# Interface Hommes/Systèmes

## *II - Architectures*

- - - - - - - - - -

**1 – Architectures logicielles**

**2 – Architectures d'IHS**

- - - - - - - - - -

*ENSMA A3-S5 - période A*

*2023-2024*

M. Richard
richardm@ensma.fr

# Sommaire

# I - Architectures logicielles

*1 – Définitions*
*2 – Architecture logique*
*3 – Architecture physique*
*4 – Concrètement. . .*

## ↪ Structuration d'une application

- On distingue 2 catégories d'architecture :
  - Architecture selon un découpage logique
  - Architecture selon un découpage physique

- l'application peut être structurée selon différentes vues
  - Vue en couches (layer view) = vue logique
    - montrant le découpage fonctionnel de l'application
    - indépendante des contraintes physiques (machine, serveur, ...)
    - modèle de référence en 5 couches
  - Vue en niveaux (tier view) = vue physique de la structuration de l'application

## ↪ Objectifs de la vue en couches

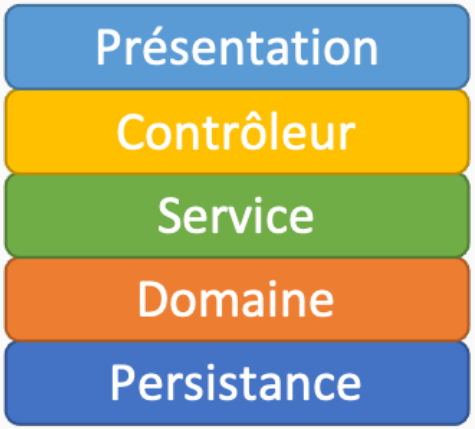
- maitriser la complexité des applications
- optimiser le temps de développement
- localiser les problèmes d'enchainement
- améliorer et structurer la communication entre composants

## ↪ Lien entre couches :

- utilisation des *interfaces logicielles* pour le passage d'une couche à une autre (notion d'API)
- utilisation du *pattern design DTO* (Data Transfer Object) pour la liaison de certaines couches

## ↪ Modèle de référence en 5 couches

- le modèle de référence propose une structuration en 5 couches :
  - Présentation
  - Contrôleur
  - Service
  - Domaine
  - Persistance




- chaque couche a son propre rôle, i.e. sa propre *responsabilité*
- chaque couche utilise les services offerts par la couche inférieure et propose des services à la couche supérieure

## ↪ Couches Présentation et Contrôleur

- Couche Présentation :
  - Responsable de l'interface graphique de l'application
    - construction graphique
    - gestion des événements
  - on disingue ici 3 types de clients :
    - *Client léger* : aucun déploiement sur le poste client. L'interface est construite par le serveur et envoyé au client via un navigateur Web
    - *Client lourd* : tout l'interface est déployée sur le poste client
    - *Client riche (Smart client)* : compromis entre les 2 précédents ; interface via un navigateur, mais une partie du code est exécutée dans le navigateur et donc sur le poste client
- Couche Contrôleur :
  - Responsable du contrôle du dialogue entre la présentation et les services de l'application
    - filtre des événements
    - cohérence Interface/Services proposées
- Comme nous le verrons plus tard, cette partie du modèle de référence est le plus souvent améliorée par la mise en place d'une architecture spécifique (MVC, MVP, MVVM...)

