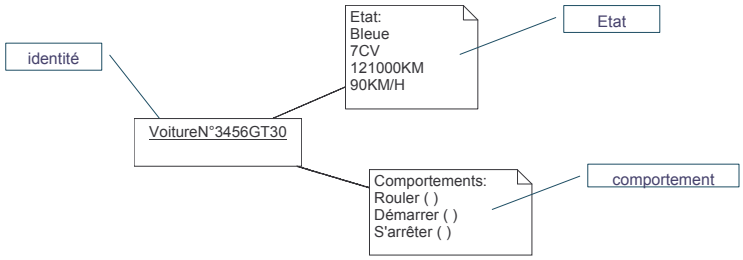
Approche objet

Cours de SI
Bordeaux I 2004

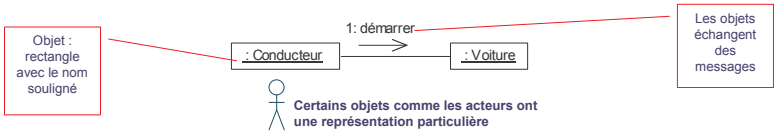
U M L — Qu'est-ce qu'un objet ?

Objet = identité + état + comportement

Dans la phase d'analyse objet (OOA) , avant de parler de classe, on peut représenter les objets



UML propose des formalismes particuliers pour représenter les objets notamment dans les diagrammes de collaboration et les diagrammes objet



Cours de SI
Bordeaux I

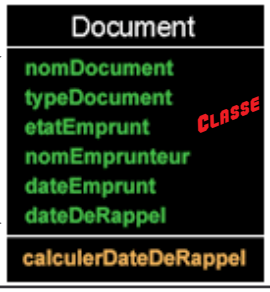
U M L — Qu'est-ce qu'une classe ?

```
struct Document
{
  char nom_doc[50];
  Type_doc type;
  Bool est_emprunte;
  char emprunteur[50];
  struct tm date_emprunt;
  int delai_avant_rappel;
} DOC[MAX_DOCS];
```

```
int calculer_delai_rappel(Type_doc type)
{
  switch(type)
  {
    case LIVRE:
      return 20;
    case CASSETTE_VIDEO:
      return 7;
    case CD_ROM:
      return 5;
    /* autres "case" bienvenus ici ! */
  }
}
```

Une classe est une abstraction d'objet qui regroupe les traits communs à un ensemble d'objet.

La classe est donc une description de la structure de tout objet appartenant à la classe



U M L — Qu'est-ce que le type d'un objet ?

Pour manipuler un objet il faut et il suffit de connaître les messages qu'il accepte

Type = ensemble des messages acceptés par l'objet

Un type spécifie abstraitement des opérations applicables à un domaine d'objets, indépendamment de leur implémentation physique.

Un type fournit donc une spécification du comportement des opérations (il n'a donc pas nécessairement de méthodes, ni attributs, ni associations).

A noter qu'il n'est pas facile de faire la distinction conceptuelle entre le type et la classe.

Le type est à la spécification d'un élément
La classe est la réalisation de ce type

c'est-à-dire qu'elle décrit les méthodes qui permettent de réaliser les opérations définies par le type.

Pour manipuler un objet il faut connaître son type c'est-à dire :

-Les messages acceptés = souvent les *paramètres* de la méthode = signature

- La réaction aux messages (traitement/comportement) = code de la méthode

U M L Qu'est-ce que le type d'un objet ?

Un exemple usuel est celui des types primitifs d'un langage :

Le type *Integer* est défini par les opérations applicables sur le domaine des nombres entiers.

Par exemple le résultat de la division de 2 nombres entiers doit être un entier.

On ne se préoccupe nullement de savoir comment réaliser cette opération. Mais la règle doit être respectée par tout objet ayant ce type.

La classe d'implémentation de ce type, doit, elle, comporter une méthode division() qui donne un résultat conforme à la règle énoncée par le type.

Le type est donc bien une spécification et la classe la réalisation de ce type

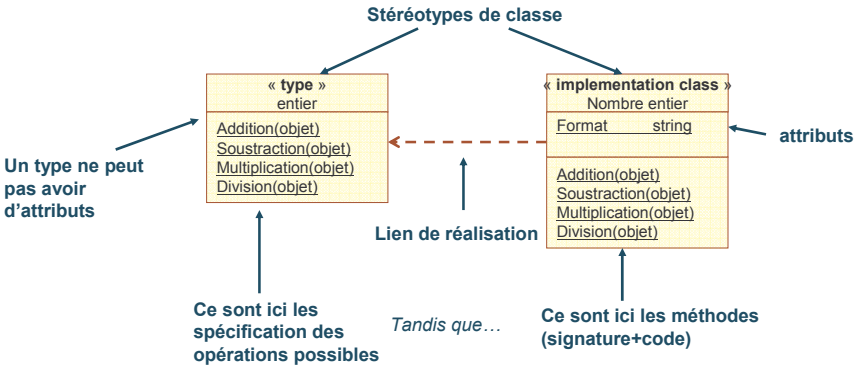
Niveau conceptuel	Objet (type qui décrit le comportement, implémentation qui décrit la structure)
Niveau Logique	Classe (attributs méthodes (signature, code))

U M L Qu'est-ce que le type d'un objet ?

Avec UML on représente un type par une classe de stéréotype « type »

Exemple :

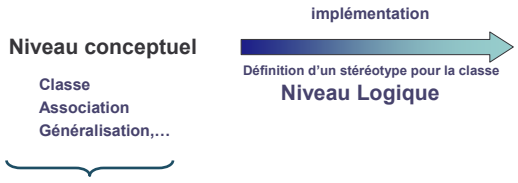
Reprenons notre exemple du type entier, et représentons un nombre entier.



U M L — Qu'est-ce qu'un stéréotype ?

Le stéréotype représente la manière standard d'implémentation d'un objet.

C'est un élément de modélisation qui étend la sémantique du métamodèle d'UML c'est-à-dire qui étend le dictionnaire de base du langage (objet, classe, …).



Supposées être indépendantes du langage de prog et de l'implémentation. Mais ce n'est que rarement le cas

=>

UML propose les stéréotypes prédéfinis suivants :
<<Implementation Class>>
<<Type>>
<<Interface>>
<<Utility>>
<<metaclass>>

U M L — Encapsulation

- L' *interface* (ce qu'on voit d'un objet)

Etat + comportement visibles

Objet = identité + état + comportement

Etat + comportement cachés

-L'implémentation (la façon dont on va pratiquement représenter les choses)

Le fait de ne pas voir l'implémentation (le code), c'est l'encapsulation.

Les applications clientes n'ont pas à se soucier du fonctionnement interne.

Cela a une conséquence, si je modifie le code mais pas l'interface, l'application cliente n'a pas à être modifiée.

Encapsulation

Exemple en VB.net :

Prenons un objet d'une classe ListBox:

L'interface c'est ListBox.Visible ListBox.AddItem...Je la vois , je peux l'utiliser.

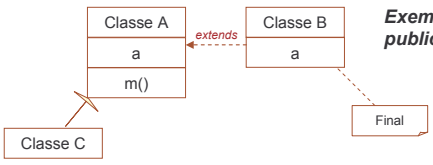
L'implémentation, je ne la vois pas, c'est le code qui gère la ListBox, la définition des éléments;

c'est une 'boite noire', je ne sais pas ce qui s'y passe, je n'y est pas accès, et c'est tant mieux!

Encapsulation

Il y a deux manières de représenter l'encapsulation avec UML

Ne pas autoriser la possibilité de faire dériver une classe de la classe B



Exemple en Java:
public final class B extends A { ...
}

Autoriser ou non l'accès à un attribut ou à une méthode à partir d'une autre classe

- **UML définit 3 niveaux de visibilité :**
  - *public* **(+) : qui rend l'élément visible à tous les clients de la classe**
  - *protected* **(#) : qui rend l'élément visible aux sous-classes de la classe**
  - *private* **(-) : qui rend l'élément visible à la classe seule**

U M L Représentation d'une interface

Une interface est une spécification des opérations qui sont visibles par l'environnement d'une classe.

Elle ne présente qu'une partie du comportement de la classe correspondante

Elle ne possède que des opérations (pas d'attributs, ni d'associations, ni d'états).

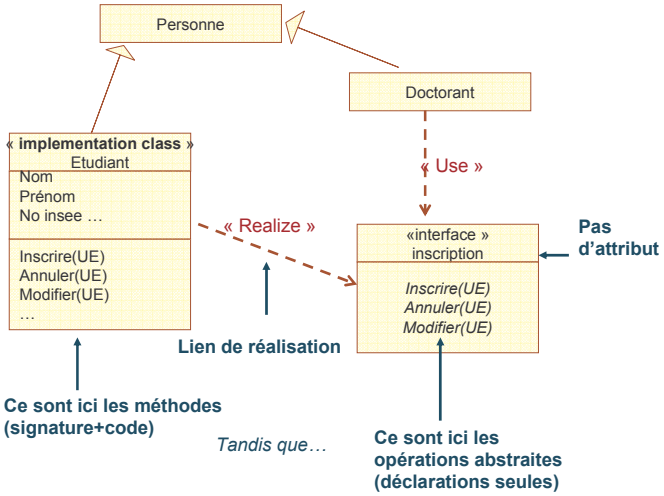
On représente les interfaces par le stéréotypes « interface » et le concept de classe abstraite. (puisqu'elles ne sont pas implémentées)

Une classe abstraite ne contient que des déclarations de méthodes mais pas leur code.

