CONCEPTION ORIENTÉE OBJET

UML = langage pour faciliter les échanges entre les acteurs d'un procédé.

Phases de développement :

- 1| Besoins
- 2| Analyse
- 3| Design
- Implementation
- Test
- Déploiement
- Maintenance

} UML utilisé ici

→ **Besoins** = ce qui est necessaire à la création du Σ

Répond aux questions :
- qu'est ce que c'est ? → Besoin non fonctionnel
- qu'est ce que ça fait ? → Besoin fonctionnel

→ **Analyse**
Comprendre le contexte du Σ → identités utiles
→ leurs propriétés et liens.

→ **Design** :
Comment resoudre le pbs
- Structurel : sépare le Σ en | composants et sous Σ logiques
sous Σ Ψ (ordinateurs et réseaux)
- Comportemental : collaboration des composants pour assurer les services.

→ **Implementation** → coder les spécifications Σ
→ **Tester** : verifier si objectifs atteints
→ **Déploiement** : livraison aux utilisateurs
→ **Maintenance** : correction des bugs et extensions.

UML pendant les Ψ de design et Analyse

Notation graphique → crée modèles abstraits des Σ

But: pour le | modéliste → comprendre le pbs
designer → explorer solutions possibles
développeur → mettre en œuvre solutions.

Diagrammes : ∃ 13 - comportementaux et structurels -

- Comportementaux
  - * use case : services que les acteurs peuvent demander au Σ
  - * Machine à état : cycle de vie d'un objet.
  - * activité : modélisat° du contrôle et flux de données simultanés (≡ RdP)

sous groupe :
- Diagrammes d'interaction
  - * diag. de séquence : échanges de msg entre des groupes d'objets
  - * diag global d'interaction : montre les ≠ scenarios d'interaction pour une ℑm collaborat°.

* Collaborat° / Communication : = diag de séquence
* Timing : états d'un elemt UML en f° du tps

- Structure
  * Classe → entités et leurs relations
  * diag struct. composite : montre comment une classe est faite ⊕ comment les parties d'une classe sont connectées entre elles.
  * Composants : montre la struct du Σ comme caisses noires et leurs intéractions.
  * Déploiement : archi durée Σ ...
  * Objects :
  * Package : organiser modèles des elemts ⊕ dépendances

Diagramme des Cas d'utilisat° / Use case

→ caract internes et intéractions entre d'un nouveau Σ.

1 use case donne 1 ou plusieurs scenarios → Comment le Σ devrait intéragir avec l'utilisateur final ou un autre Σ pour accomplir un certain but ?

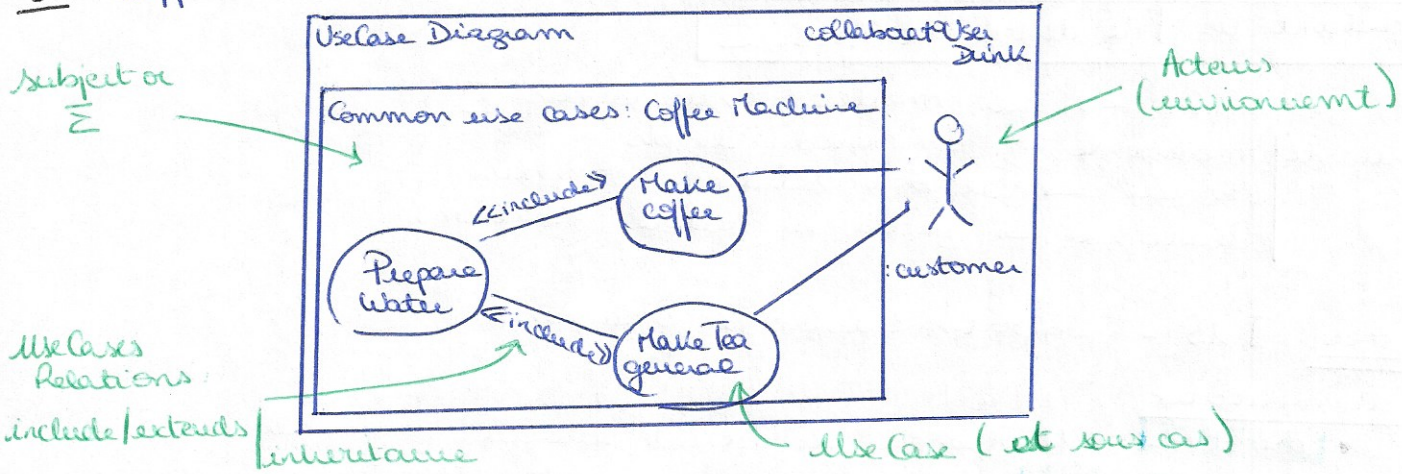
Basé sur Spec. des besoins du Σ (SRS) : identifier → sujets
→ verbes
→ détails / commentaires.

