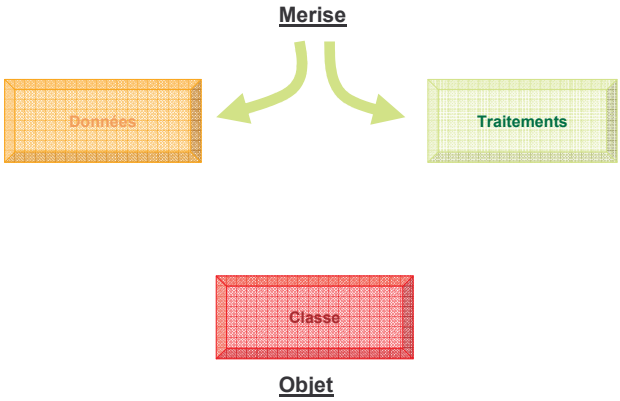
Approche objet

Objet

- Origines de l'approche objet
- Caractéristiques de l'approche objet
  - Héritage
  - Polymorphisme
  - Encapsulation
  - Qu'est-ce qu'un objet ?
  - Les familles de langage
- Pourquoi l'objet aujourd'hui ?
- Les méthodes d'analyse objet

U M L — Origines de l'approche objet



langages de simulation dans les années 1960
Intelligence artificielle dans les années 1970-80

Pourquoi ?
Pour permettre plus facilement des modifications *fondamentales* dans la façon de représenter et de traiter les données.

U M L — Origines de l'approche objet

Dans ces disciplines le modèle n'est plus déterministe.

Par exemple en intelligence artificielle, l'état suivant du système n'est pas connu

C'est le système lui-même qui va le déterminer en fonction de ses apprentissages antérieurs (machine neuronale)

On ne peut connaître l'état suivant du système qu'en faisant des statistiques sur l'utilisation qui a été préalablement faite du système.

Or nous ne pourrons effectuer ces statistiques qu'en créant justement un premier modèle opérationnel.

=> auto-référence

Ce système dit « apprenant » nécessite donc que l'on puisse changer le traitement des données facilement par un paramétrage rapide.

On doit pouvoir modifier le comportement d'un même traitement sur les données en fonction de l'évolution de son environnement.

Exemple

Soit une application qui utilise des nombres complexes

Dans cette application il faut qu'il soit possible de saisir et de lire des coordonnées en représentation polaire ou cartésienne selon ce qui nous arrange le mieux.

Par les méthodes de programmation structurée il faut alors faire la chasse à toutes les références utilisant des nombres complexes et à réécrire, sans jamais rien omettre, toutes les lectures et écritures là où cela est nécessaire.

Une autre manière de voir les choses est de faire une boite noire (nombre complexe) pour lui *affecter* des valeurs en représentation polaire ou cartésienne, pour pouvoir les *lire* en polaires ou en cartésiennes.

La classe nombre complexe modifie son comportement en fonction du besoin de son environnement.

C'est un moteur ou un compilateur qui se charge de trouver les références à la classe.

U M L Origines de l'approche objet

Exemple

Objectif :

Faire évoluer une application de gestion de bibliothèque pour gérer une médiathèque, afin de prendre en compte de nouveaux types d'ouvrages (cassettes vidéo, CD-ROM, etc...)

Nous allons nous intéresser à la modification de la fonction qui réalise l'édition des "lettres de rappel ».

Une lettre de rappel est une mise en demeure, qu'on envoie automatiquement aux personnes qui tardent à rendre un ouvrage emprunté). Si l'on désire que le délai avant rappel varie selon le type de document emprunté, il faut prévoir une règle de calcul pour chaque type de document

Cours de SI
Bordeaux I

Origines de l'approche objet

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

void mettre_a_jour(int ref)
  {
   /* ... */
   switch(DOC[ref].type)
   {
     case LIVRE:
       maj_livre(DOC[ref]);
       break;
     case CASSETTE_VIDEO:
       maj_k7(DOC[ref]);
       break;
     case CD_ROM:
       maj_cd(DOC[ref]);
       break;
   }
   /* ... */
  }
```

```
void lettres_de_rappel()
{
 /* ... */
 for (i = 0; i < NB_DOCS; i++)
 {
  if (DOC[i].est_emprunte)
  {
    switch(DOC[i].type)
    {
    case LIVRE:
      delai_avant_rappel = 20; break;
    case CASSETTE_VIDEO:
      delai_avant_rappel = 7; break;
    case CD_ROM:
      delai_avant_rappel = 5; break;
    }
  }
 }
/* ... */
}
```


Origines de l'approche objet

1ère amélioration : centraliser dans les structures de données