Rq : L'interface est donc une sorte de type particulier qui s'applique à une classe d'implémentation

U M L Représentation d'une interface

Exemple : Les étudiants et les doctorants doivent s'inscrire à des UE à l'université. Les opérations sont les mêmes pour tous. On peut donc définir une interface valable pour tous.



U M L Représentation d'une interface

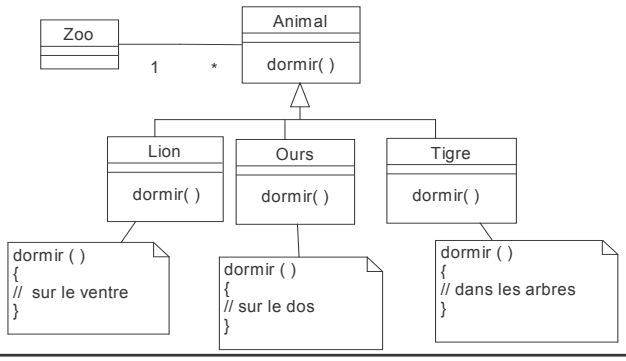
Autre symbolisme plus simple mais qui présente l'inconvénient de ne pas montrer le contenu de l'interface

U M L Polymorphisme

L'objectif est de créer des mécanismes suffisamment généraux pour être encore valides dans le futur, quand seront créées de nouvelles classes

Le terme polymorphisme désigne dans ce cas particulier le polymorphisme d'opération, c'est-à-dire la possibilité de déclencher des opérations différentes en réponse à un même message

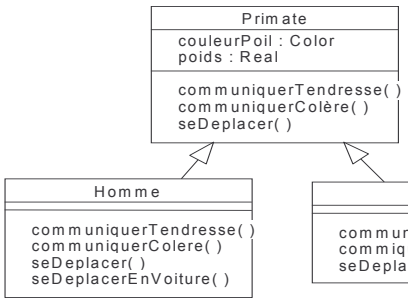
Chaque sous-classes hérite de la spécification des opérations de ses super-classes, mais a la possibilité de modifier localement le comportement de ces opérations



U M L Polymorphisme (overiding)

Le polymorphisme par héritage :

Exemple :




```
class Homme extends Primate {
        communiqueTendresse()
                {sourire();}
        communiquerColere ()
                {insulter ();}
        seDeplacer(){}
        seDeplacerEnVoiture(){}
        }
```

```
class Singe extends Primate {
        communiqueTendresse()
                {gemir();}
        communiquerColere ()
                {crier ();
                 sauter()}
        seDeplacer(){marcher a
        quatre pattes()}}
```

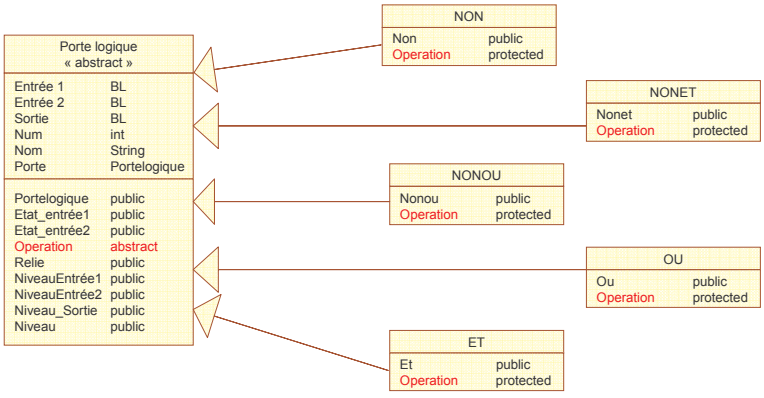
Les méthodes communiqueTendresse ( ) et communiquerColere( ) sont redéfinie (réécriture) pour redéfinir le comportement de l'homme et du singe (polymorphisme).

U M L Polymorphisme (overiding)

La classe abstraite

La réalisation du polymorphisme en UML passe par la création d' une classe abstraite et la création des classes qui étendent cette classe abstraite.

Une classe est dite abstraite à partir du moment où au moins une méthode est de type abstrait.



U M L Polymorphisme (overiding)

pour avoir un fonctionnement polymorphique une variable objet doit être:
- déclaré comme une variable de la classe abstraite
- puis instanciée comme un objet de la classe étendue.

Une méthode est dite abstraite si elle est déclarée dans la classe mais que son implémentation n'y est pas précisée.

Porte logique « abstract »	
Entrée 1	BL
Entrée 2	BL
Sortie	BL
Num	int
Nom	String
Porte	Portelogique
Portelogique	public
Etat_entrée1	public
Etat_entrée2	public
Operation	abstract
Relie	public
NiveauEntrée1	public
NiveauEntrée2	public
Niveau_Sortie	public
Niveau	public

```
abstract  PorteLogique {

/*une classe abstraite est une classe comportant des méthodes
qui ne sont pas définies dans la classe. elles sont définies dans la
classe qui en étend*/

   boolean Entree1, Entree2, sortie;
    int       num;
   String     nom;
   PorteLogique porte;

    Public void PorteLogique(String nom) {…}
    Public void EtatEntree2(boolean entree) {…}
    abstract  operation();
    Public void relie(PorteLogique porte,  n) {…}
    Public void niveauEntree1() {…}
    …
}
```

Contrairement aux autres méthodes elle n'est pas décrite ici mais le sera dans les classes dérivées

U M L Polymorphisme (overiding)

Porte logique « abstract »	
Entrée 1	BL
Entrée 2	BL
Sortie	BL
Num	int
Nom	String
Porte	Portelogique
Portelogique	public
Etat_entrée1	public
Etat_entrée2	public
Operation	abstract
Relie	public
NiveauEntrée1	public
NiveauEntrée2	public
Niveau_Sortie	public
Niveau	public

OU	
Ou	public
Operation	protected

ET	
Et	public
Operation	protected

```
Ou extends PorteLogique {

   Ou(String nom){
   super(nom);
   }

    operation() {
   boolean anciens;
   anciens = sortie;
   sortie = (Entree1 | | Entree2);
    (anciens = sortie) {
      (num  1) {
      porte.EtatEntree1(sortie);
      }
       (num  2) {
      porte.EtatEntree2(sortie);
      }
   }
   }
}
```

```
Et extends PorteLogique{

   Et(String nom){
   super(nom);
   }

    operation() {
   boolean anciens;
   anciens = sortie;
   sortie = (Entree1 & & Entree2);
    (anciens = sortie) {
      (num  1) {
      porte.EtatEntree1(sortie);
      }
       (num  2) {
      porte.EtatEntree2(sortie);
      }
   }
   }
}
```

Cours de SI
Bordeaux I

U M L — Autres Polymorphismes

Le polymorphisme de surcharge ou en anglais overloading.

Permet d'avoir des fonctions de même nom, avec des fonctionnalités similaires, dans des classes sans aucun rapport entre elles .

Par exemple, la classe Texte, la classe image et la classe Titre peuvent avoir une fonction affiche.

Le polymorphisme paramétrique (également généricité ou en anglais template)

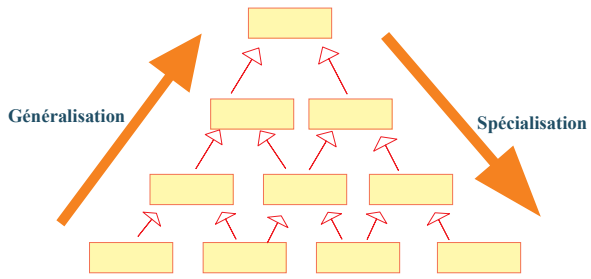
Le polymorphisme paramétrique, représente la possibilité de définir plusieurs fonctions de même nom mais possédant des paramètres différents (en nombre et/ou en type).

U M L — L'Héritage

L'héritage est un concept qui permet de créer de nouveaux objet depuis des objets existants.

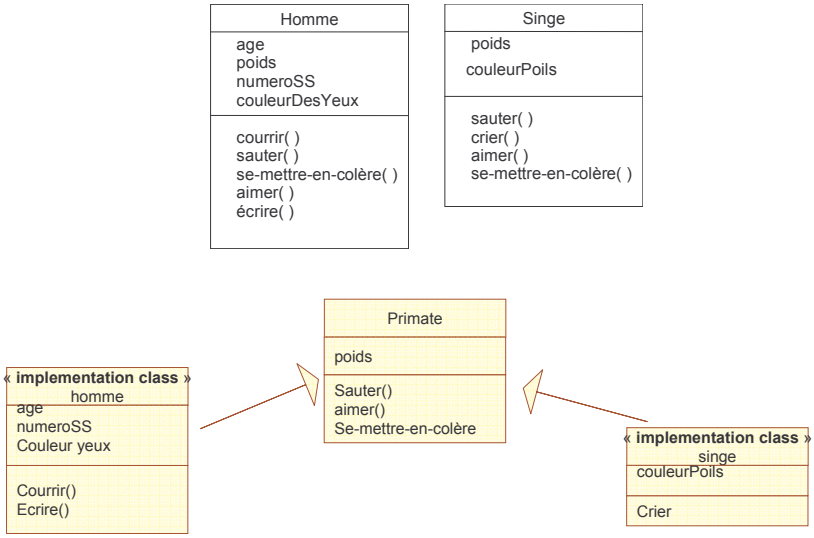
L'objet dérivé contient les attributs et les méthodes de sa superclasse (l'objet dont il dérive).

