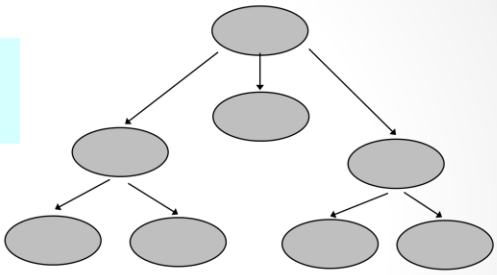
UML 2

Les concepts objet.
Modéliser avec UML

Carina ROELS

I. Approche fonctionnelle vs. approche objet

La méthode de conception fonctionnelle descendante (décomposition fonctionnelle)



\+ organisé, réfléchi, logique

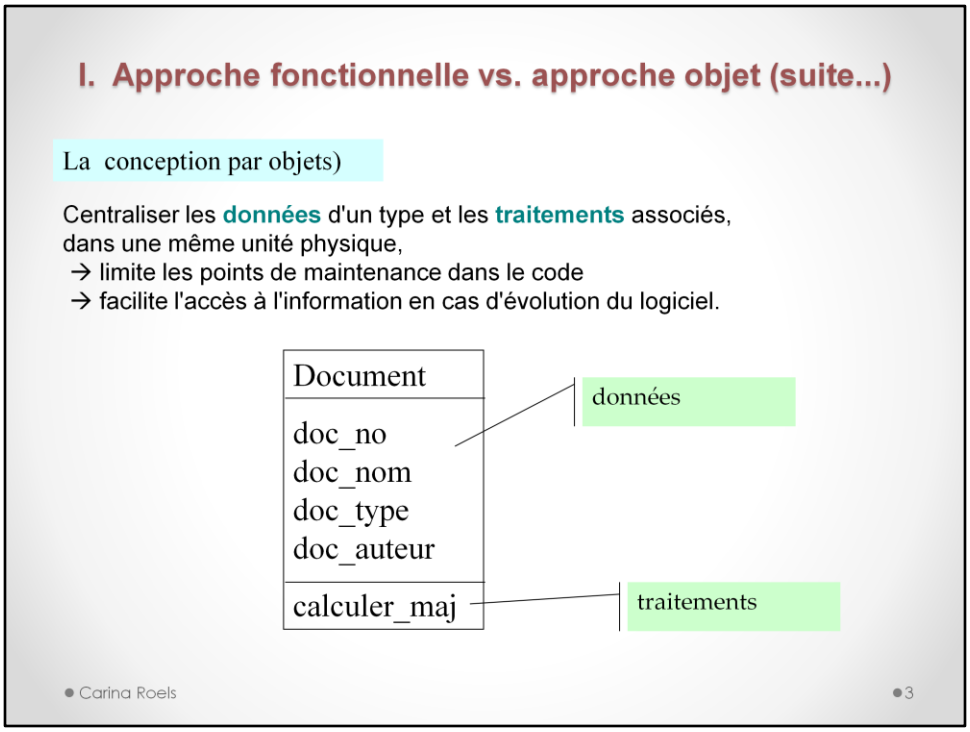
\-
- pas de prise en compte de la nature évolutive des logiciels
- l'utilisation des fonctions pour construire le système ➔ données négligées.
- méthode descendante : ne favorise pas la réutilisabilité

Carina Roels 2

- **Le logiciel est composé d'une hiérarchie de fonctions**, qui ensemble, fournissent les services désirés, ainsi que de données qui représentent les éléments manipulés.
  La démarche est logique, cohérent et intuitif.

- **L 'avantage de l'approche fonctionnelle : la factorisation des comportements**
  Pour réaliser une fonction du logiciel, on peut utiliser un ensemble d'autres fonctions, déjà disponibles, à condition qu'on rende ces dernières suffisamment génériques.

- **Le revers de la médaille : maintenance complexe en cas d'évolution**
  Ce découpage n'a pas que des avantages. Les fonctions sont devenues interdépendantes : une simple mise à jour du logiciel à un point donné, peut impacter en cascade d'autres fonctions. On peut minorer cet impact, pour peu qu'on utilise des fonctions plus génériques et des structures de données ouvertes. Mais respecter ces contraintes rend l'écriture du logiciel et sa maintenance plus complexe.