```
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];
```

Délai avant rappel est bien un attribut de tout média

```
void lettres_de_rappel()
{
 /* ... */
 for (i = 0; i < NB_DOCS; i++)
 {
   if (DOC[i].est_emprunte) /* SI LE DOC EST EMPRUNTE */
   {
     /* IMPRIME UNE LETTRE SI SON
     DELAI DE RAPPEL EST DEPASSE */

     if (date() >= (DOC[i].date_emprunt + DOC[i].delai_avant_rappel))
        imprimer_rappel(DOC[i]);
   }
 }
}
```

Origines de l'approche objet

2ème amélioration : centraliser les traitements associés à un type

```
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 ! */
  }
}
```

```
void ajouter_document(int ref)
{
  DOC[ref].est_emprunte = FAUX;
  DOC[ref].nom_doc = saisir_nom();
  DOC[ref].type = saisir_type();
  DOC[ref].delai_avant_rappel = calculer_delai_rappel(DOC[ref].type);
}
```

```
void afficher_document(int ref)
{
  printf("Nom du document: %s\n",DOC[ref].nom_doc);
  /* ... */
  printf("Delai avant rappel: %d jours\n",DOC[ref].delai_avant_rappel);
  /* ... */
}
```

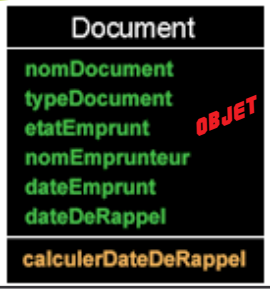

Origines de l'approche objet