Par ce moyen, on créé une hiérarchie de plus en plus spécialisée d'objets.



L'Héritage

Exercice : généralisez les deux classes suivantes à l'aide d'une classe primate



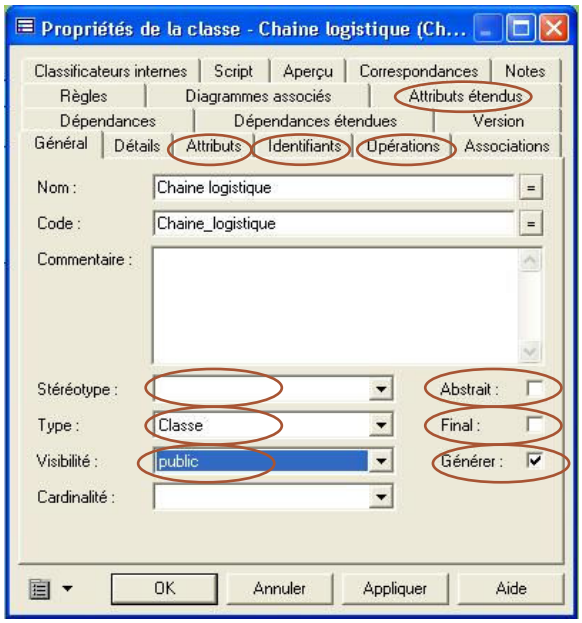
Check point

Point de compréhension no 7

Nous avons revu et précisé

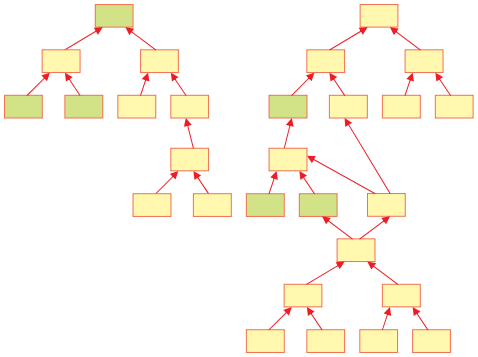
- **Les concepts d'objet, de classe et de type**
- **Ce qu'est un stéréotype**
- **Le concept d'encapsulation et de ce qu'est l'interface**
- **Le concept de polymorphisme et des classes abstraites**
- **Le concept d'héritage**

U M L — Check point



U M L — Au début d'une modélisation…

Au début de l'approche, on identifie des classes dont on va avoir besoin.



Puis on identifie les instances de ces classes en les spécialisant

Puis on identifie leurs super-classes en les généralisant

Mais la démarche est trop empirique

Car on ne sait pas où on se situent dans la hiérarchie finale des classes.

Cette approche présente donc toutes les caractéristiques de projet dont on connaît la date de début mais dont il est impossible de connaître la fin.

Sans nier l'importance de l'intuition et de la puissance de la logique déductive, il n'en demeure pas moins que nous avons besoin d'une méthode pour encadrer l'analyse.

U M L Trouver des méthodes

Une méthode d'analyse

Pour définir les limites du système étudié

(De quoi parle-t-on?
Jusqu'où doit-on aller ?)

Pour définir les « points de vue » desquels on se place pour décrire le système

(Comment en parlons-nous ?)

U M L Les représentations disponibles

➢**Un langage de modélisation Qui permet de modéliser tous les phénomènes de l'activité de l'entreprise :**

Processus métier, systèmes d'information, systèmes informatiques , composants logiciels, …

➢**Un langage de modélisation au sens de la théorie des langages**

Il contient donc des concepts, une syntaxe, une sémantique

Eléments de modélisation	Diagrammes	Vues
Acteurs	**De Use Case**	**Use case**
Classes	**D'Objets**	Logique (conception)
Objets	**De séquence**	Composants (Gestion)
Liens	**De classes**	Processus (exécution)
Noeuds	De collaboration	Déploiement (performance)
Message	D'états transitions	
Composant	D'activités	
État transition	De composant	
Activités transition	De déploiement	

U M L Les représentations disponibles

Quel diagramme utiliser et pour représenter quoi ?

Vue / Diagramme	Use case	Logique	composants	processus	déploiement
Cas d'utilisation	Acteurs Use case				
Objets	Acteurs Objets liens	Acteurs, Classes Objet, liens			
collaboration	Acteurs, Objets Message, liens	Acteurs, Classes Objet, liens		Classes Objets , liens	
Séquence	Acteurs Objets message	Acteurs Objets message		Objets message	
Classes		Acteurs, Classes Paquetages Relations			
Etat-transition	Etat-transition	Etat-transition		Etat-transition	
Activités	Activités	Activités		Activités	
Composants			Composants	composants	composants
Déploiement					Nœuds Liens

U M L Les représentations disponibles

Point de départ : Diagramme de cas d'utilisation

Diagramme de collaboration

Diagramme d'activité

Diagramme de séquence

Diagramme d'état transition

Diagramme de classe

Diagramme d'objet

Diagramme de composant

Point d'arrivée : diagramme de déploiement

Et les limites ????

UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles !

Il y a bien l'OOA (Object Oriented Analysis) : consiste à définir et qualifier dans un premier temps les éléments du système sous forme de types

Mais cela ne sert ni à fixer le périmètre de l'étude ni à déterminer les limites de l'effort d'abstraction

D'où

La nécessité de se recentrer sur le besoin fonctionnel de l'utilisateur

D'où

Des concessions à faire à l'approche fonctionnelle pour initialiser le projet

D'où

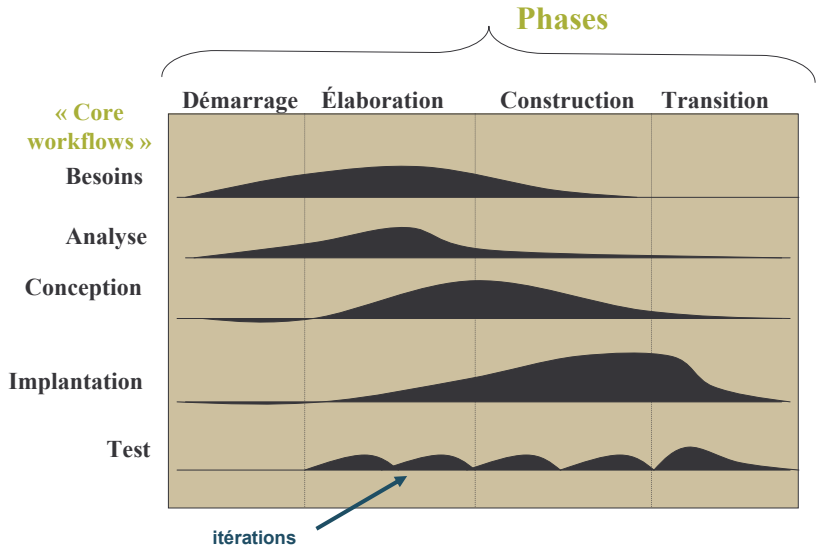
Des méthodes basées sur la vue utilisateur et l'utilisation de diagrammes de cas d'utilisation

UML

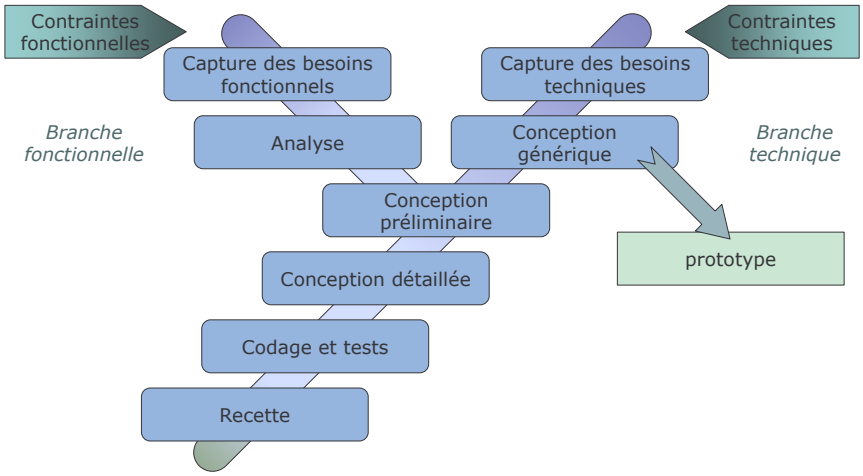
La méthode d'analyse

- Organisation en 4 phases :
  - pré-étude
  - élaboration
  - construction
  - transition
- Activités de développement définies par 5 workflows fondamentaux :
  - capture des besoins
  - analyse
  - conception
  - implémentation
  - test

Planification de la charge de travail



Le processus de développement en Y : le 2TUP ("2 Tracks UP")



U M L La méthode : un processus unifié

Le processus de développement en Y : le 2TUP

- Branche fonctionnelle :
  - capitalisation de la connaissance du métier de l'entreprise
- Branche technique :
  - capitalisation d'un savoir-faire technique
- Les 2 branches sont modifiables indépendamment l'une de l'autre.