Une structure de données (ici : document) est manipulée par des fonctions (ici : la fonction calculer_maj).
L ’ensemble des propriétés cohérentes et les traitements associés s ’appelle un **objet**.

Un objet possède :

•une identité (un nom)

•des attributs qui caractérisent l ’état de l ’objet

•un ensemble d ’opérations (méthodes) qui définissent le comportement

Un **objet** est une instance de classe (une occurrence d ’un type abstrait)

Une **classe** est un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à des objets.

II. Les principes objet

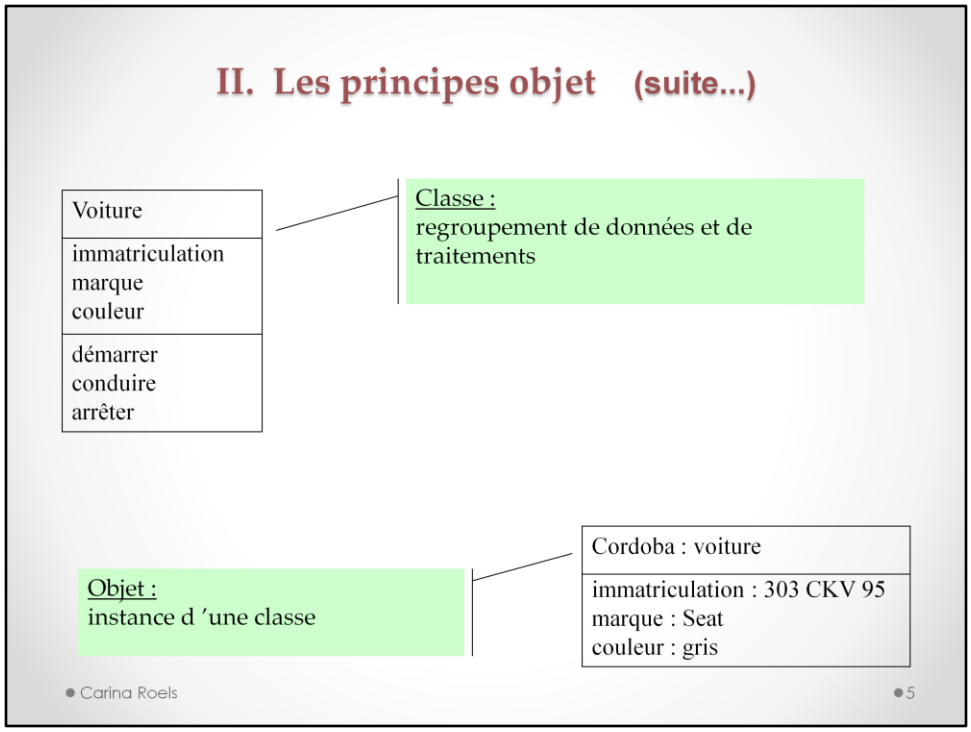
OBJET

n'importe quoi d'identifiable

- entité physique (ex. une voiture)
- entité intangible (ex. une maladie, un service…)

LA CONCEPTION PAR LES OBJETS

Méthode qui conduit à des architectures logicielles **fondées sur les objets** que tout système ou sous-système manipule, plutôt que sur " la " fonction qu'il est censé réaliser.