```
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 ! */
  }
}
```

D' une structure de données, manipulée par des fonctions, nous avons fait une entité autonome

qui regroupe un ensemble de propriétés cohérentes et de traitements associés.



U M L — Origines de l'approche objet

Au départ l'objet était donc une méthode de programmation plus qu'autre chose.

Aujourd'hui, l'Objet est vu davantage comme une approche utile, un paradigme, qu'une simple technique de programmation

L'Objet n'a cessé d'évoluer aussi bien dans son aspect théorique que pratique et différents *métiers* et *discours marketing* à son sujet ont vu le jour comme l'analyse objet (AOO ou OOA en anglais), la conception objet (COO ou OOD en anglais) ou base de données objet (SGBDOO).

U M L — Caractéristiques de l'approche objet

HERITAGE

les systèmes sont uniquement constitués d'entités (appelées *objet*) ayant chacune des caractéristiques qui sont définies par leur type

Deux grandes théories du type :

Liskov

Typage de premier ordre

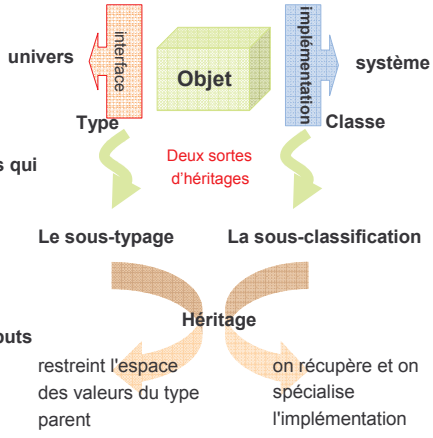
Type = interface qui rassemble ses propriétés qui sont les aspects visibles de l'objet

Classe = façon dont ces propriétés sont implémentées

Attributs = espaces mémoire , données

méthodes = opérations faites sur les attributs

JAVA - C++



U M L Caractéristiques de l'approche objet

POLYMORPHISME

Liskov ne résoud pas le problème posé par le sous-typage des types récursifs (les types avec au moins une opération qui accepte un objet de même type)

Deux grandes théories du type :

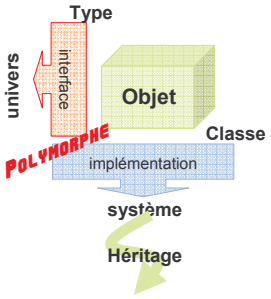
Cook

Typage de second ordre — les concepts de classe et de type ne sont plus duals mais imbriqués

Classe = définit l'implémentation d'une famille *polymorphique* finie et liée de types

Le type est devenu un élement d'une classe

Attributs = espaces mémoire , données
méthodes = opérations faites sur les attributs
Type = interface qui rassemble ses propriétés qui sont les aspects visibles de l'objet



Smalltalk - Eiffel

La sous-classification

un héritage va décrire l'arborescence entre classes (et donc implicitement entre type)

U M L Caractéristiques de l'approche objet

HERITAGE

En résumé:

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

Ce mécanisme est appelé "héritage" car l'objet dérivé (nouvellement créé) 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.

Par exemple les carrés ne sont qu'une spécialisation des rectangles et ceux-ci une spécialisation des quadrilatères.

U M L Caractéristiques de l'approche objet

POLYMORPHISME

En résumé:

Ce mécanisme essentiel de la programmation orientée objet caractérise la possibilité de définir plusieurs fonctions de même nom mais possédant des paramètres différents (en nombre et/ou en type).

Par exemple la méthode *getArea()* retournera l'aire du polygône sur lequel elle sera appliquée en utilisant la méthode appropriée pour un carré ou pour un rectangle.

U M L Caractéristiques de l'approche objet

ENCAPSULATION

Quelque soit l'approche adoptée on découple :

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

-**L'implémentation (la façon dont on va pratiquement représenter les choses par un savant mélange de *mémoire* et d'opérations)**

En fait l'implémentation est cachée de l'extérieur par l'interface

Les objets n'utilisent des autres objets que les propriétés définies par leur interface, quelque soit leur représentation interne.

Exemple :
Le programmeur ne sait pas si le nom qu'il emploie est une variable ou une fonction sans argument