## ↪ Couches Service et Domaine

- Couche Service :
  - correspond aux traitements que propose l'application
  - assure l'implémentation de la logique des *use-case* fonctionnelle

- Couche Domaine :
  - implémente le *modèle* métier
  - on parle *d'objets métiers*
    - correspond à une coloration des données manipulées par les règles adaptées au contexte de l'application
    - ils offrent des services élémentaires qui seront utilisés par la couche service pour proposer les use-case
- Ces deux couches peuvent être regroupées dans une seule : la couche *Business*
  - en fonction de la complexité de l'application, couche service et couche domaine peuvent être fusionnées

## ↪ Couche Persistance

- Présente si nécessité de stocker des données :
  - assure la persistance complète du système d'information en utilisant une ou plusieurs source·s de données :
    - SGBD, XML, Excel...
  - définit un niveau d'abstraction des données (*pattern DAO* (Data Access Object))
    - assure l'accès CRUD (Create, Read, Update, Delete) à la source de données
    - offre une vision objet des données quel que soit le modèle de stockage utilisé

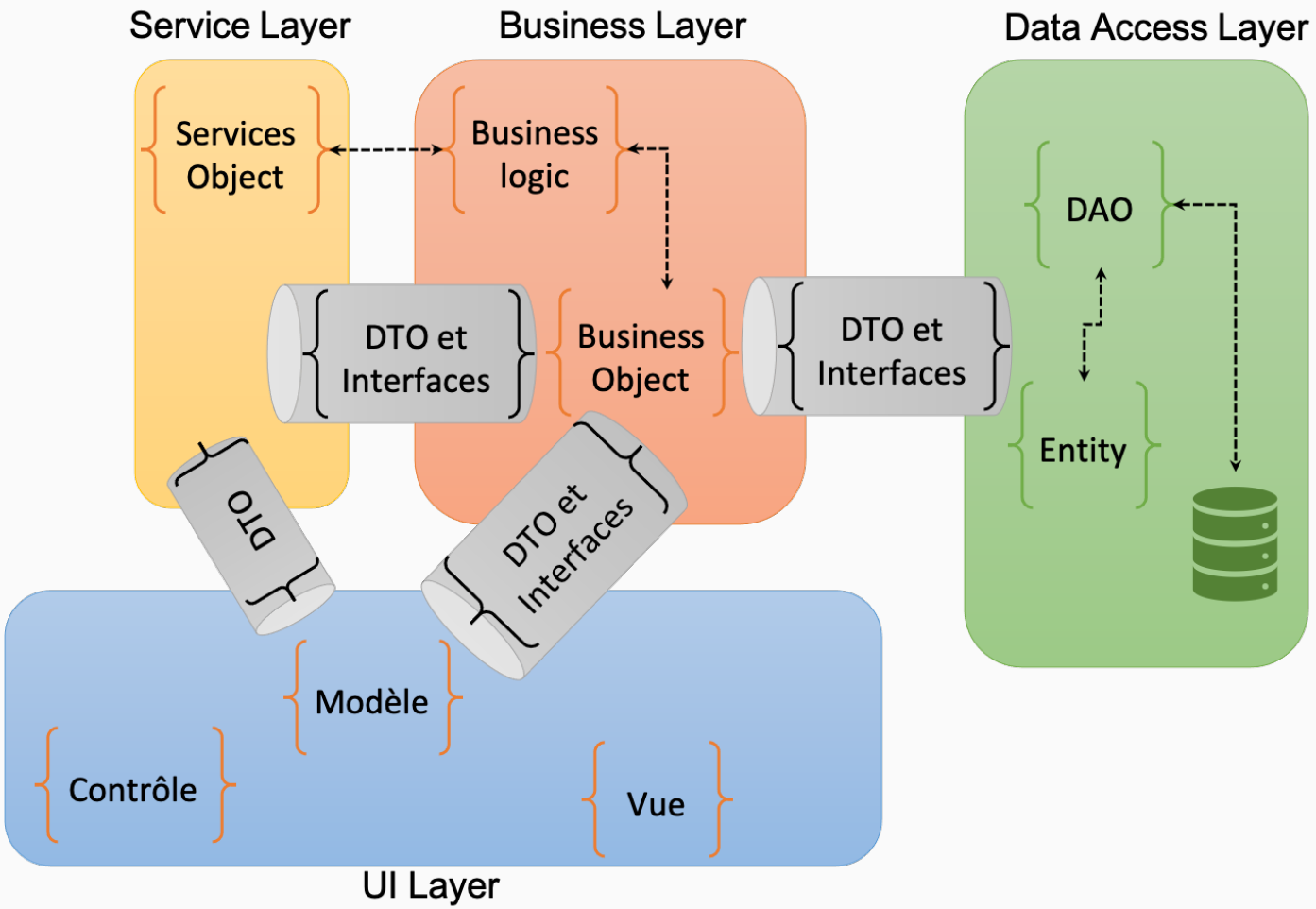
## ↪ En conclusion :