Cours de SI Bordeaux I

Le processus de développement en Y : le 2TUP

- ***Incrément :*** ensemble d'étapes de développement aboutissant à la construction de tout ou partie du système.
- Chaque incrément passe en revue toutes les activités du processus en Y (l'effort consacré à chaque activité variant d'un incrément à l'autre).

U M L La méthode : un processus unifié

Le processus de développement en Y : le 2TUP

- Chaque phase de gestion de projet peut inclure un ou plusieurs incréments :
  - en phase de pré-étude : évaluation de la valeur ajoutée du développement et de la capacité technique à réaliser ce développement
  - en phase d'élaboration : analyse de l'adéquation du système aux besoins des utilisateurs
  - en phase de construction : livraison progressive des fonctions du système
  - en phase de transition : correction et évolution du système

Cours de SI Bordeaux I

Héritage de l'approche fonctionnelle

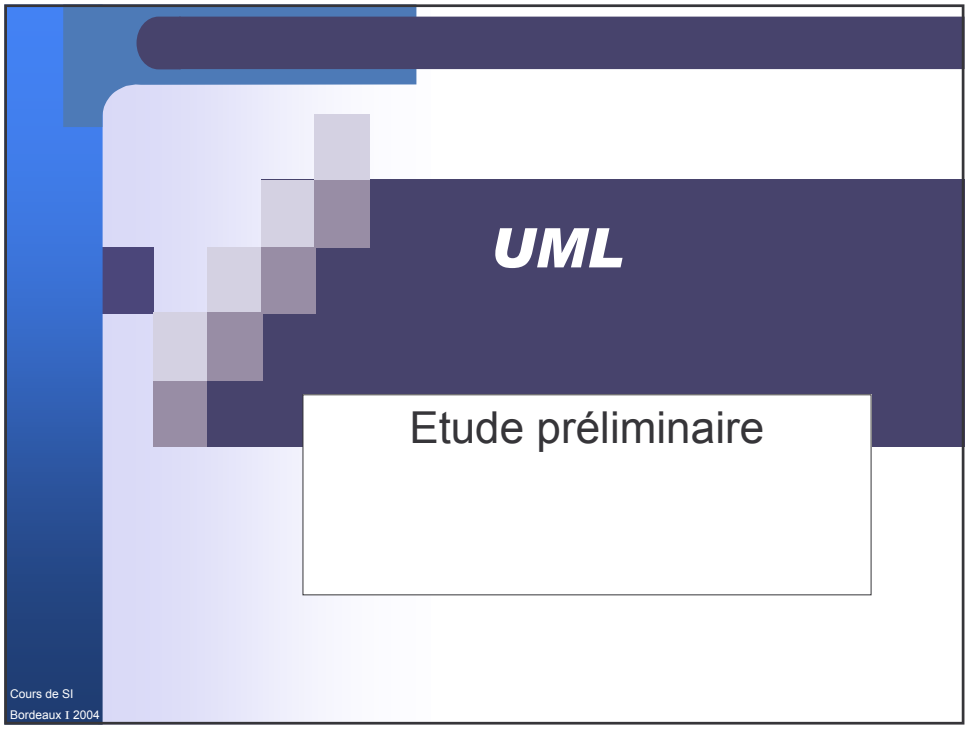
L'étude de l'existant ne donne pas lieu à autant de modélisations, mais elle est comprise dans l'étude préliminaire et inclue les aspects techniques.

Dans la branche fonctionnelle, la définition des packages et des modèles correspond à un découpage fonctionnel traditionnel.

Innovations apportées

Prise en compte des contraintes d'architecture applicative et technique dès le début du projet.

L'incrément : Un processus itératif qui est fondamental

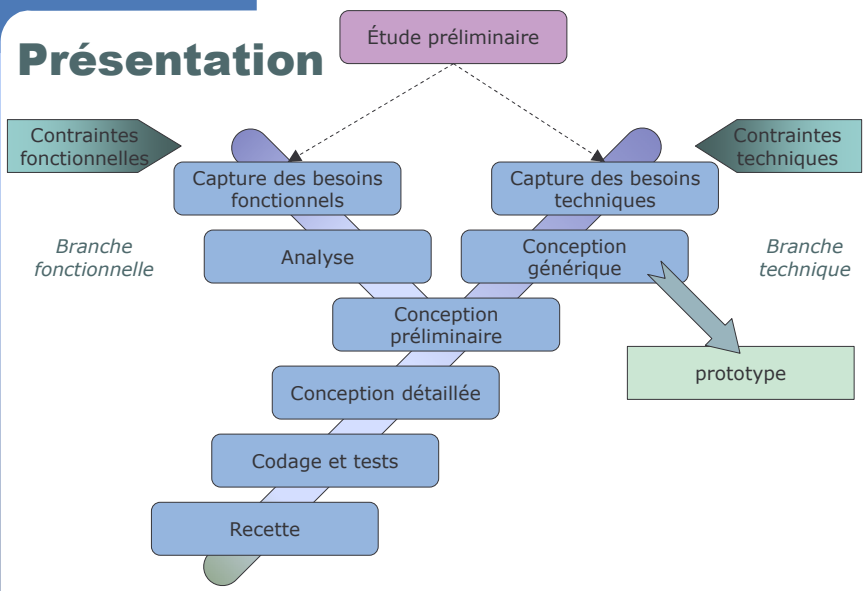


Présentation

- Rôle :
  - poser les bases de la capture des besoins fonctionnels et opérationnels
  - prépare les étapes formelles de capture des besoins

U M L Etude préliminaire

Présentation



Cours de SI
Bordeaux I

Activités

- L'étude préliminaire doit être faite en :
  - identifiant les acteurs,
  - identifiant les messages,
  - réalisant des diagrammes de contexte.

Identifier les acteurs

- Un *acteur* représente l'abstraction d'un **rôle** joué par des entités **externes** (utilisateur, dispositif matériel ou autre système) qui interagissent **directement** avec le système étudié.
- Un acteur peut consulter et/ou modifier directement l'état du système en émettant et/ou recevant des messages éventuellement porteurs de données.

Acteurs du système SIVEx

- **Réceptionniste** (saisie, annulation des commandes)
- **Client** (consultation des en-cours de commande, réception des confirmation de commande)
- **Progiciel de comptabilité** (récupération des données de facturation)
- **Comptable** (pointage des commandes, établissement des factures et avoirs, recouvrement des factures)
- **Répartiteur** (création et consultation des missions)
- **Chauffeur** (suivi des missions, avertissement au système en vas d'incident)
- **Opérateur de quai** (identification et pesée des colis, pointage des colis, réalisation des inventaires de quai)
- **Responsable logistique** (définition du réseau et maintien de la stratégie de transport)
- **Administrateur système** (gestion des profils)

Identifier les messages

- Un *message* représente la spécification d'une **communication** entre objets qui transporte de l'**information** avec l'intention de **déclencher une action** chez le récepteur.
- La réception d'un message est généralement considérée comme un *événement*.
- On ne s'intéresse ici qu'aux messages impliquant le système (en émission ou en réception).

Messages du système SIVEx

- Messages émis par SIVEx :
  - données de facturation pour le progiciel de comptabilité,
  - statistiques de transport pour le responsable logistique,
  - confirmations de commande pour le client,
  - incidents de mission pour le répartiteur,
  - …

U M L Etude préliminaire

Messages du système SIVEx

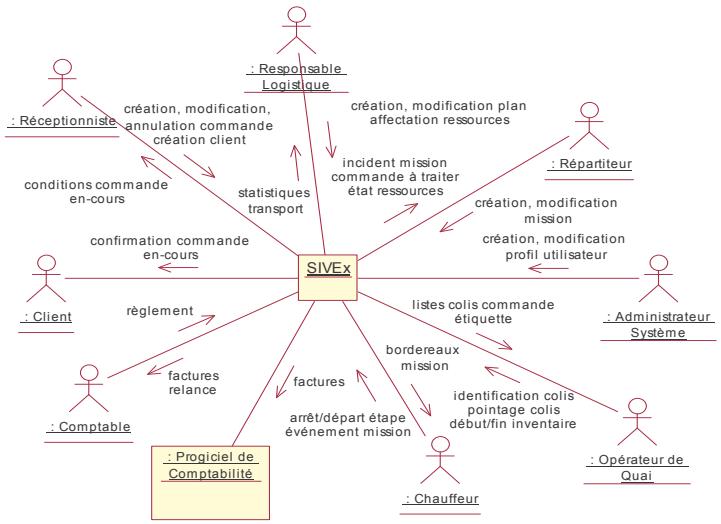
- Messages reçus par SIVEx :
  - créations, modifications ou annulations de commande par le réceptionniste,
  - règlements de facture par le comptable,
  - créations de mission par le répartiteur,
  - informations de suivi de mission par le chauffeur,
  - identification et pointage des colis par l'opérateur de quai,
  - …

Cours de SI Bordeaux I

Modéliser le contexte dynamique

- Un diagramme de contexte dynamique représente de façon synthétique les messages système ↔ acteurs identifiés.
- Ce diagramme est représenté au moyen d'un *diagramme de collaborations UML* en utilisant les « règles » suivantes :
  - le système est l'objet central ;
  - il est entouré des acteurs ;
  - des liens relient le système aux acteurs ;
  - sur chaque lien sont montrés les messages en entrée et en sortie du système.