Ou
Un langage qui permet de définir alternativement un même nom comme celui d'une variable ou d'une fonction selon l'empilement des contextes (« *localisation des noms* ») à l'exécution.

U M L Caractéristiques de l'approche objet

ENCAPSULATION

L'encapsulation est un mécanisme qui permet à un objet de laisser voir aux autres objets seulement l'information à laquelle ils ont droit.

C'est un mécanisme qui permet de regrouper, classifier et utiliser les données contenues dans un objet afin d'en assurer l'intégrité.

Les objets externes peuvent utiliser un ensemble bien défini de méthodes pour accéder aux fonctionnalités et données d'un autre objet.

Trois niveaux de visibilité sont utilisés :

public: les attributs dits publics sont accessibles à tous.

protégé: les attributs dits protégées sont accessibles seulement aux classes dérivées.

privé: les attributs privés sont accessibles seulement par l'objet lui-même.

U M L Caractéristiques de l'approche objet

Qu'est-ce qu'un OBJET ?

Un objet est caractérisé par une entité et des propriétés

Ses propriétés sont définies par son type

Ses propriétés peuvent être classées en trois catégories :

L'identification ou OID:

Permet généralement à un objet d'être référencé par d'autres de façon unique et constante pendant toute sa durée de vie

Géré par le système et donc transparent pour le développeur

L'état de l'entité

Valorise l'entité à un instant précis. Il est toujours conforme à son type.

Le comportement de l'entité

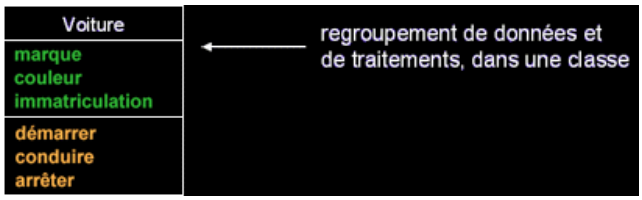
Définit les opérations que peut accomplir l'objet, invoqué via l'interface de son type . Peut impacter l'état de l'entité.

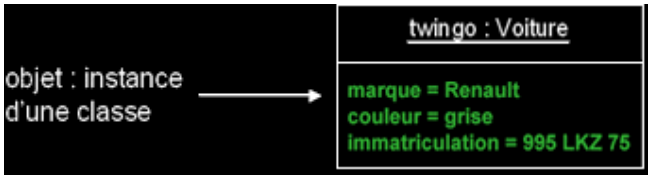
U M L Caractéristiques de l'approche objet

Qu'est-ce qu'un OBJET ?

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 et permettant de créer des objets possédant ces propriétés.





U M L Caractéristiques de l'approche objet

Famille de langages

On distingue deux grandes familles de langage objet dans la déclaration du type des objets.

Typage dynamique

Le type des objets est implicite et déterminé à l'exécution lors de la création desdits objets

JAVA - C++ - Eiffel

Typage statique

Le type des objets est spécifié explicitement par le développeur lors de leur déclaration

Smalltalk - Python

U M L Pourquoi l'approche objet aujourd'hui ?

Objet

langages de simulation dans les années 1960

Intelligence artificielle dans les années 1970-80

Le développement des applications Web 1990-2000

Les autoroutes de l'information (Al gore) LE RENOUVEAU

Une architecture imposée : le renouveau du middleware

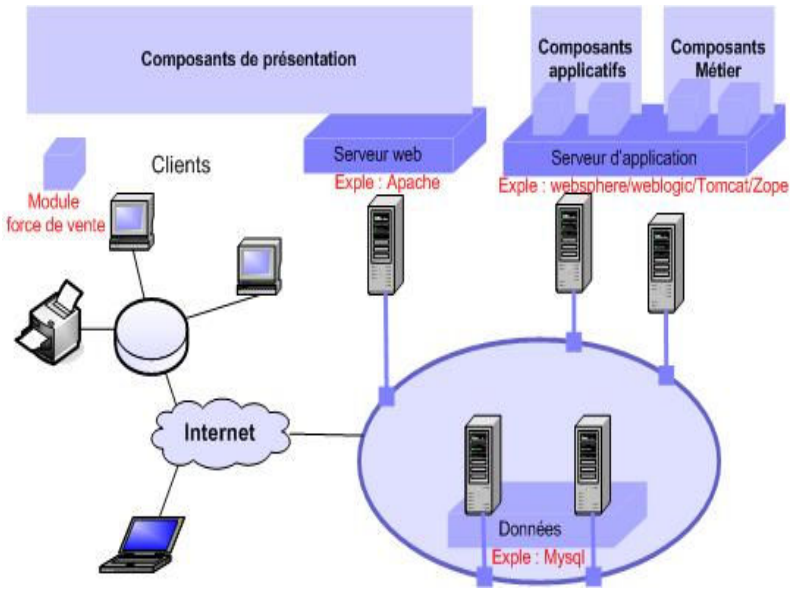
Des liaisons distantes peu performantes

Une délocalisation des utilisateurs et des applications

Un besoin de mobilité et d'adaptation rapide des organisations

U M L Pourquoi l'approche objet aujourd'hui ?

De fortes contraintes d'architecture



U M L Pourquoi l'approche objet aujourd'hui ?

Du 3 tiers au multi-niveaux

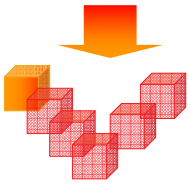
3 tiers

Performance

Traffic réseau

Maintenance

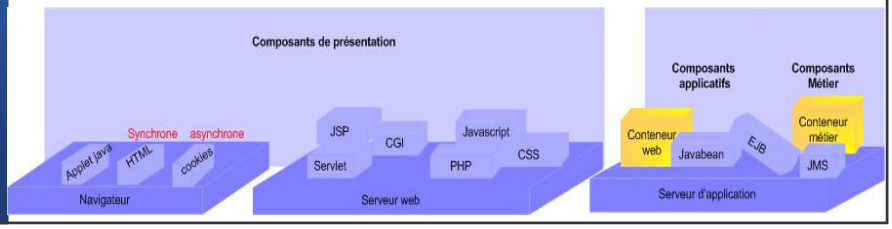
Multi niveaux



Réutilisation

Evolutivité

=> Les concepts de polymorphisme, d'encapsulation trouvent là toute leur place



U M L Pourquoi l'approche objet aujourd'hui ?

Délocalisation des applications et des acteurs

Les applications doivent pouvoir être implémentées sur différentes plate-formes => intéropérabilité => encapsulation et polymorphisme