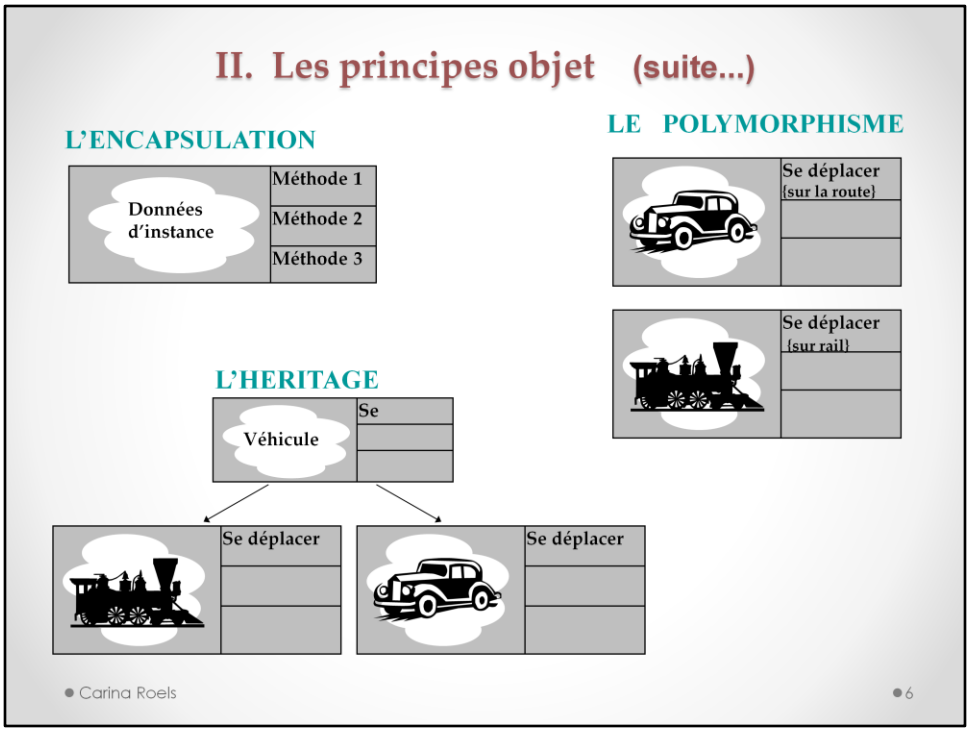


CLASSE : terme générique, pour décrire des ensembles de structures de données.

OBJET : instance d'une classe.

Les objets ont 3 propriétés qui les rendent particulièrement utiles (les piliers pour créer, assembler, réutiliser les objets)

- l'encapsulation
- l'héritage (généralisation / spécialisation)
- le polymorphisme



❑ **L'encapsulation**

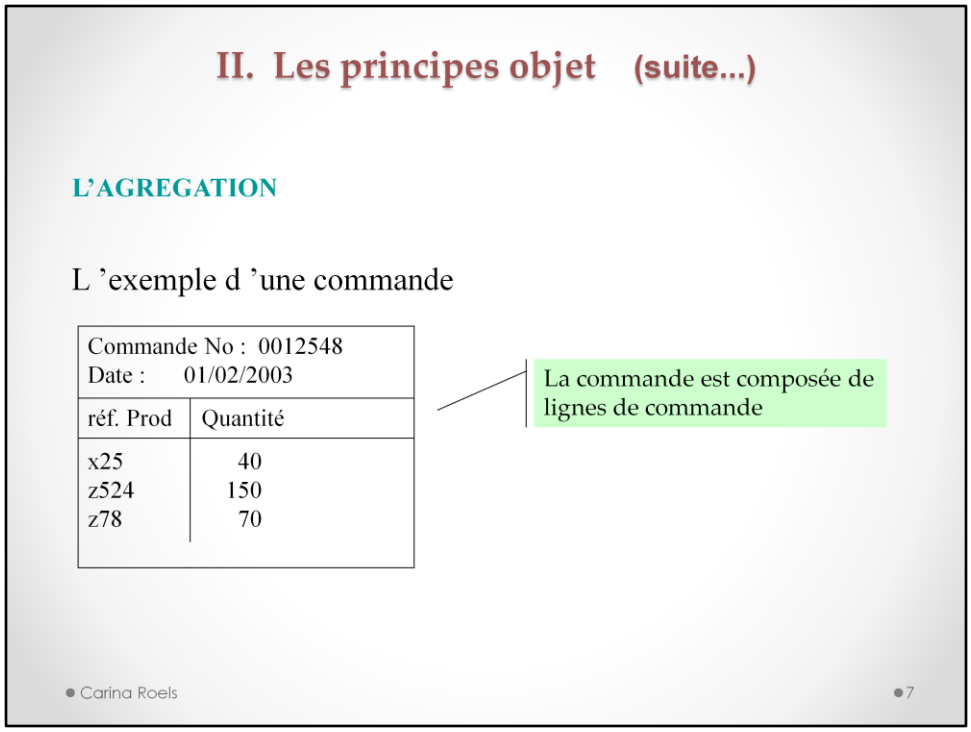
- Consiste à masquer les détails d'implémentation d'un objet, en définissant une interface.
- L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.
- L'encapsulation facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface.
- L'encapsulation garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'accesseurs).

