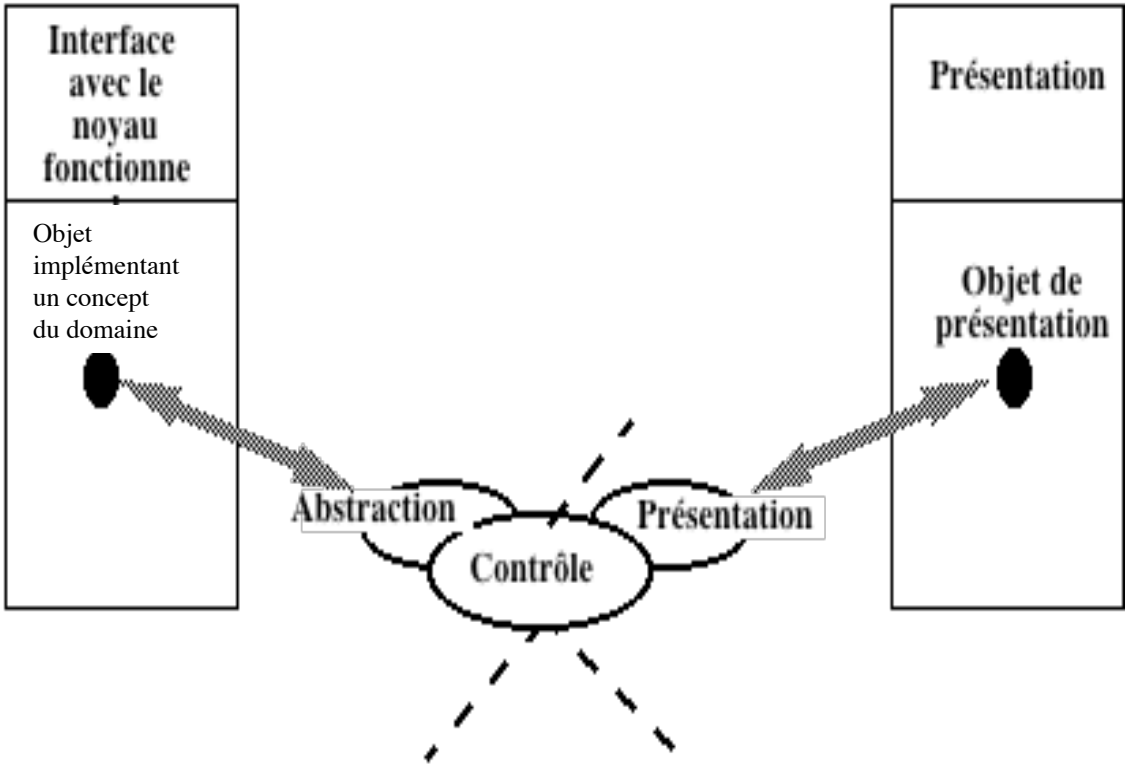
- Conserver la décomposition fonctionnelle de Arch
  (bon cadre de raisonnement)
- Conserver la modularité et le parallélisme des modèles à agents
- Permettre la délégation sémantique dans l'IHM via les facettes A des agents (performance)



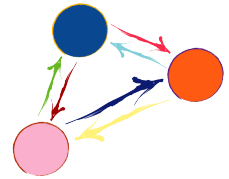
PAC-Amodeus



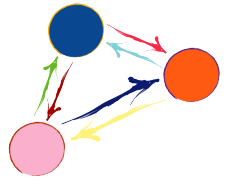
- Agent PAC :



PAC-Amodeus

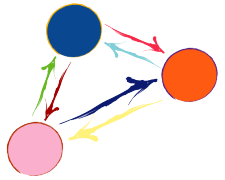


- 1. Liaison avec les objets implémentant les concepts du domaine
  - un agent lié participe à la chaîne de transformations du monde Noyau Fonctionnel vers le monde IHM concrète
  - un agent non lié est indépendant du domaine (améliorations conceptuelles)
- 2. Les facettes d'un agent
  - un agent sans A (ou A minimaliste)
    - est une extension de la boîte à outils
    - est en liaison directe avec un objet-concept 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)

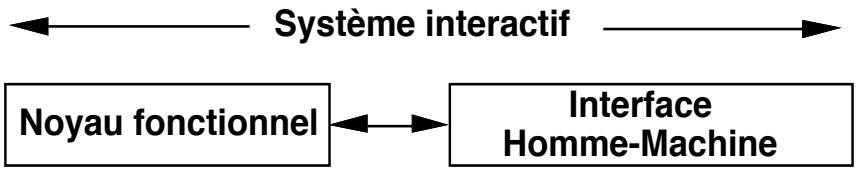


Conclusion générale

Les grands messages à retenir



- L'architecture logicielle est un bien nécessaire
- Les repères fondamentaux sont :



Modèle ARCH

Modèles multi-agents : MVC, PAC

Modèle hybride : PAC-Amodeus

- « Rien ne se perd, tout se transforme ! » Les efforts, en phase amont, de conception ergonomique paient pour la qualité du système, y compris la qualité logicielle interne : ils paient en conception logicielle aussi !