Les composants métiers ou techniques communs à plusieurs applications doivent pouvoir être réutilisés afin de standardiser les processus et diminuer le risque d'erreur => Héritage et polymorphisme

Maintenir les composants malgré la complexité des architectures applicatives et techniques => Meta-classes

Architecture de bus de composants métier (CORBA)

Donc des concepts adaptés à la complexité et aux contraintes d'architecture technique des applications web.

U M L Les méthodes d'analyse objet

L'approche objet c'est :

- Un ensemble de concepts stables, éprouvés et normalisés.
- Une solution destinée à faciliter l'évolution d'applications complexes.
- Une panoplie d'outils et de langages performants pour le développement.

Mais…

- Une démarche non intuitive
- Un vocabulaire précis
- Un impératif de rigueur

Pour la mettre en pratique il nous faut :

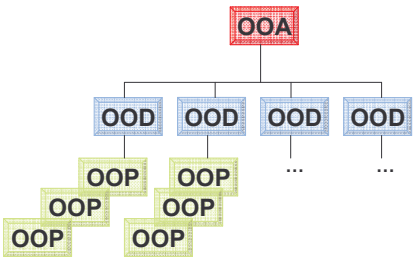
- Un langage
- Une démarche d'analyse et de conception objet

U M L Les méthodes d'analyse objet

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

OOD (Object Oriented Design) : consiste à proposer des solutions techniques pour représenter les éléments définis dans le système informatique.

OOP (Object Oriented Programming) : consiste à implémenter les solutions techniques à l'aide d'un langage de programmation



Cette méthode peut donc être facilement intégrée à des cycles de vie courts basés sur la réalisation de prototypes.

U M L Les méthodes d'analyse objet

Pour écrire ces différents modèles, différents langages et méthodes ont été proposées

En fait plus de 50 méthodes sont apparues dans les années 1990-1995 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE...) .

Les premiers consensus apparaissent en 1995:

James Rumbaugh

OMT : vues statiques, dynamiques et fonctionnelles d'un système
Issue du centre de R&D de General Electric.
Notation graphique riche et lisible.

Grady Booch

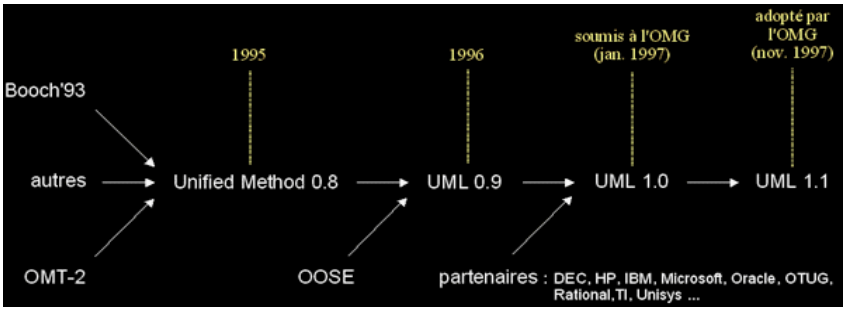
OOD : vues logiques et physiques du système
Définie pour le DOD, afin de rationaliser de développement d'applications ADA, puis C++.
Ne couvre pas la phase d'analyse dans ses 1ères versions (préconise SADT).
Introduit le concept de package (élément d'organisation des modèles).

Ivar Jacobson

OOSE : couvre tout le cycle de développement
Issue d'un centre de développement d'Ericsson, en Suède.
La méthodologie repose sur l'analyse des besoins des utilisateurs.

U M L Les méthodes d'analyse objet

Un langage avec UML



Pas de consensus sur la méthode mais un leadership

Le leader sur le marché de la modélisation est Rational Rose

Mais il y en a d'autres : GDPro, ObjectTeam, Objecteering, OpenTool, Rhapsody, STP, Visio, Visual Modeler, WithClass

Il intègre dans son AGL la méthode d'analyse RUP (Rational, unified Process)

Un méhode de la même famille est la « 2trackup » ou méthode de développement en Y

Le langage UML

Références

www.uml.free.fr

Cours de philippe ramadour IUP MArseille

Sommaire

UML

- Introduction
- Les outils de représentation
  - Diagrammes
  - Vues
- La méthode 2TUP: un processus unifié
- Étude préliminaire
- Les symbolismes de base

Introduction

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

La définition du formalisme UML ne résulte pas d'un processus innovant de recherche mais d'un processus consensuel de stabilisation des pratiques Industrielles éprouvées.

UML n'est que le reflet fidèle des pratiques majoritaires utilisées vers la fin des années 2000 par la profession.

UML ne vise pas l'innovation, mais la consensualité.