❑ **☹❼♒éritage**

- mécanisme de transmission des propriétés d'une classe (ses attributs et méthodes) vers une sous-classe.
- Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines. Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes. La spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple.
- L'héritage évite la duplication et encourage la réutilisation.

❑ **Le polymorphisme** représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.

- Le polymorphisme augmente la généricité du code.

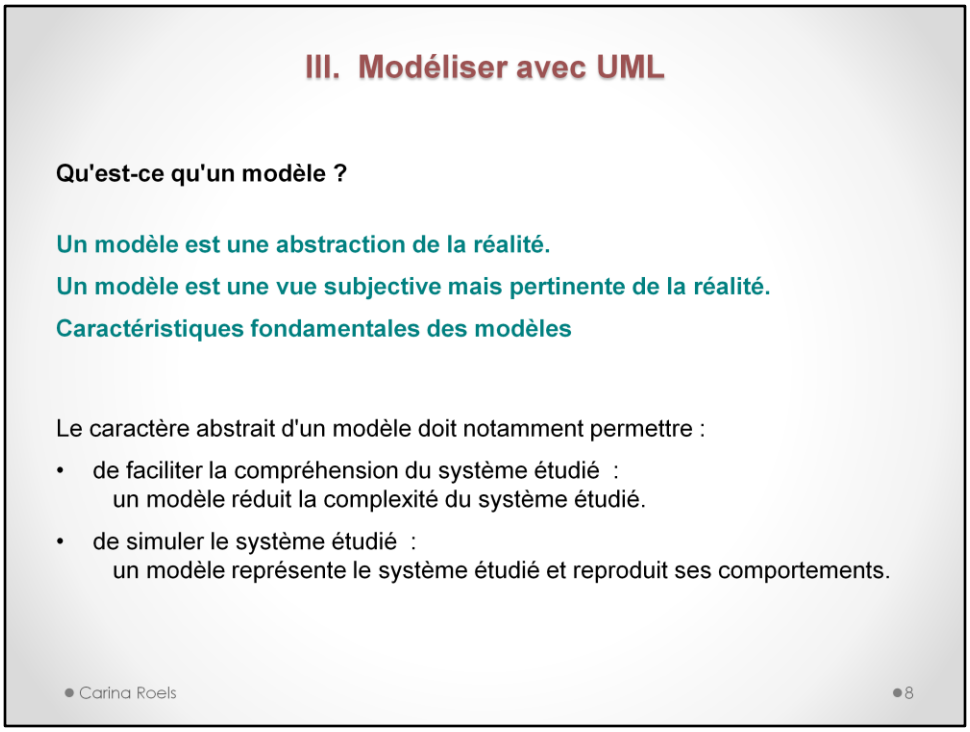


L'agrégation

- Il s'agit d'une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe.

  Une relation d'agrégation permet donc de définir des objets composés d'autres objets.

- L'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes.



Un modèle est une abstraction de la réalité
L'abstraction est un des piliers de l'approche objet.