Diagramme de contexte dynamique du système SIVEx



Description textuelle des messages du contexte

- Pour plus de lisibilité du diagramme de contexte dynamique, la description textuelle des messages est effectuée à part.
- Cette description doit indiquer le contenu des messages ainsi que le type de message (synchrone, asynchrone ou périodique).

Description textuelle des messages du contexte de SIVEx

- **factures :** ce message périodique (émis à la fin de chaque jour ouvrable) contient la liste des factures correspondant aux commandes réalisées avec succès dans la journée, ainsi que la consolidation quotidienne.
- **conditions commande :** ce message est émis systématiquement par SIVEx en réponse à une création ou une modification de commande effectuée par le réceptionniste. Il contient en particulier le coût estimé de la prestation ainsi que les dates prévues d'enlèvement et de livraison.
- **création mission :** ce message émis par le répartiteur lors de la création d'une nouvelle mission contient les données suivantes : type de mission, liste des étapes, commandes concernées, chauffeur et véhicule affectés, dates prévues de départ et d'arrivée.

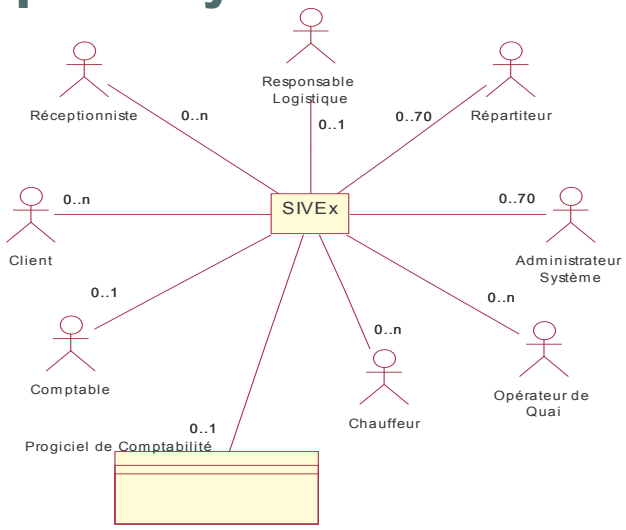
Modéliser le contexte statique

- Un diagramme de contexte statique peut compléter le diagramme de contexte dynamique.
- Le diagramme de contexte statique spécifie le nombre d'instances d'acteurs reliées au système à un moment donné.
- Ce diagramme est un *diagramme de classes UML* ne faisant intervenir que les acteurs et le système.

Contexte statique du système SIVEx

- Il permet de montrer de façon synthétique qu'il existe :
  - un seul responsable logistique, comptable et progiciel de comptabilité dans la société,
  - autant de répartiteurs et d'administrateurs système que d'agences (70),
  - un nombre non défini de clients, réceptionnistes, chauffeurs et opérateurs de quai.

Diagramme de contexte statique du système SIVEx



Décomposer le système en sous-systèmes fonctionnels

- Pour les systèmes importants, cette décomposition est faite lors de la modélisation du contexte comme suit :
  - élaborer le modèle de contexte dynamique du système ;
  - traiter le système comme un objet composite contenant les différents sous-systèmes fonctionnels grâce à une inclusion graphique,
  - répartir les flots de messages du niveau système entre les sous-systèmes concernés ;
  - ajouter les principaux flots de messages entre les sous-systèmes deux à deux.

On utilise pour cela les packages

- Un package UML représente un espace de nommage qui peut contenir :
  - des éléments d'un modèle,
  - des diagrammes qui représentent les éléments d'un modèle,
  - d'autres packages.
- Les éléments contenus dans un package :
  - doivent représenter un ensemble fortement cohérent,
  - sont généralement de même nature et de même niveau sémantique.
- Chaque package de cas d'utilisation occasion la création d'un diagramme.

Il faut comprendre que tous les concepts d'héritage, d'encapsulation, et de polymorphisme s'appliquent à tous les objets du langage

La modélisation dynamique va donc représenter les packages et les messages qu'ils échangent

La modélisation statique va elle représenter la hiérarchie éventuelle des packages

Des classes peuvent participer à plusieurs packages. Dans ce cas il y a un couplage qui doit faire poser la question du regroupement des packages ou non.

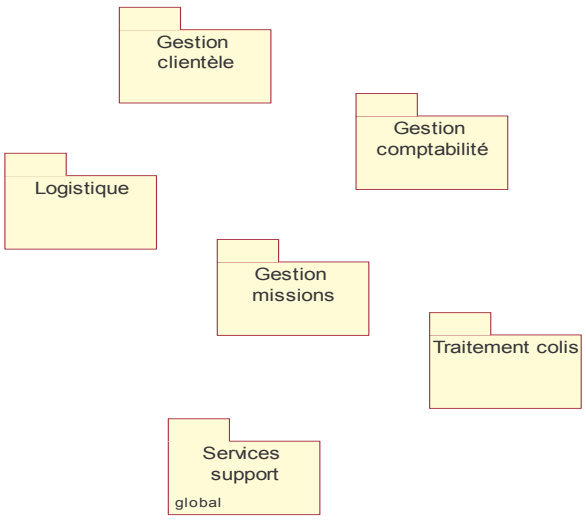
Il ne doit pas y avoir de couplage fort entre package (pas d'héritage entre classes, seulement des associations sont possibles)

Les packages de SIVEx

Cas d'utilisation	Acteurs	Package
Gestion de commande	Réceptionniste, Client	Gestion clientèle
Gestion des clients	Réceptionniste	
Consultation d'en-cours	Client, Réceptionniste	
Planification des missions	Répartiteur, Chauffeur	Gestion missions
Suivi de mission	Chauffeur, Répartiteur	
Facturation	Progiciel de comptabilité, Comptable	Comptabilité
Suivi des règlements	Comptable	
Inventaire de quai	Opérateur de quai	Traitement colis
Manipulation de colis	Opérateur de quai	
Définition du plan de transport	Responsable logistique	Logistique
Définition des ressources	Responsable logistique	
Définition des profils utilisateurs	Administrateur	Services support

U M L Etude préliminaire

Les packages de SIVEx



Cours de SI
Bordeaux I

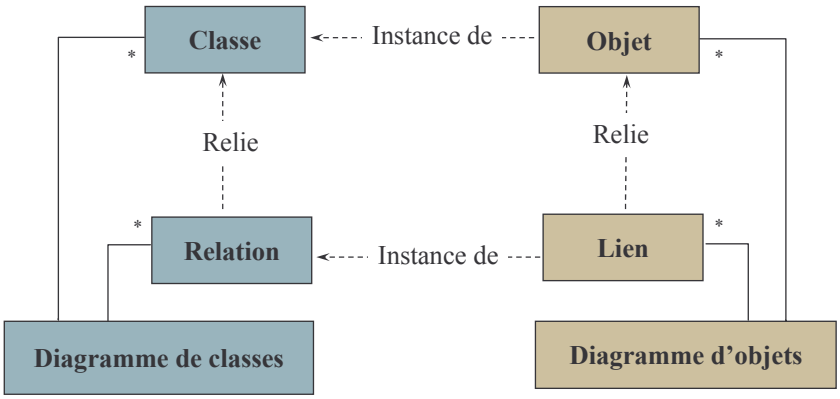
Conclusion

- Objectifs de l'étude préliminaire :
  - établir un recueil initial des besoins fonctionnels et opérationnels,
  - modéliser le contexte du système, considéré comme une boîte noire, en
    - identifiant les entités externes au système qui interagissent directement avec lui (acteurs),
    - répertoriant les interactions (émission/réception de messages) entre ces acteurs et le système,
    - représentant l'ensemble des interactions sur un modèle de contexte dynamique, éventuellement complété par un modèle de contexte statique, ou décomposé pour faire apparaître les principaux systèmes fonctionnels.

UML

Le symbolisme de base

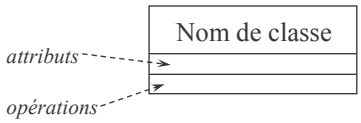
Classes et Objets



Classes

- Une *classe* est une description d'un **ensemble d'objets** ayant des **propriétés similaires** (attributs), un **comportement commun** (opérations), des **relations communes** avec d'autres objets et des **sémantiques communes**.

Représentation d'une classe



Nom de classe

Nom du paquetage::Nom de classe

Exemples :

Personne

Poste de travail

Département

Personne
Nom Prénom Date naissance
Age()

Personne
{auteur = Jean,
état = vérifié,
par = Julie}

Une classe avec des propriétés

Attributs d'une classe

- Ils définissent les propriétés des objets d'une classe.

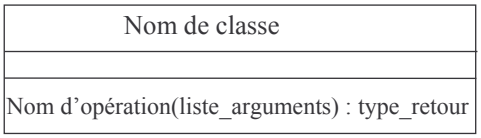
Nom de classe
Nom d'attribut : type = valeur initiale

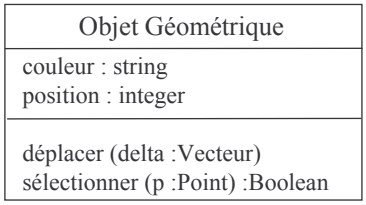
Personne
nom : string prénom : string date_naiss : date

Film
titre : string réalisateur : string date_production : date date_sortie : date

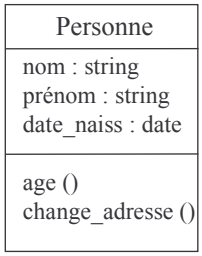
Opérations d'une classe

- Elles définissent les fonctions appliquées à des objets d'une classe.