+ Entités
  = sujets des besoins ; certains seront acteurs ou comp. internes du Σ.

+ Cas d'utilisation
  ≡ Id verbes du SRS → use case
  ↳ restrictions / détails
  ↳ autres elemts du modèle.

+ Détails nécessaires pr les classes / attributs / messages

ex: Coffee Machine



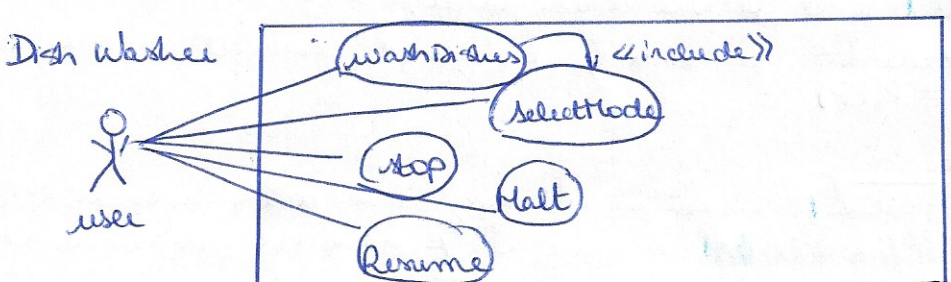


Diagramme de Séquence — SD

Représente les échanges entre les composants requis pour implémenter un cas d'utilisation spécifique.

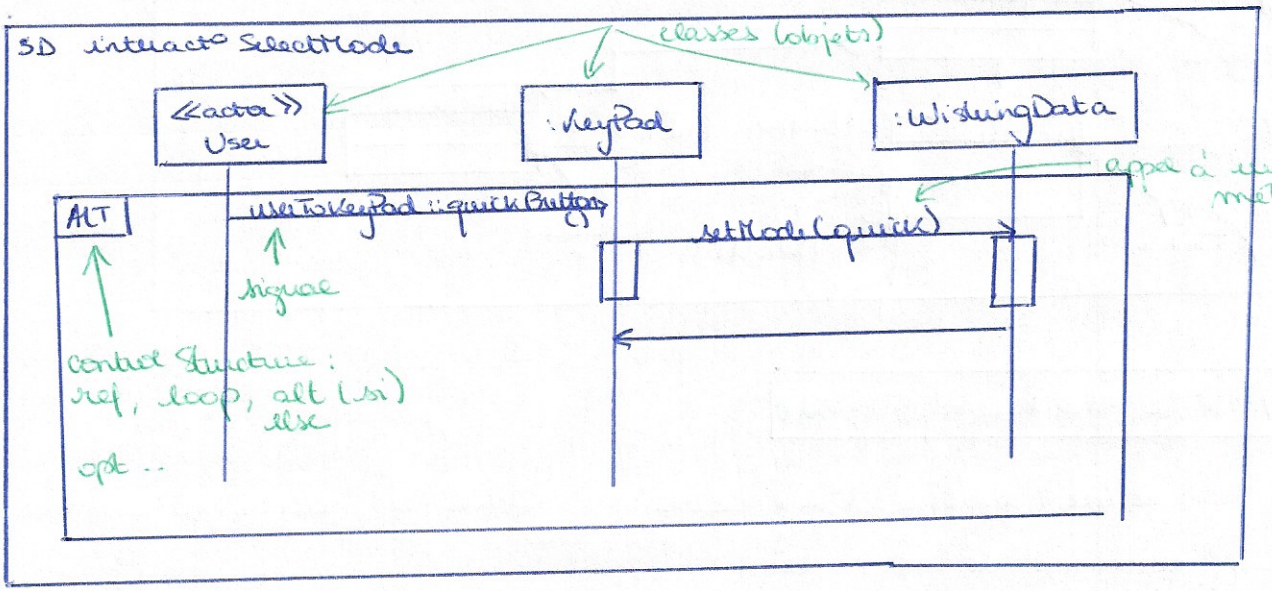
Décrit l'ordre ds le tps des échanges entre les composants Σ.