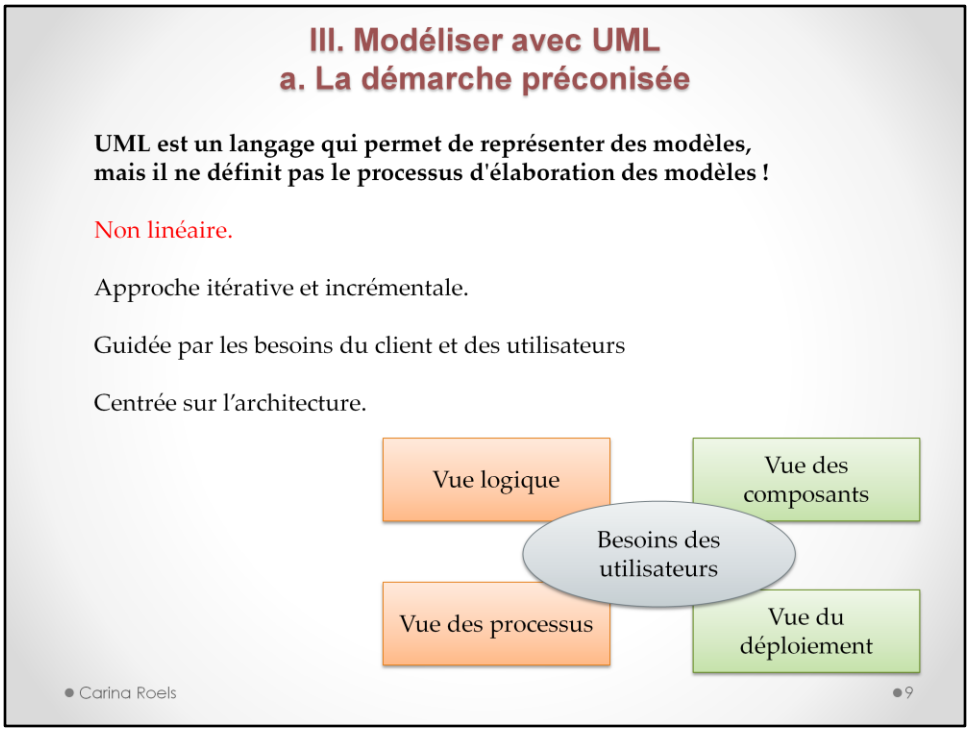
•Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise.

•L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.

Un modèle est une vue subjective mais pertinente de la réalité

•Un modèle définit une frontière entre la réalité et la perspective de l'observateur. Ce n'est pas "la réalité", mais une vue très subjective de la réalité.

•Bien qu'un modèle ne représente pas une réalité absolue, un modèle reflète des aspects importants de la réalité, il en donne donc une vue juste et pertinente.



Une démarche itérative et incrémentale
La modélisation d'un système complexe se fait en plusieurs fois, en affinant la représentation (compréhension) par étapes.
Le but est de mieux maîtriser la part d'inconnu et d'incertitudes qui caractérisent les systèmes complexes.

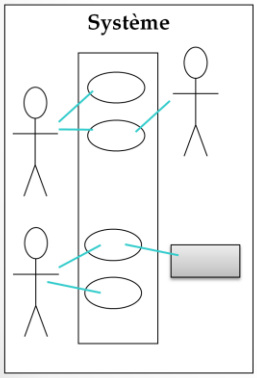
Une démarche guidée par les besoins du client et des utilisateurs
Le périmètre du système à modéliser est défini par les besoins des utilisateurs.
Les besoins des utilisateurs servent de fil rouge, tout au long du cycle de développement (itératif et incrémental) :

- Lors de la phase d'analyse → clarification, affinage et validation des besoins des utilisateurs.
- Lors de la phase de conception et de réalisation → vérification de la prise en compte des besoins des utilisateurs.
- Lors de la phase de test → vérification de la satisfaction des besoins.

Une démarche centrée sur l'architecture
Une architecture adaptée est la pierre angulaire d'un développement.
Elle décrit des choix stratégiques qui déterminent en grande partie les qualités du logiciel (adaptabilité, performances, fiabilité...).

III. Modéliser avec UML
b. Les besoins des utilisateurs

Description de ce qui sera vu par les acteurs du système. Cela correspond aux besoins attendus par chaque acteur (QUOI / QUI).