- ce modèle en couche propose un cadre pour la conception d'une architecture logicielle
- ce modèle peut être adapté aux différents types d'application
  - regroupement de couches ou découpage plus fin si nécessaire

## ↪ Vue physique ou vue en niveau

- la vue en niveau (tier view) permet d'exprimer le fait que les composants peuvent être répartis sur des composants matériels distincts
  - un changement de niveau dans cette architecture est définie par l'utilisation d'un intermédiaire de communication par un composant pour en utiliser un autre
- On distingue communément 3 modèles définissant un nombre de tiers différent :
  - modèle à 1-tiers :
    - toutes les couches de la persistance à la présentation s'exécute sur le même matériel
  - modèle à 2-tiers :
    - correspond au modèle client/serveur de première génération (utilisation de SGBD sur un serveur commun pour la couche persistance)
  - modèle à n-tiers :
    - modèle générique permettant la définition d'un nombre N de niveaux
    - apparu avec le développement des applications Web
    - les couches présentation, services et domaines pouvant être distribuées (3-tiers, 4-tiers, 5-tiers)

## ↪ Une architecture complète et cohérente



# II - Architectures d'IHS

*1 – Principes*
*2 – Approche globale*
*3 – Approche par composants*
*4 – Conception du dialogue*

## ↪ Objectifs

- Définir un modèle d'architecture pour la *couche interface utilisateur*
- Cadre de structuration des composants logiciel constituant une interface

## ↪ 2 familles de modèle :

- Approche globale
  - L'interface globale est considérée comme un unique composant
    - modèles ARCH, SEEHEIM
- Approche par composants
  - L'interface est structurée en différents composants (réutilisables) et pouvant communiquer entre eux
    - Modèle MVC, MVP, MVVM

## ↪ Pour quel type de client :

- dans le cadre de ma partie, nous nous intéressons uniquement à la conception de clients lourds

## ↪ Principes

- Découpage du système en modules
- chaque module possède des responsabilités bien déterminées