Classe modélisées comme life lines (verticales)

→ **Active class** : contient compatents autonomes.
→ **Passive Class** : contient attributs et méthodes utilisées par objets actifs.

Messages rep par flèches horizontale entre 2 lifelines.
↳ peuvent ê méthodes particulières → constructeurs / destructeurs.

ex:

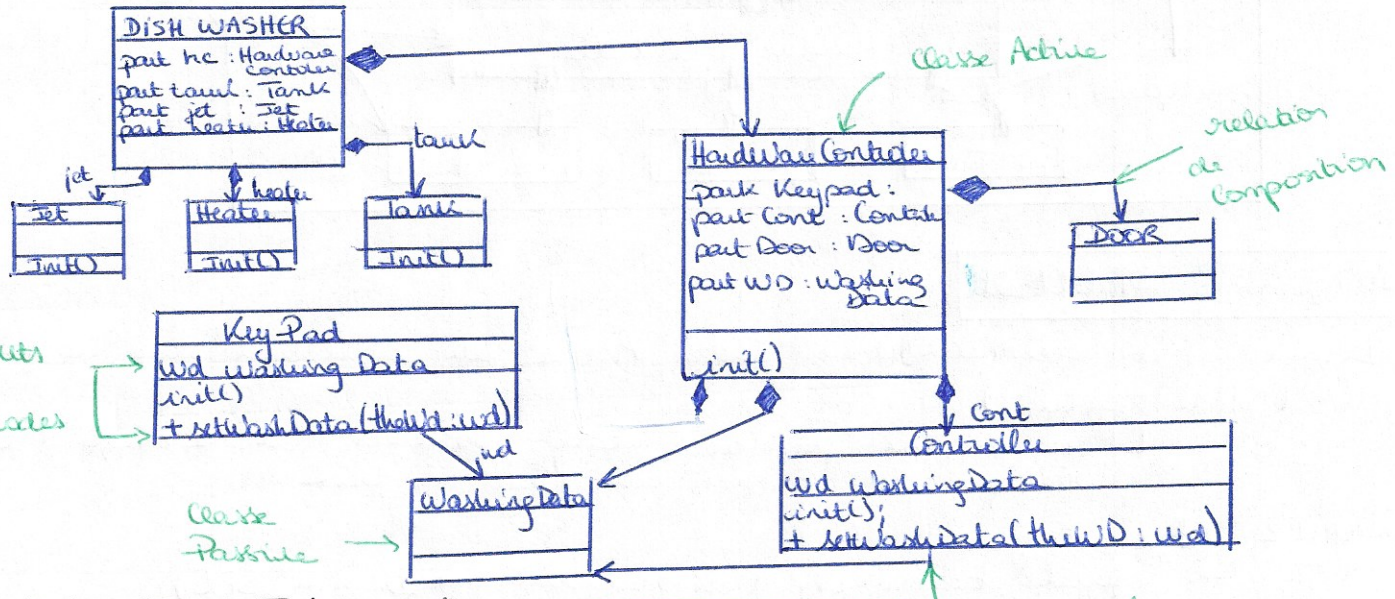


Diagrammes de classe

Représente les ≠ composants d'un Σ ainsi que leurs relations

→ décrit la vue statique des classes.
→ attributs et méthodes peuvent ê décrits ⟹ visibilité peut ê marquée:
↳ nom ↳ (type) — ↳ nom ↳ (param)(type de retour)
\+ → public
\# → protected
\- → private

ex:



une fois les classes Id on peut
- décrire leur interface externe → Component Diag
- " struct interne → Diag Composite
- " comportemt interne → Diag StateChart