Description de chaque cas d'utilisation

Nom _____
Présupposés _____
Pré-conditions _____

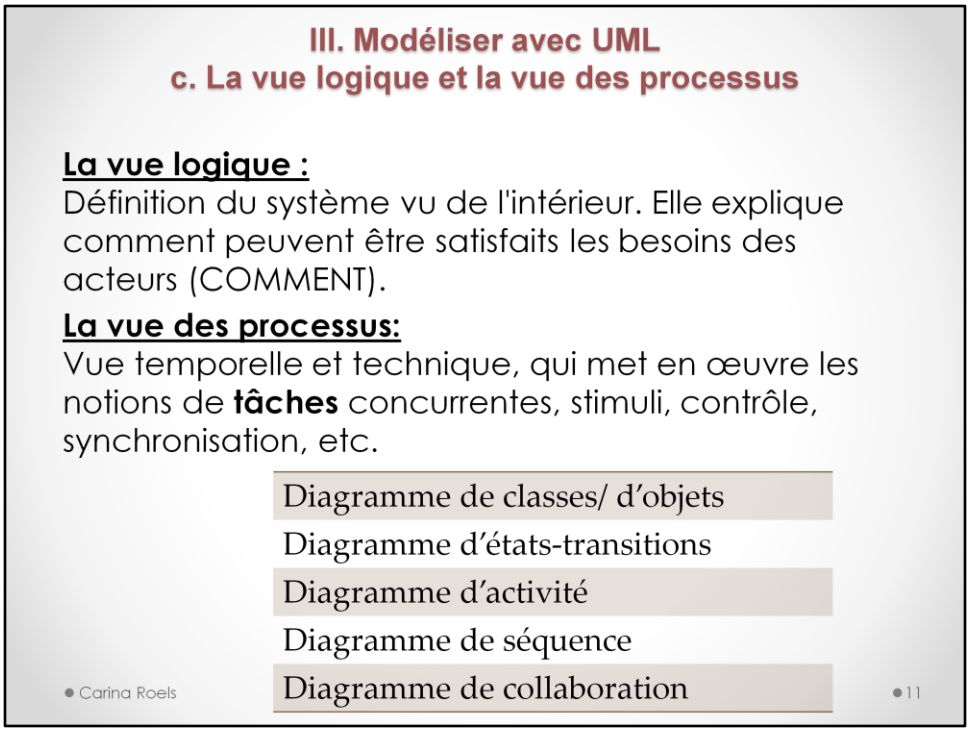
Dialogue
_____
_____
_____

Post-conditions _____
Exceptions _____
Améliorations futures
_____
Problèmes non résolus
_____

Diagramme des cas d'utilisation (Use Case Diagram) :
Identification des interactions entre le **système** et les **acteurs** (intervenants extérieurs au système) → les **fonctionnalités** que doit fournir le système.

Le diagramme de cas d'utilisation est la base pour la compréhension des besoins utilisateur.

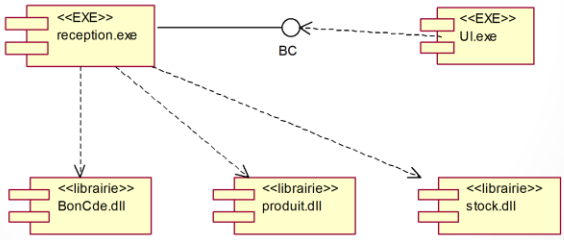
Il servira de guide pour la réalisation et la vérification des autres vues.
Il permet également d'identifier les interfaces qui mériteraient une attention particulière.



- **Le diagramme de classes** permet de décrire les éléments de modélisation statique (la structure).
- **Le diagramme d'objets** permet de décrire les instances de classes dans un état particulier, ainsi que les relations entre ces instances.
- **Le diagramme d'états-transition** permet de décrire l'évolution de l'état d'un objet ou d'un composant.
- **Le diagramme d'activité** permet de représenter graphiquement le déroulement d'un cas d'utilisation.
- **Un diagramme de séquence** est une représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs. Il est utilisé pour compléter la description d'un cas d'utilisation.
- **Le diagramme de collaboration** décrit les interactions entre objets (instances de classes et acteurs).