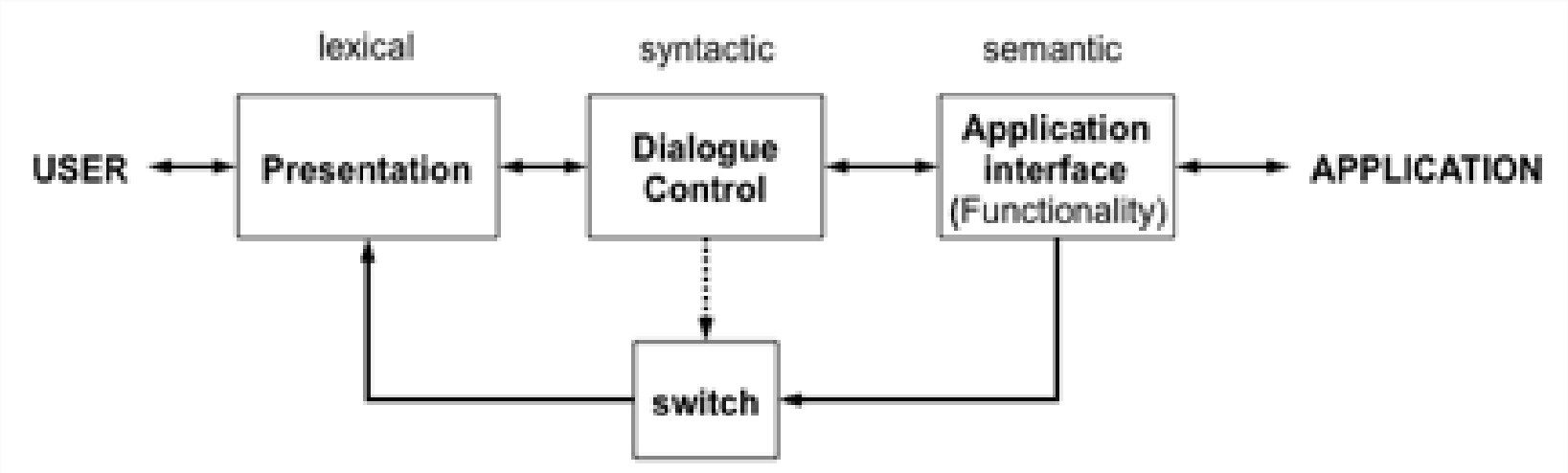
## ↪ Intérêts

- conception itérative et modulaire
- cadre de pensée pour le réalisateur

## ↪ Limites

- centralisation très poussée
- imprécision des échanges entre modules

## ↪ SEEHEIM 1/2


lexical
syntactic
semantic
USER
Presentation
Dialogue Control
Application interface (Functionality)
APPLICATION
switch

## ↪ SEEHEIM 2/2

*Présentation (P) :*

- Image du système
- Point d'entrée des commandes pour l'utilisateur ⇒ *Lexique*
  - Lexique : prise en compte des E/S vers l'utilisateur

*Contrôleur de Dialogue (CD) :*

- Médiateur entre l'Interface avec l'Application et la Présentation ⇒ *Syntaxe*
  - Syntaxe : Séquencement & formatage des phrases à partir du lexique
    - pour l'accès aux fonctionnalités
  - Interprétation des phrases
    - pour leur présentation dans le lexique de l'utilisateur
- Gestionnaire de l'état de l'interaction

*Interface avec l'application (IA) :*

- Abstraction de l'Application ⇒ *Sémantique*
  - Sémantique :
    - Mise à disposition des fonctionnalités du système
- Vérification sémantique des requêtes envoyées à l'Application

## ↪ Objectifs

- Un système interactif est défini comme une *collection d'agents*.

*Un agent/composant comporte :*

- Un état interne
- Une certaine connaissance du domaine
- Une capacité d'action et de réaction

*Pourquoi une approche par composant :*

- Palier les lacunes des modèles globaux
- Modularité et parallélisme
- Très proche de l'approche objet

*Les différents modèles à agents historiques :*

- *PAC*
- *MVC*
- *MVP*

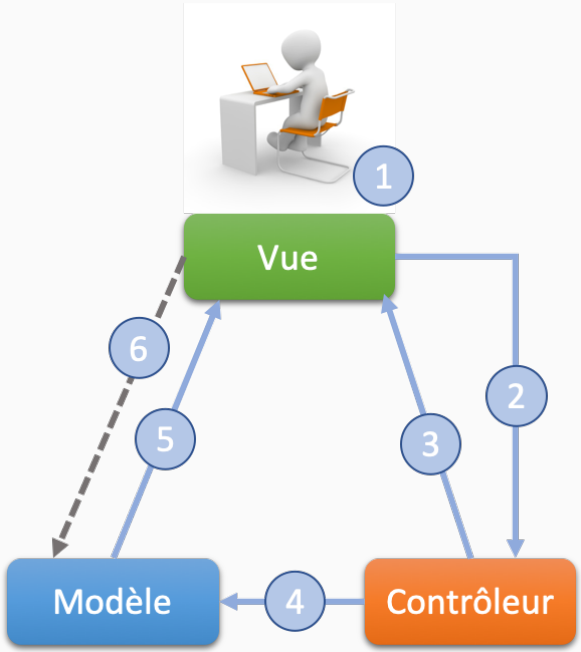
## ↪ Modèle MVC - *Modèle-Vue-Contrôleur*