Diagramme de Composants · Component Diagram

Décrit comment les classes actives vont communiquer entre elles en utilisant :

- interfaces requises (OUT)
- interfaces implémentées (IN)
- par le biais de ports.

ports et signaux envoyés et reçus sur ces ports. Signaux regroupés en interfaces

- Qd un composant peut traiter les signaux : le comp implémente ou "provide" l'interface
- Si un comp produit le signal : il a besoin de l'interface

ex:

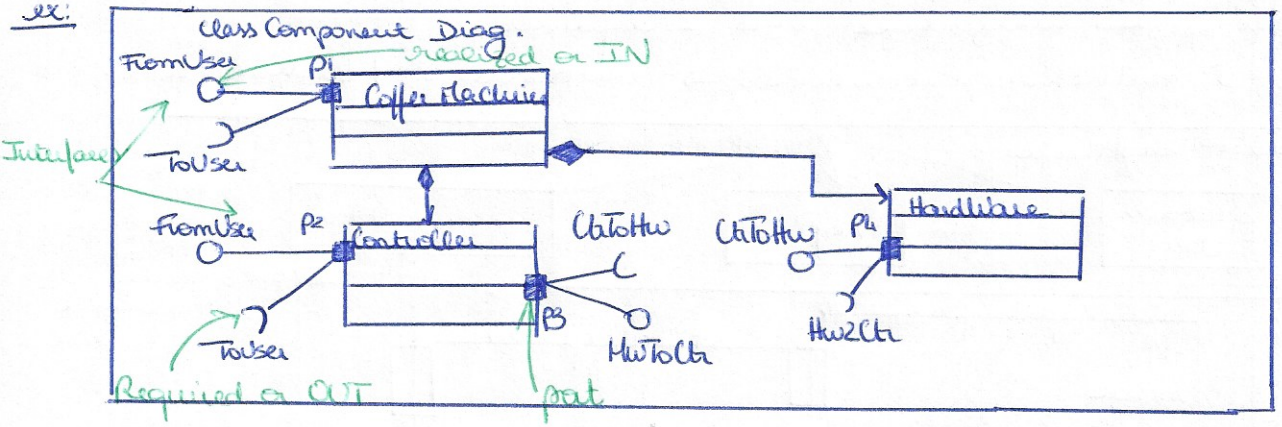


Diagramme de structure Composite

décrit la struct interne d'une classe, incluant les parties et connexions entre elles.

Parties → relations de Composition.

Container class → a des ports d'entrée / Sortie de signaux pr communiquer avec son environement.

lignes connectant ports internes et externes = connecteurs (uni / bidirectionnels)

ex:

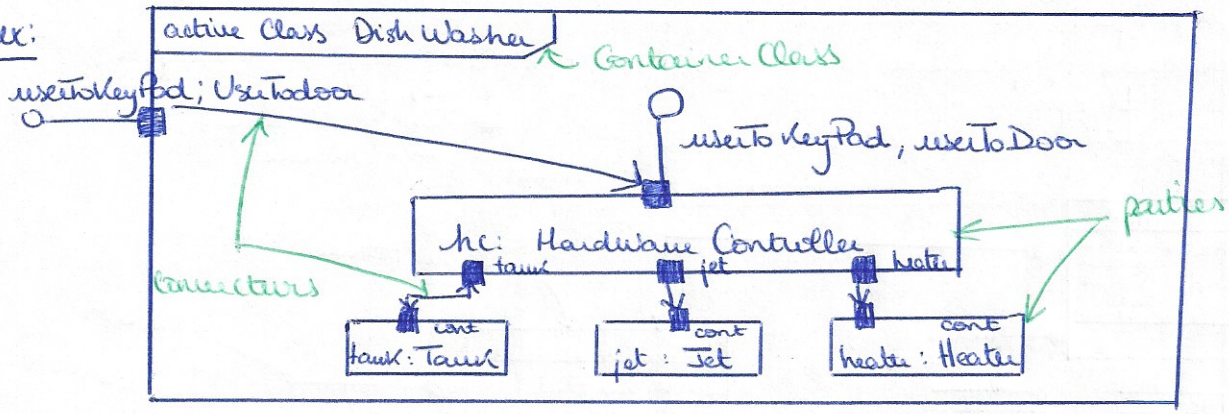


Diagramme StateChart

Décrire le comportement d'une Classe Active en utilisant une Machine à état (State Machine)