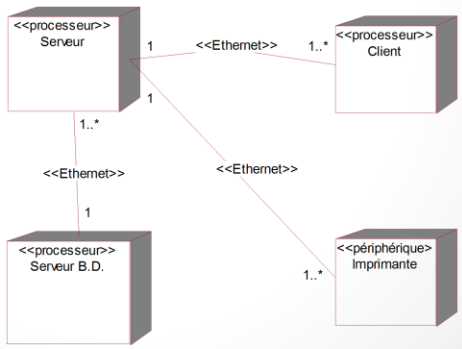
III. Modéliser avec UML
d. La vue des composants

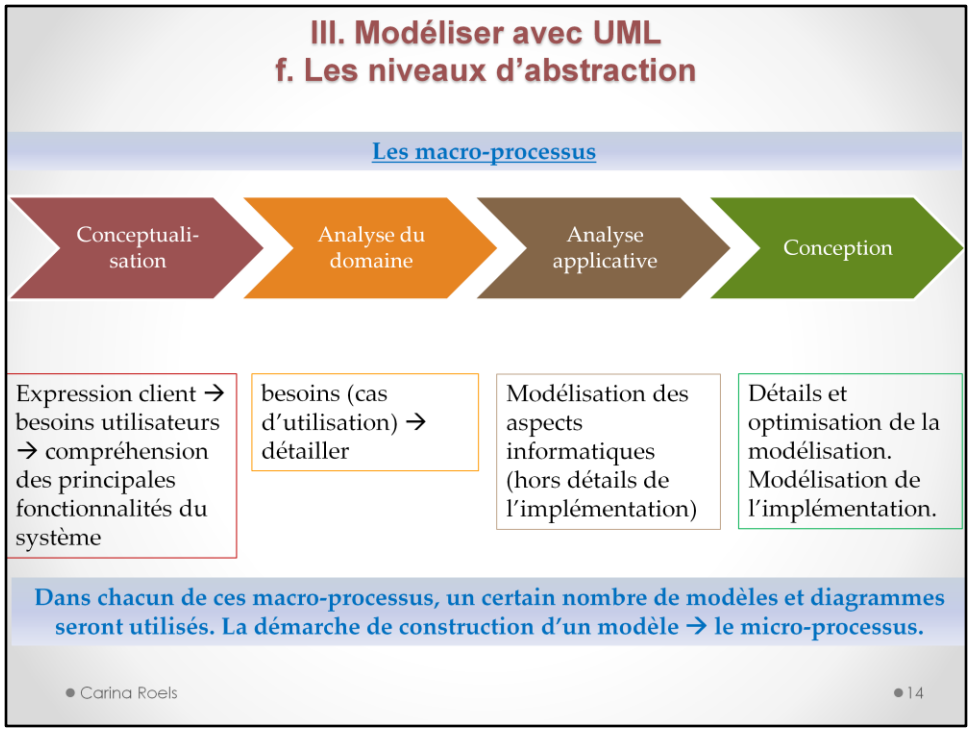
La **vue des composants** décrit les éléments qui vont réaliser les classes (fichiers source, librairies, bases de données, exécutables, etc.) et leur dépendances.



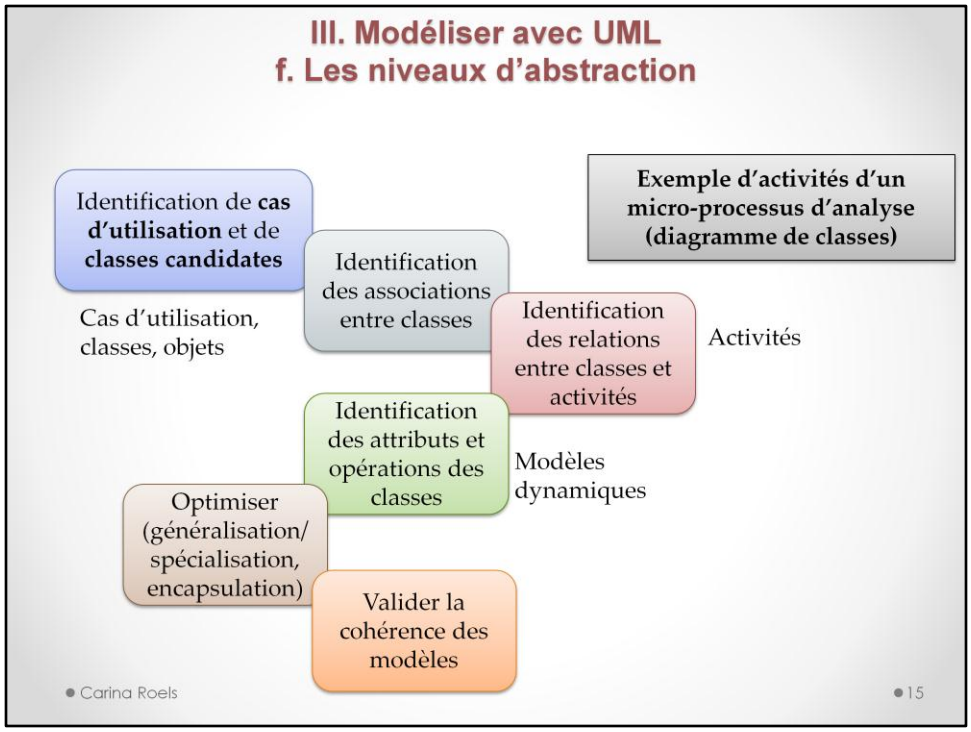
III. Modéliser avec UML
e. La vue de déploiement