Origine :

- Issu du langage SmallTalk
  - 1980
  - Utilisé comme modèle pour les composants de la BO SWING

Structure & Fonctionnement :

- *Hiérarchie d'agents* communiquant
- Découpage de chaque agent en :
  - une *entité Modèle*
  - une *entité Contrôleur*
  - une *entité Vue*
- lien entre Vue et Contrôleur par l'entité ≪ Modèle ≫

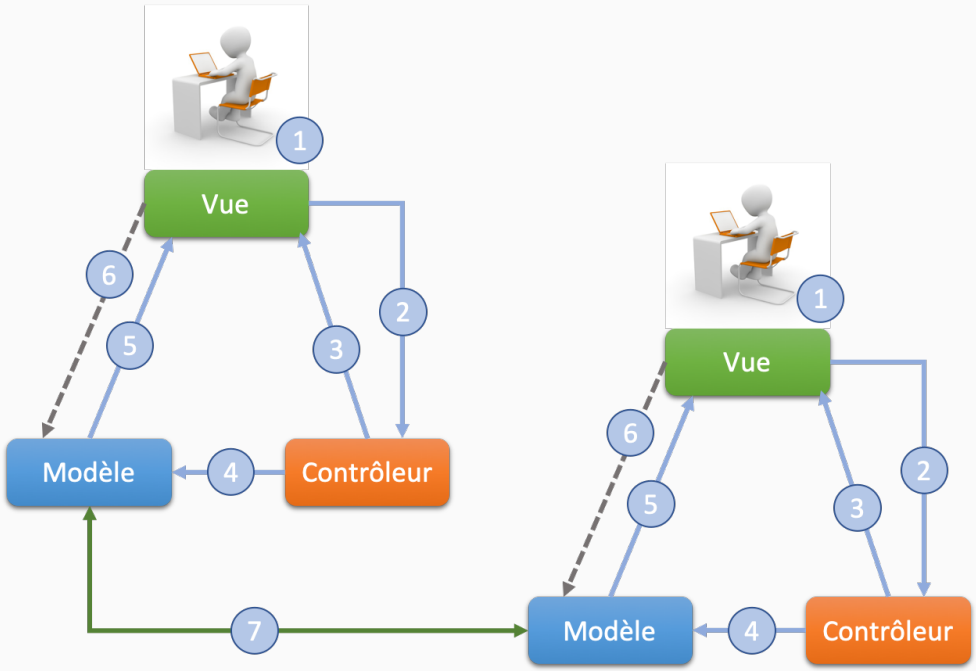


## ↪ Modèle MVC - Multi-composants

- Hiérarchie d'agents *liés par leur entité Modèle*
- assure la cohérence sémantique

Structure Hiérarchique des entités Vue :

- Assure la cohérence graphique
- fenêtre, sous-fenêtres, boutons, etc.
- lien fait le plus souvent par un *pattern observateur*



## ↪ Modèle MVC - Fonctionnement

- *Modèle* :
  - réunit l'accès à l'ensemble des fonctionnalités du système
  - prévient certains composants des modifications :
    - Vue ou autres modèles en relations
- *Vue* :
  - définit la présentation (en sortie uniquement. . .)
  - interroge le modèle pour l'affichage des modifications survenues
  - prévient le contrôleur des modifications affectant les entrées
- *Contrôleur*
  - gère et interprète les actions de l'utilisateur
  - prévient le Modèle et la Vue des modifications