Il y a deux façons de réaliser un consensus:
à minima (par intersection) ou à maxima (par union).

Ces deux tendances se retrouvent dans les groupes d'influence qui gèrent l'évolution du langage (vendeurs d'AGL, etc.)

La tendance maximaliste a souvent été majoritaire (!)

- Spécification actuelle : UML 1.5
- Spécification en cours de rédaction : UML 2.0

U M L

Introduction

- **Principes et caractéristiques** :
  - incrémental ⇨ meilleure gestion des risques (techniques et fonctionnels)
  - itératif ⇨ production de traces pour le contrôle du changement
  - centré sur l'architecture
  - conduit par les besoins (cas d'utilisation) ⇨ centré sur les utilisateurs (Nouveauté UML)
  - piloté par les risques
  - création et maintenance de modèles
  - orienté composant ⇨ réutilisation et maintenance accrues

Cours de SI
Bordeaux I

Introduction

Fin 1997, UML est devenu **une norme OMG** (Object Management Group). Cela vous semble certainement sans importance, mais pourtant...

L'OMG est un organisme à but non lucratif, créé en 1989 à l'initiative de grandes sociétés (HP, Sun, Unisys, American Airlines, Philips...). Aujourd'hui, l'OMG fédère plus de 850 acteurs du monde informatique.

Son rôle est de promouvoir des standards qui garantissent l'interopérabilité entre applications orientées objet, développées sur des réseaux hétérogènes.

U M L

Introduction

L'OMG propose notamment l'architecture CORBA (Common Object Request Broker Architecture),

CORBA fait partie d'une vision globale de la construction d'applications réparties appelée **OMA** (Object Management Architecture) et définie par l'OMG.

UML a été adopté (normalisé) par l'OMG et intégré à l'OMA, car il participe à cette vision et parce qu'il répond à la "philosophie" OMG.

Cours de SI
Bordeaux I

Architecture

- L'architecture de modélisation de l'OMG est composée de 4 couches :

Méta-métamodèle *Définit un langage pour spécifier des méta-modèles*
Méta-modèle *Une instance d'un méta-métamodèle* *Définit un langage pour spécifier des modèles*
Modèle *Une instance d'un méta-modèle* *Définit un langage pour spécifier des applications*
Application *Une instance d'un modèle* *Définit les informations particulières de l'application*

Architecture

- Illustration de cette architecture en 4 couches :

Méta-métamodèle *Méta-Classe, Méta-Attribut, Méta-Opération*
Méta-modèle *Classe, Attribut, Opération*
Modèle *Compte, nomDuClient, Transfert*
Application *Compte_324, Dupont, Transfert(Compte_324, Compte_876)*

U M L Introduction

- UML permet de modéliser des systèmes
  - logiciels (BD, télécoms, …)
  - d'information (au sens large)
- Ses objectifs sont :
  - documenter (les besoins, les architectures, la conception, le code, …)
  - visualiser
  - spécifier
  - Construire
- Description d'un système selon une architecture trois-tiers :
  - **objets de l'interface,**
  - **objets du serveur d'application,**
  - **objets du serveur de données**

UML

Les outils de représentation

L'objectif d'UML est de produire un/des Modèle(s)

Un modèle est une abstraction de la réalité

L'abstraction est 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.

Quelques exemples de modèles

Modèle météorologique
Modèle économique
Modèle démographique, …

Modèle(s)

Modèles UML

- ***Au niveau de la capture des besoins :*** le modèle trace les frontières fonctionnelles du système.
- ***Au niveau de l'analyse :*** le modèle est une abstraction des concepts manipulés par les utilisateurs au point de vue statique et dynamique.