↳ a un, ou plusieurs états possibles et 1 chgt d'état est provoqué par la réception d'un signal

En UML2, ∃ 2 notations :

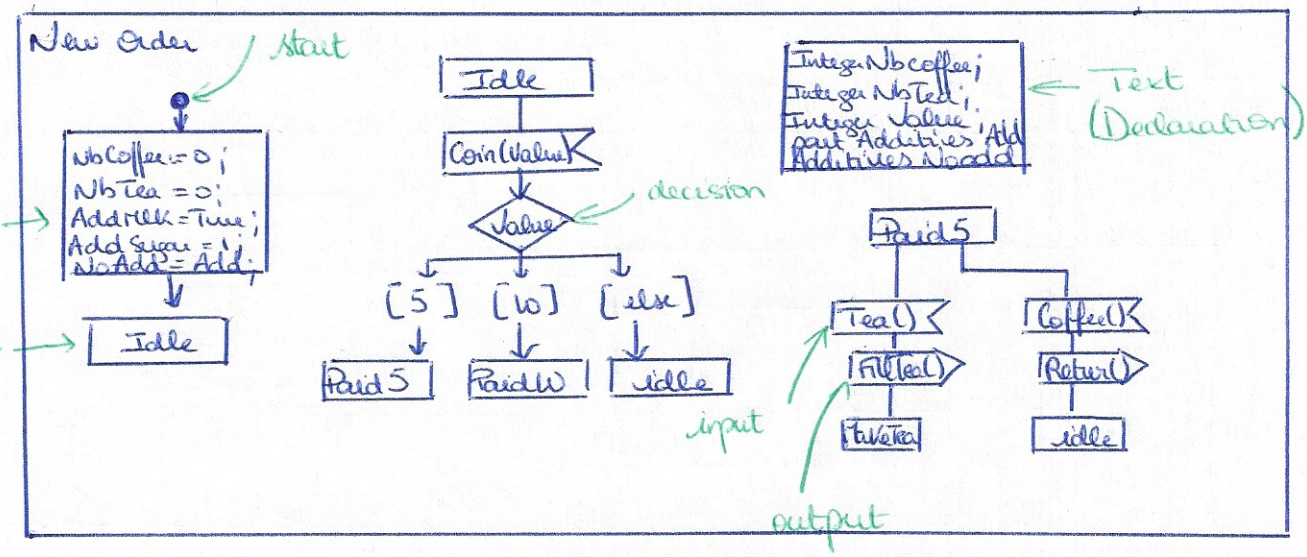
→ orienté transition (ok pour outillage Design SDL)

→ orienté état : large design.

Une fois le stateChart généré, les outils UML => générer code pour mimer
=> implementer Σ

Tests : manuels ou auto pour vérifier si implementation comportementale correspond au comportement spécifié ds valeurs des champs.

ex:



When UML is The Center of the Software Engineering Process

Generating implementation: MDE

- Model Driven Engineering (MDE) → usage systematique de modèles tout au long du cycle de vie.

Peut être appliqué → software
→ Σ
→ data engineering

→ **MDD** Model Driven Dvpt
→ **MDA** Model Driven Architecture

→ **MDD**

catégories d'approches de dvpt basé sur les modèles software comme première étape

* Certains modèles peuvent être construits jusqu'à un certain niveau puis code fait à part à la main

OU

* models complets faits incluant exécutables.
* Code peut être généré à partir des modèles.

→ **MDA**

contient un ens de lignes à suivre pr structurer les spécifications. définit le Σ en utilisant PIM : Platform independent Model
puis avec PDM platform definition Model => PIM convertit en PSMs (platform specific Models) exécutables sur ordi.

PSM peut utiliser # langages (Java, C #, Python etc).
→ CIM / PIM / PSM.