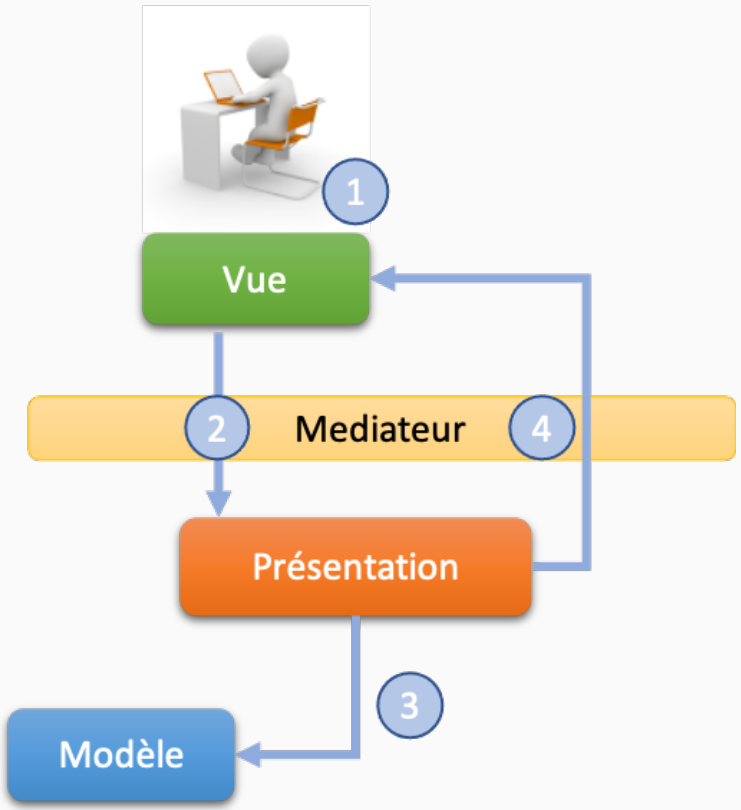
*Bilan*

- *Avantages :*
  - Capacité d'extension
  - Utilisation du pattern observateur entre vue et modèle
  - Cohérence globale de l'application grâce à la communication entre modèle
- *Inconvénients :*
  - Élément du pattern assez fortement couplé (malgré le pattern observateur)
  - Dialogue potentiellement réparti dans différents composants
  - Boucle d'action longue et assez complexe à mettre en œuvre

## ↪ Modèle MVP - *Modèle-Vue-Présentation*

- Amélioration de MVC
- Objectif :
  - réduire le couplage entre Vue et Modèle
    - utilisation d'un pattern médiateur
  - centraliser le dialogue

*Architecture :*



## ↪ Modèle MVP - Fonctionnement

- *Modèle* :
  - uniquement dédié au pont avec la logique métier
  - totalement déconnecté du dialogue avec l'utilisateur
  - totalement indépendant de la vue
- *Vue* :
  - uniquement responsable de la gestion graphique (en entrée et en sortie)
  - n'implémente aucune logique
  - indépendante de la présentation (par interface)
  - indépendante du modèle (aucune liaison)
- *Présentation* :
  - contient toute la logique de présentation
  - implémente le dialogue utilisateur
  - indépendante de la bibliothèque graphique
  - il s'agit en fait ici du pattern médiateur

## ↪ MVP multi-agents

- Hiérarchie d'agents *liés par leur entité Présentation*
- assure la cohérence sémantique

## ↪ Introduction 1/2

*Propriétés des systèmes interactifs : trois grandes catégories*

- *La complétude de la tâche* :
  - l'utilisateur doit pouvoir effectuer avec succès l'ensemble des tâches prévues dans l'application
- *Les propriétés de souplesse* :
  - s'intéressent à la diversité des moyens dont disposent système et utilisateur pour communiquer entre eux
    - atteignabilité,
    - non-préemption,
    - ...
- *Les propriétés de robustesse* :
  - prend en compte la sûreté de l'interaction

Le·s formalisme·s choisi·s devra·ont donc permettre d'exprimer ces différentes propriétés.