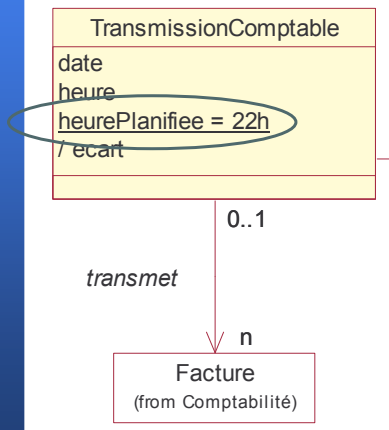
Opérations avec leur signature



Visibilité des attributs et des opérations

- UML définit 3 niveaux de visibilité :
  - `public` (+) : qui rend l'élément visible à tous les clients de la classe
  - `protected` (#) : qui rend l'élément visible aux sous-classes de la classe
  - `private` (-) : qui rend l'élément visible à la classe seule

Attributs et opérations de classes

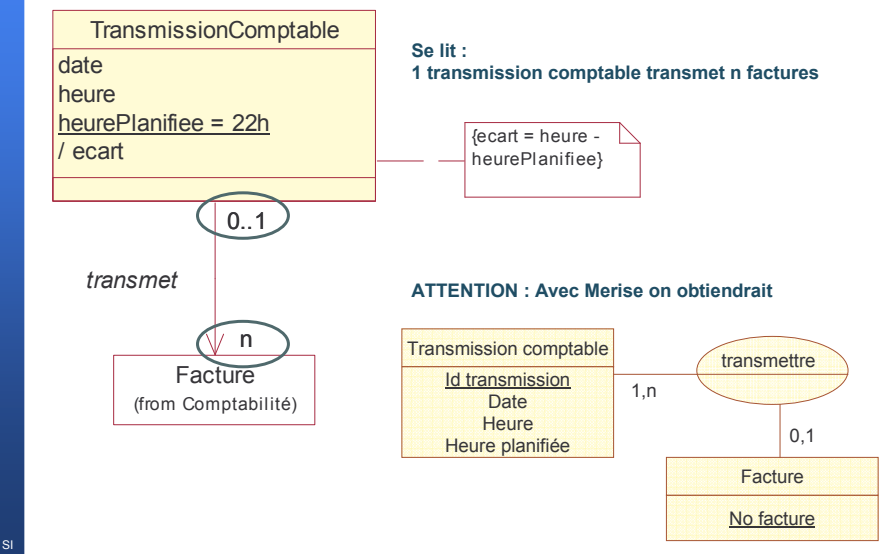


Certains attributs et certaines opérations peuvent être visibles globalement, dans toute la portée lexicale de la classe.

Ces attributs et ces opérations sont alors communs à toutes les instances de la classe.

Ils sont signalés en étant soulignés.

Multiplicité



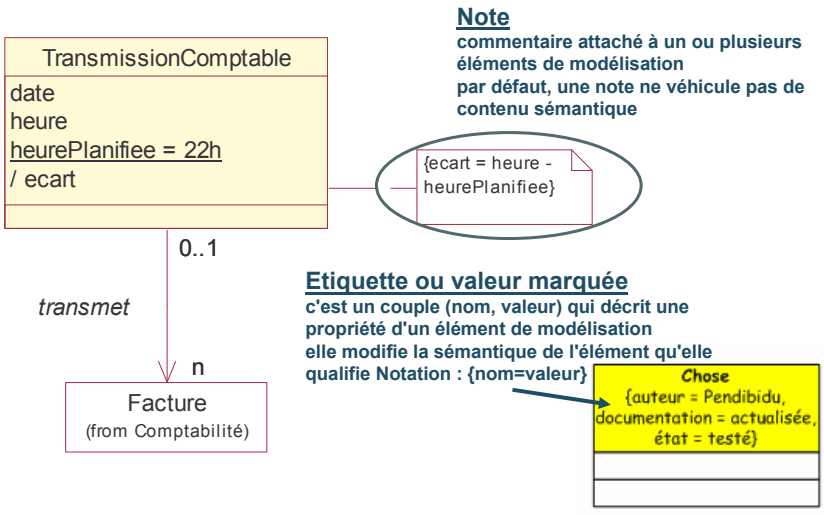
Se lit :
1 transmission comptable transmet n factures

ATTENTION : Avec Merise on obtiendrait

Multiplicité

1	Un et un seul
0..1	Zéro ou un
M..N	De M à N (entiers naturels)
* ou 0..*	De zéro à plusieurs
1..*	De un à plusieurs

Notes et étiquettes

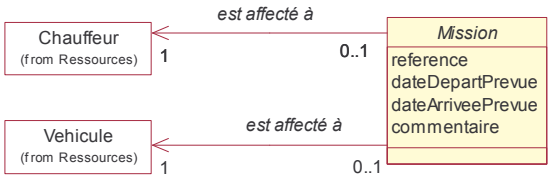


Note
commentaire attaché à un ou plusieurs éléments de modélisation
par défaut, une note ne véhicule pas de contenu sémantique

Etiquette ou valeur marquée
c'est un couple (nom, valeur) qui décrit une propriété d'un élément de modélisation
elle modifie la sémantique de l'élément qu'elle qualifie Notation : {nom=valeur}

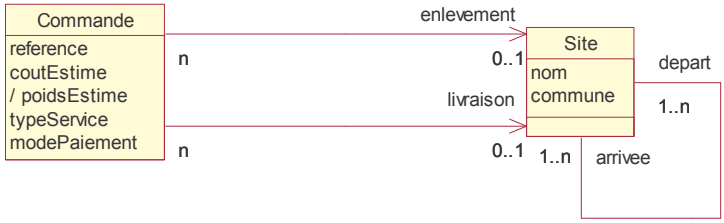
Associations

- Elles représentent une **relation structurelle** entre objets.
- Une association symbolise une information dont la durée de vie n'est pas négligeable par rapport à la dynamique générale des objets instances des classes associées.



Rôles d'associations

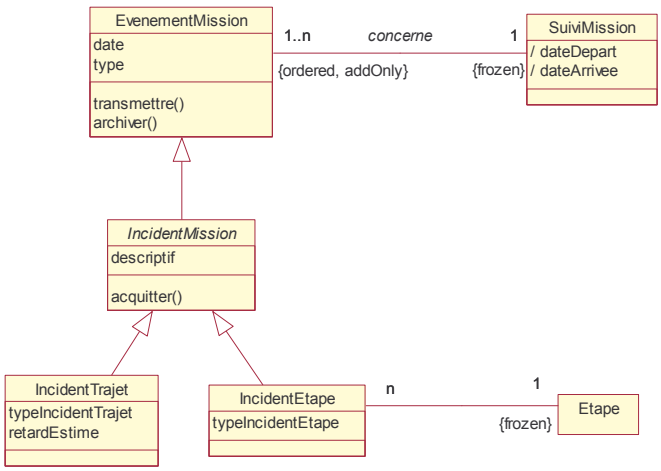
- Chaque association binaire possède 2 rôles.
- Le rôle décrit comment une classe voit une autre classe au travers d'une association.
- Le nommage des associations et le nommage des rôles ne sont pas exclusifs l'un de l'autre.



Propriétés des rôles

- UML propose un certain nombre de propriétés standards, notamment :
  - l'ordonnancement (« `{ordered}` ») ;
  - « `{frozen}` » indique qu'un lien ne peut être modifié ni détruit ;
  - « `{addOnly}` » signifie que de nouveaux liens peuvent être ajoutés depuis un objet de l'autre côté de l'association mais non supprimés.

Propriétés des rôles



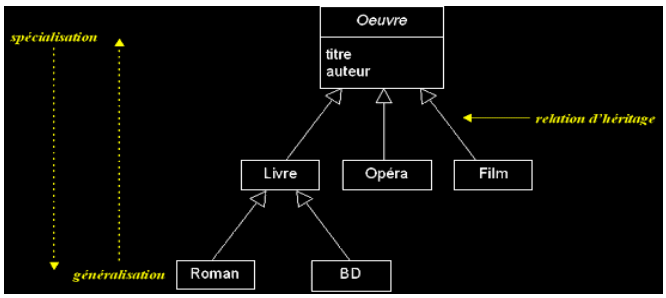
Formalisation de l'héritage

L'héritage est un 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.