- ***Au niveau de la conception :*** le modèle représente les concepts utilisés par les outils, les langages ou les plates-formes.
- ***Au niveau du déploiement :*** le modèle représente les matériels et les logiciels à interconnecter.

Modèle(s)

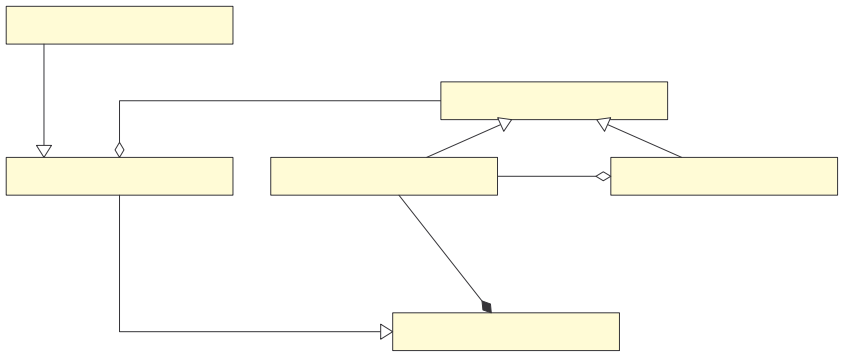
- UML permet de produire un ensemble de modèles dont la définition n'est pas figée :
  - modèle des cas d'utilisation
  - modèle d'analyse
  - modèle de conception
  - modèle de déploiement
  - modèle d'implantation
  - modèle de test

Construction de modèles

- Les éléments d'un modèle sont regroupés en paquetages.
- Un modèle est donc organisé sous la forme d'une arborescence de paquetages.

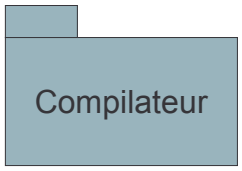
Modèles UML

Construction de modèles



Modèle

Packages



Un *package* est un conteneur pour organiser les autres éléments en groupes.

Un package peut contenir des classes, d'autres packages ou des diagrammes.

Les Packages peuvent être utilisés pour fournir un accès contrôlé entre classes de différents packages.

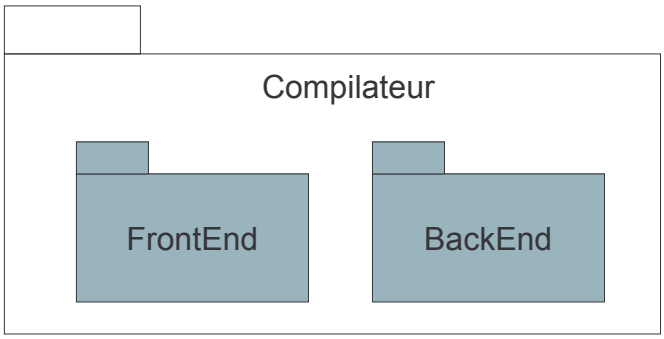
Référence

Elément de

P

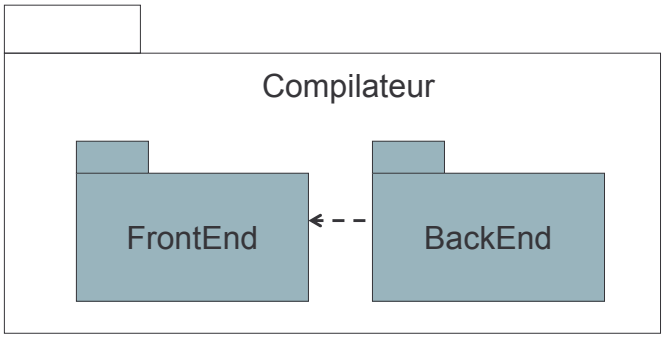
Packages

Les classes dans le package *FrontEnd* et les classes dans le package *BackEnd* n'accèdent pas l'une à l'autre.



Packages

Les classes de *BackEnd* ont accès aux classes de *FrontEnd*.



Modèles UML

Construction de modèles

- Chaque élément de modélisation possède une spécification qui contient la description unique et détaillée de toutes les caractéristiques de l'élément.
- Il existe des mécanismes communs à tous les modèles.

➢**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	

Diagrammes

- Les 9 types de diagrammes sont proposés par UML suivant deux modes de représentation :
  - statique : structure du système "au repos«
  - dynamique : système "en fonctionnement"

Diagrammes

- Représentation statique :
  - ***diagrammes de cas d'utilisation :*** structure des fonctionnalités
  - ***diagrammes de classes :*** structure des entités
  - ***diagrammes d'objets :*** illustration des structures de classes
  - ***diagrammes de composants :*** structure des composants d'exploitation et concepts de configuration logicielle
  - ***diagrammes de déploiement :*** structure du réseau et insertion des composants dans cette structure

Diagrammes

- Représentation dynamique :
  - ***diagrammes d'états :*** cycle de vie des objets
  - ***diagrammes d'activités :*** règles d'enchaînement des activités dans le système
  - ***diagrammes d'interaction*** : échange de messages entre objets
    - ***diagrammes de collaboration :*** modélisation du contexte du système, conception de méthodes
    - ***diagrammes de séquence :*** scénarii d'utilisation

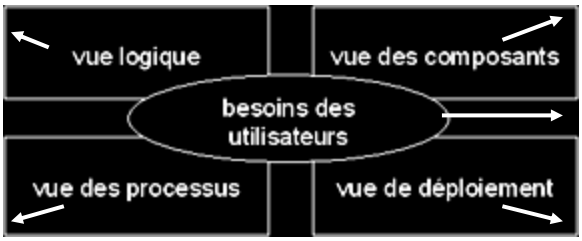
Diagrammes

- Les diagrammes sont composés :
  - de nœuds (éléments de base)
  - d'arcs (relations)
- Un même diagramme peut être présenté à différents niveaux de détails (imbrication de diagrammes).

Vues

Ph. Kruchten, un des pères de la méthode RUP,propose différentes perspectives, indépendantes et complémentaires, qui permettent de définir un modèle d'architecture (publication IEEE, 1995).

Comment découper le système pour le développer ?



Quelles sont les briques logicielles nécessaires ?

Qu'est-ce qui est prioritaire pour l'utilisateur ?

Comment gère-t-on les threads ?

Comment déploie-t-on l'apllication? Quels sont les contraintes ?

Les vues sont faites pour préciser l'architecture applicative en la considérant de plusieurs point de vue.