## ↪ Introduction 2/2

*Choix du formalisme de spécification :*

- Si formalisme inadapté :
  - activité de spécification difficile et temps de spécification plus long
  - source d'imprécisions et erreurs de conception
- Comment choisir ?
  - compromis entre les différentes propriétés des différents formalismes considérés
  - Différentes propriétés peuvent être décrites par la définition de critères
- Critères d'évaluation des formalismes :
  - sont classés en trois catégories :
    - Le pouvoir d'expression,
    - Les capacités génératrices
    - Extensibilité et utilisabilité

*Modèles, formalismes et notations utilisés :*

- répertoriés en quatre catégories, en fonction des théories desquelles ils sont issus :
  - Des sciences cognitives,
  - De la théorie des graphes,
  - Des approches algébriques/logiques,
  - Des approches hybrides.

## ↪ Les automates

- *Sommets* :
  - correspondent aux états du dialogue entre l'utilisateur et le système
- *arcs* :
  - correspondent aux évènements utilisateurs ou systèmes

*Trois types d'automates :*

- *Les diagrammes de transition* :
  - modélisation du langage d'entrée
  - réaction du système : modélisé en étiquetant chaque état par une action déclenchée lorsque l'état est atteint
- *Les réseaux de transition récursifs (RTN)* :
  - Permet le découpage du dialogue en sous-dialogues ⇒ Modularité
  - Appel d'un sous-dialogue : en étiquetant un arc par le nom du sous-dialogue
- *Les réseaux de transition augmentés (ATN)* :
  - considère le contexte grâce à l'ajout de registres
  - les valeurs des registres sont des variables du dialogue

*Problème majeur :*

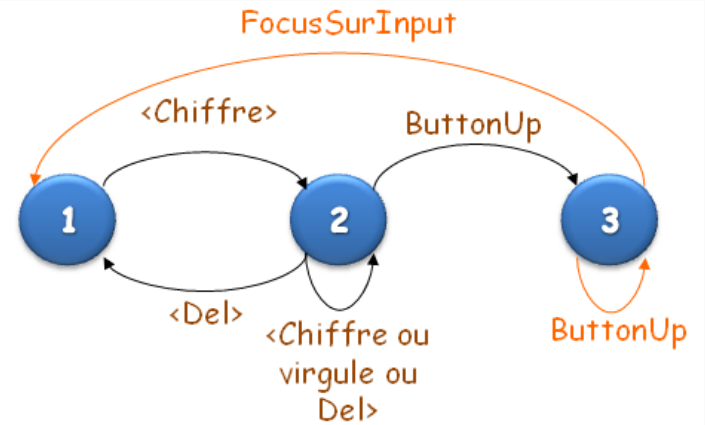
- l'explosion combinatoire

## ↪ Les automates

Exemple :

- États :
  1. Champs "Input" et "Output" vides
  2. "Input" contient au moins un chiffre ou un nombre
  3. "Output" contient la valeur convertie
- Transitions :
  - `<Chiffre>` : chiffre au clavier
  - `<Del>` : 'del' au clavier
  - `<ButtonUp>` : clic sur un bouton
- Actions
  - Action de conversion
- Contrôle de l'interaction
  - Activer/Désactiver un bouton
- Contrôle complet du dialogue
  - Afficher/Supprimer le résultat

*Remarque : ne pas oublier le pattern État . . .*




```
État T1 : 'Pas de nombre à convertir'
<Chiffre> : ActiverBoutons -->T2

État T2 : 'Nombre à convertir'
<Chiffre ou virgule>: Rien --> T2
<Del> : DésactiverBoutons --> T1
ButtonUp : Conversion
DésactiverUnBouton --> T3

État T3 : 'Nombre converti'
ButtonUp : Conversion
DésactiverUnBouton --> T3
FocusSurInput : EffacerOutput
DésactiverBoutons --> T1
```