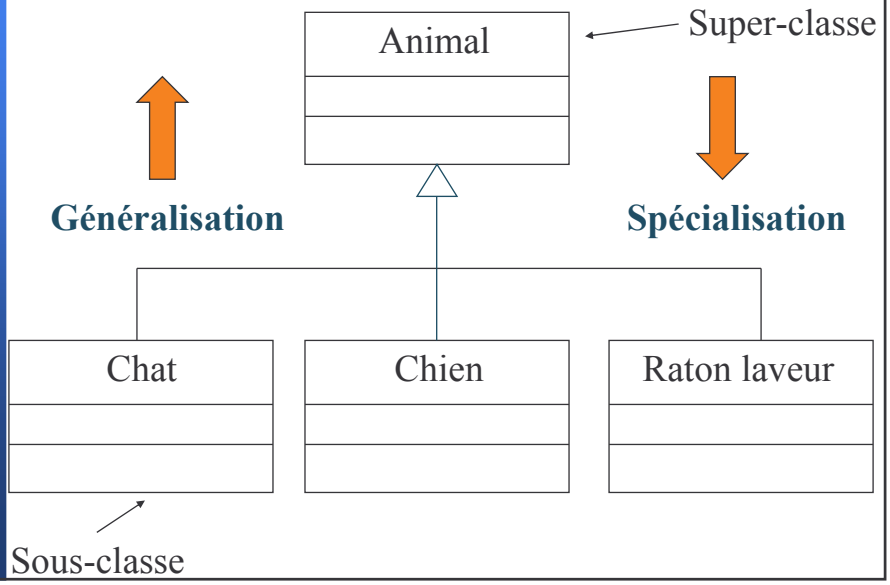
Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.

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.



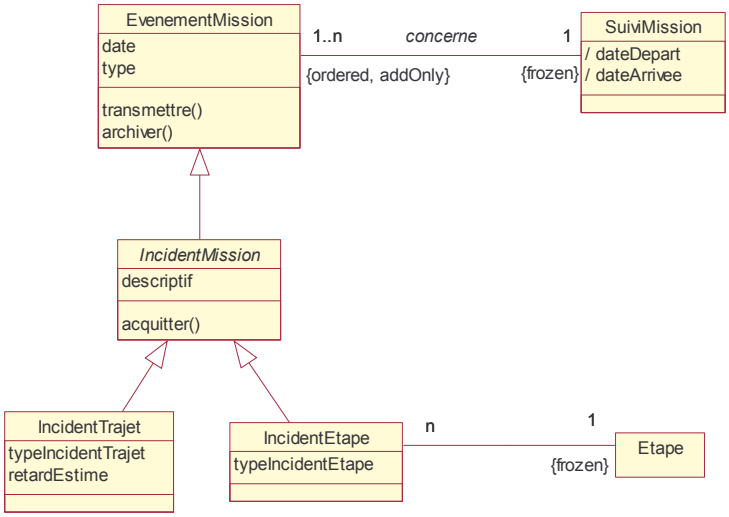
Généralisation simple

Formalisation de l'héritage



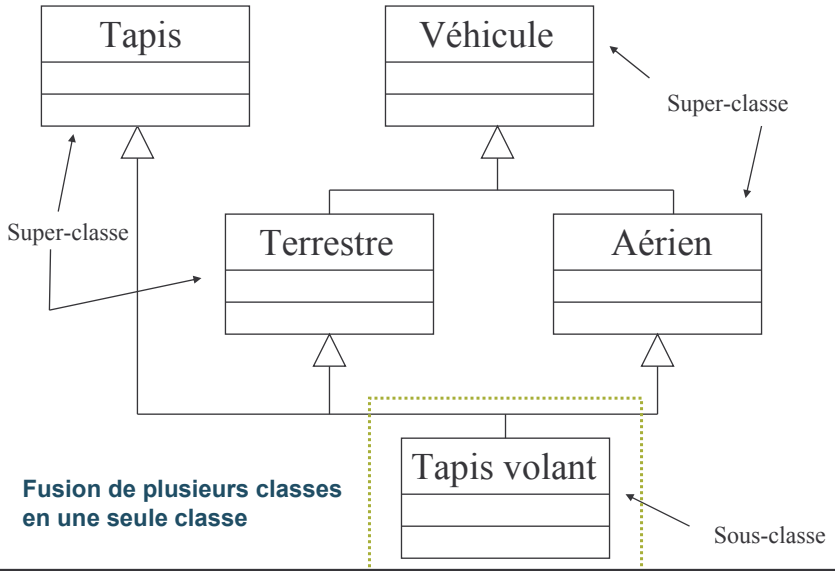
Généralisation simple

Formalisation de l'héritage



Généralisation multiple

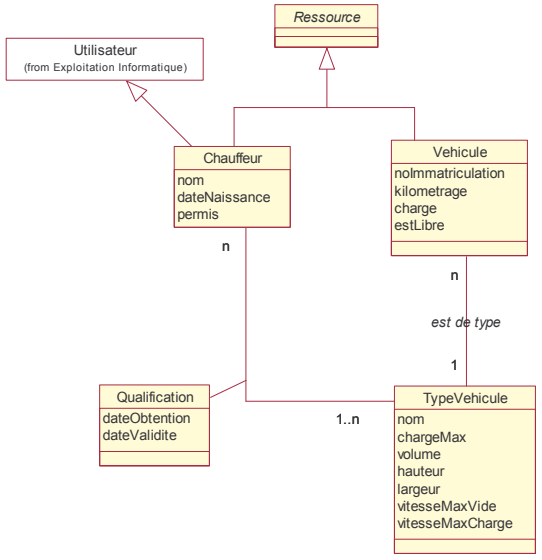
Formalisation de l'héritage



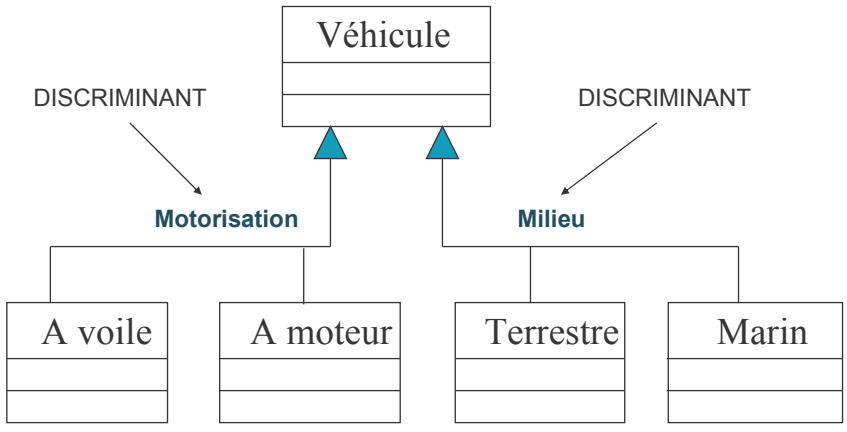
Fusion de plusieurs classes en une seule classe

Généralisation multiple

Formalisation de l'héritage



Généralisation avec discriminant



Généralisation

Formalisation de l'héritage

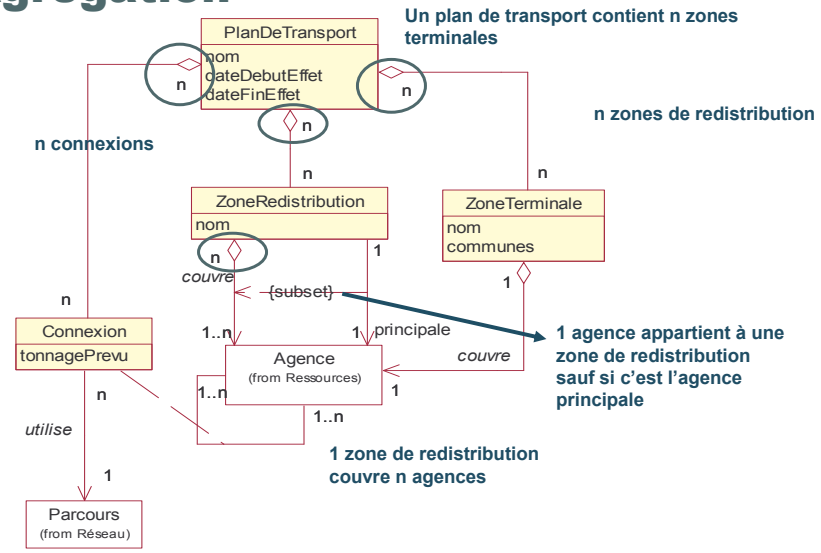
- La généralisation s'applique principalement :
  - aux classes,
  - aux paquetages
  - aux cas d'utilisation.
- Dans le cas des classes, la relation de généralisation signifie « ***est un*** » ou « ***est une sorte de*** ».

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.

- L'agrégation représente une association non symétrique dans laquelle une des extrémités joue un rôle prédominant par rapport à l'autre extrémité.
- **Les critères** suivants impliquent une agrégation :
  - ***une classe fait partie d'une autre classe ;***
  - ***les valeurs d'attributs d'une classe se propagent dans les valeurs d'attributs d'une autre classe***
  - ***une action sur une classe implique une action sur une autre classe ;***
  - ***les objets d'une classe sont subordonnés aux objets d'une autre classe.***
- L'agrégation peut être multiple, comme l'association.