La **vue de déploiement** décrit la position géographique et l'architecture physique de chaque élément du système (OÙ).





- **La conceptualisation :** Le dossier d'expression des besoins client est le point de départ. Il s'agit de capturer les besoins principaux des utilisateurs, sans pour autant chercher l'exhaustivité. Le but de la conceptualisation est :
  - de définir le contour du système à modéliser (de spécifier le "quoi"),
  - d'identifier les fonctionnalités principales du système, afin d'en fournir une meilleure compréhension,
  - de fournir une base à la planification du projet.
- **L'analyse du domaine**
  On démarre ce niveau d'analyse avec le modèle des besoins clients (les cas d'utilisation). Il s'agit de modéliser les éléments et mécanismes principaux du système. On identifie les éléments du domaine, ainsi que les relations et interactions entre ces éléments. A ce stade, on organise aussi (selon des critères purement logiques), les éléments du domaine en ***"catégories"*** pour répartir les tâches dans les équipes, pour regrouper ce qui peut être générique, etc...
- **L'analyse applicative** permet de modéliser les aspects informatiques du système, sans pour autant rentrer dans les détails d'implémentation.
  On définit les interfaces des éléments de modélisation et les relations entre ces éléments.
  Les éléments de modélisation utilisés peuvent être propres à une version du système.
- **La conception** permet de modéliser tous les rouages d'implémentation et on détaille tous les éléments de modélisation issus des niveaux supérieurs.
  Les modèles sont optimisés, car destinés à être implémentés.



❑ identifiez les classes (d'objets) :

- recherchez les classes candidates (différentes suivant le niveau d'abstraction) à l'aide de **diagrammes d'objets** (ébauches),
- filtrez les classes redondantes, trop spécifiques ou vagues, qui ne représentent qu'une opération ou un attribut et documentez les caractéristiques des classes retenues (persistances, nombre maximum d'instances, etc.)

❑ identifiez les **associations entre classes** / interactions entre objets (instances) :

- recherchez les connexions sémantiques et les relations d'utilisation,
- **Documentez les relations** (nom, cardinalités, contraintes, rôles des classes...),
- définissez la dynamique des relations entre objets (les **interactions** entre instances de classes et les **activités** associées).

❑ identifiez les attributs et les opérations des classes :

- recherchez les attributs dans les **modèles dynamiques** (recherchez les données qui caractérisent les **états** des objets),
- filtrez les attributs complexes (faites-en des objets) et au niveau spécification, ne représentez pas les valeurs internes propres aux mécanismes d'implémentation,
- recherchez les opérations parmi les **activités** et actions des objets (ne pas rentrer dans le détail au niveau spécification).

❑ optimisez les modèles :

- choisissez vos critères d'optimisation (généricité, évolutivité, précision, lisibilité, simplicité...),
- Utilisez la **généralisation et la spécialisation** (classification),
- documentez et détaillez vos modèles, **encapsulez**.

❑ validez les modèles : vérification de la cohérence, relecture…