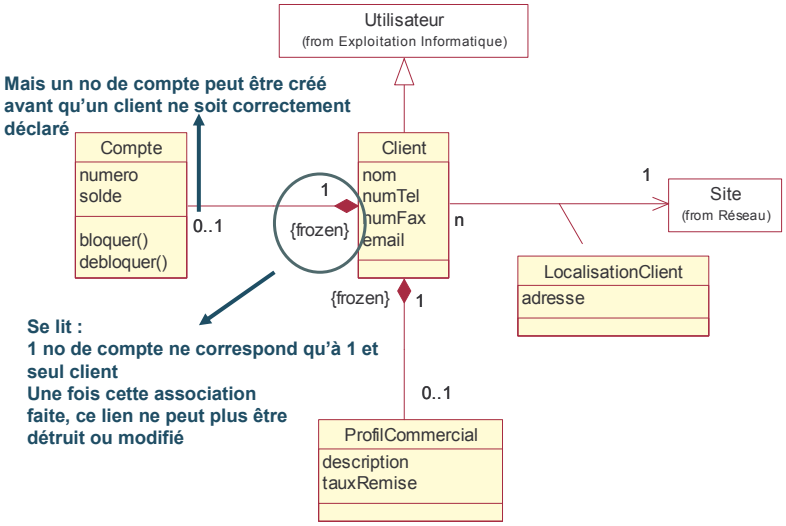
Agrégation



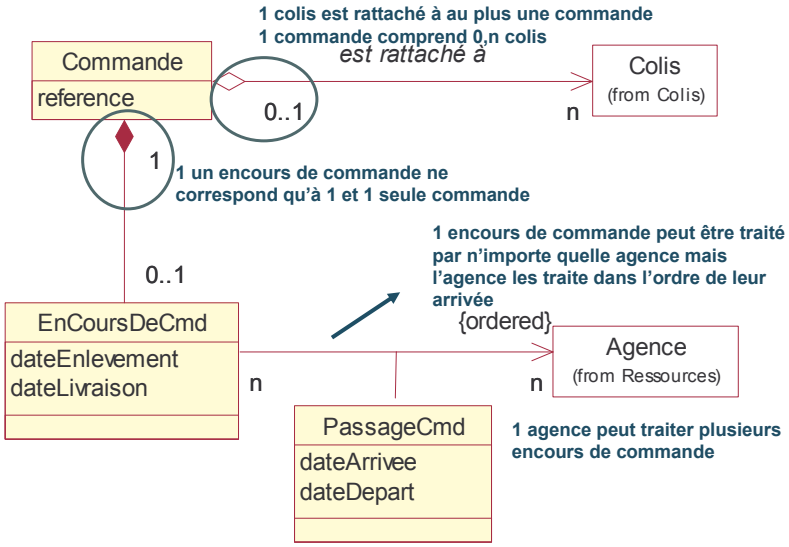
Composition

- Il s'agit d'une agrégation réalisée par valeur sur des attributs.
- Ils sont physiquement contenus dans l'agrégat.
- La composition implique une contrainte sur la valeur de la multiplicité du côté de l'agrégat : elle ne peut prendre que les valeurs 0 ou 1 (un composant ne peut être partageable).
- La destruction du composite entraîne la destruction des composants.
- La valeur 0 du côté du composant correspond à un attribut non renseigné.

Composition



Agrégation / composition



U M L Différence agrégation/généralisation

L'agrégation n'a rien à voir avec la généralisation :

La généralisation est une relation **sorte-de** entre des **classes**

L'agrégation (ainsi que la composition) est une relation **partie-tout** entre des **instances**.

composition — agrégation — généralisation

Niveau physique — **Niveau logique**

Vue composant

Vue processus

Vue logique

Une nomenclature physique utiliserait plutôt la **composition**.

L'**agrégation** indique une nomenclature logique (un article peut être composant dans plusieurs produits).

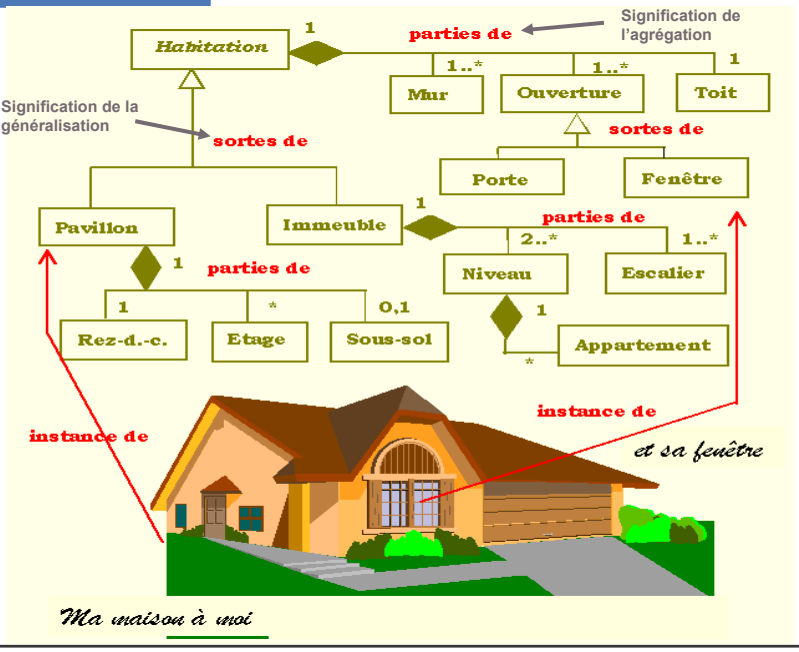
Diagramme de composant

Diagramme de déploiement

Diagramme de classe

Diagramme d'objet

U M L Différence agrégation/généralisation

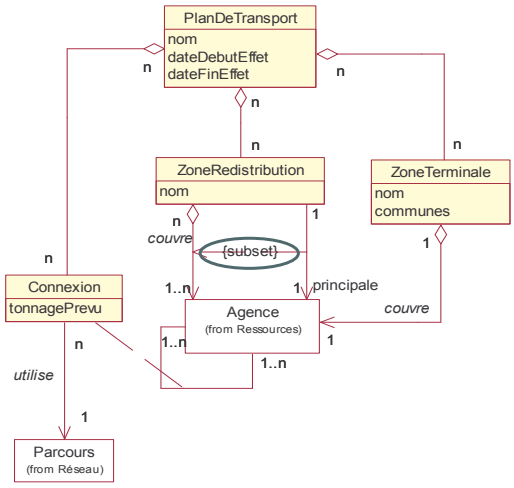


Contraintes

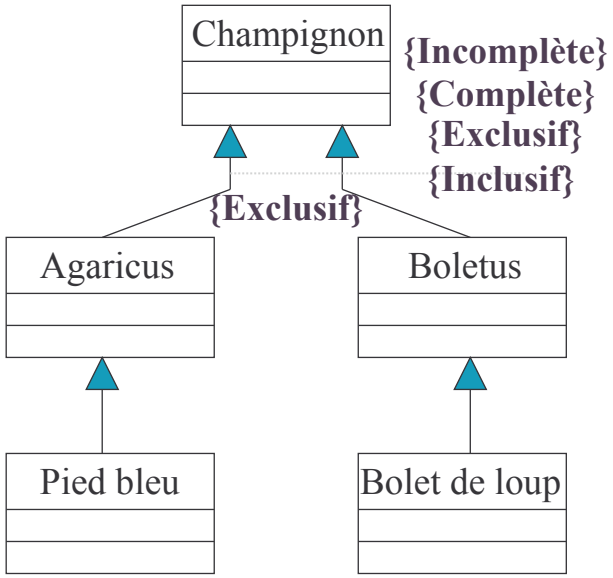
- relation sémantique (quelconque) entre éléments de modélisation
- *exemple :* `{incomplet}` sur une généralisation

Contraintes entre associations

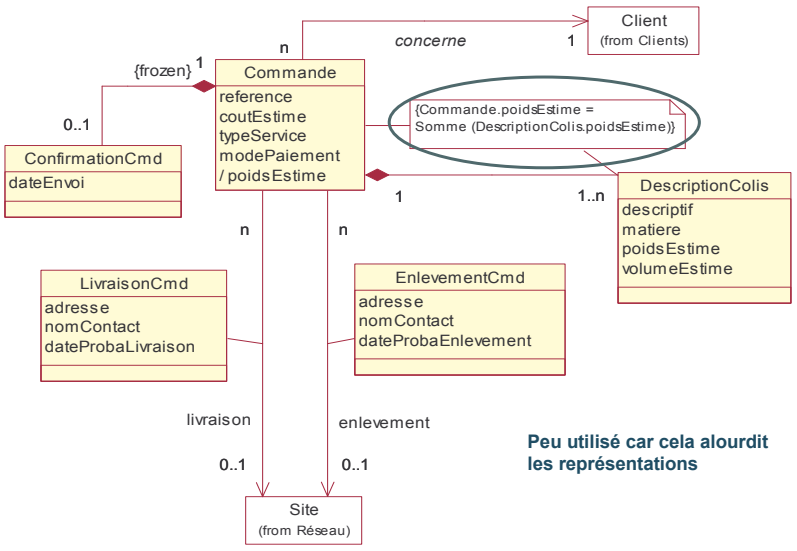
- UML définit notamment les contraintes « `{subset}` » et « `{XOR}` ».



Généralisation avec contrainte



Contraintes entre attributs



Peu utilisé car cela alourdit les représentations

Point de compréhension no 8

Nous avons parlé

Des concepts fondamentaux de l'objet

De la méthode 2 track up

Des représentations d'UML

Des outils de formalisme UML

Bibliographie

• Le processus unifié de développement logiciel – collection technologies objet/référence – 2000 '- G booch, J rumbaugh, I jacobson

• Modélisation objet avec UML 2ème édition – 2000 – PA Muller, N Gaertner

• Introduction au Rational Unified Process – collection technologies objet/référence – 2000 '- P kruchten

• Introduction au Rational Unified Process – collection technologies objet/référence – 2000 '- P kruchten

• UML par la pratique – édition eyrolles – 2004 '- Pascal Roques