Il n'y a pas de symbolisme propre à une vue. Pour décrire un point de vue on utilise simplement les diagrammes disponibles.

Vues

Vue logique

Cette vue de haut niveau se concentre sur l'abstraction et l'encapsulation, elle modélise les éléments et mécanismes principaux du système.

Elle identifie les éléments du domaine, ainsi que les relations et interactions entre ces éléments :

- les éléments du domaine sont liés au(x) métier(s) de l'entreprise,
- ils sont indispensables à la mission du système,
- ils gagnent à être réutilisés (ils représentent un savoir-faire).

Cette vue organise aussi (selon des critères purement logiques), les éléments du domaine en ***"catégories"*** :

- pour répartir les tâches dans les équipes,
- regrouper ce qui peut être générique,
- isoler ce qui est propre à une version donnée, etc...

Vues

Vue des composants

Cette vue de bas niveau (aussi appelée "vue de réalisation"), montre :

L'allocation des éléments de modélisation dans des modules (fichiers sources, bibliothèques dynamiques, bases de données, exécutables, etc...).

En d'autres termes, cette vue identifie les modules qui réalisent (physiquement) les classes de la vue logique.

L'organisation des composants, c'est-à-dire la distribution du code en gestion de configuration, les dépendances entre les composants...

Les contraintes de développement (bibliothèques externes...).

La vue des composants montre aussi l'organisation des modules en ***"sous-systèmes"***, les **interfaces** des sous-systèmes et leurs dépendances (avec d'autres sous-systèmes ou modules).

Vues

Vue des processus

Cette vue est très importante dans les environnements **multitâches**
elle montre :

La décomposition du système en terme de processus (tâches).

Les interactions entre les processus (leur communication).

La synchronisation et la communication des activités parallèles (threads).

Vues

Vue de déploiement

Cette vue est très importante dans les environnements **distribués**

Elle décrit les ressources matérielles et la répartition du logiciel dans ces ressources :

La disposition et nature physique des matériels, ainsi que leurs performances.

L'implantation des modules principaux sur les noeuds du réseau.

Les exigences en terme de performances (temps de réponse, tolérance aux fautes et pannes...).

Vues

Vue des besoin utilisateur

Cette vue (dont le nom exact est "vue des cas d'utilisation"), **guide toutes les autres.**

Dessiner le plan (l'architecture) d'un système informatique n'est pas suffisant, il faut le justifier !

Cette vue définit les besoins des clients du système et centre la définition de l'architecture du système sur la satisfaction (la réalisation) de ces besoins.

A l'aide de scénarios et de cas d'utilisation, cette vue conduit à la définition d'un modèle d'architecture pertinent et cohérent.

Cette vue est la "colle" qui unifie les quatre autres vues de l'architecture.

Elle motive les choix, permet d'identifier les interfaces **critiques** et force à se concentrer sur les problèmes importants.

U M L Notions de base

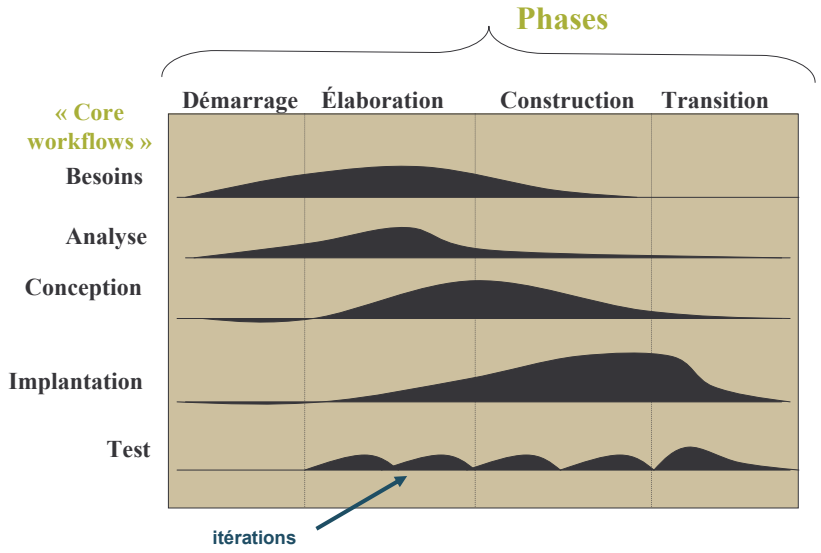
Vue / Diagramme	Use case	Logique	composants	processus	déploiement
Cas d'utilisation	Acteurs Use case				
Objets	Acteurs Objets liens	Acteurs, Classes Objet, liens			
collaboration	Acteurs, Objets Message, liens	Acteurs, Classes Objet, liens		Classes Objets , liens	
Séquence	Acteurs Objets message	Acteurs 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

UML

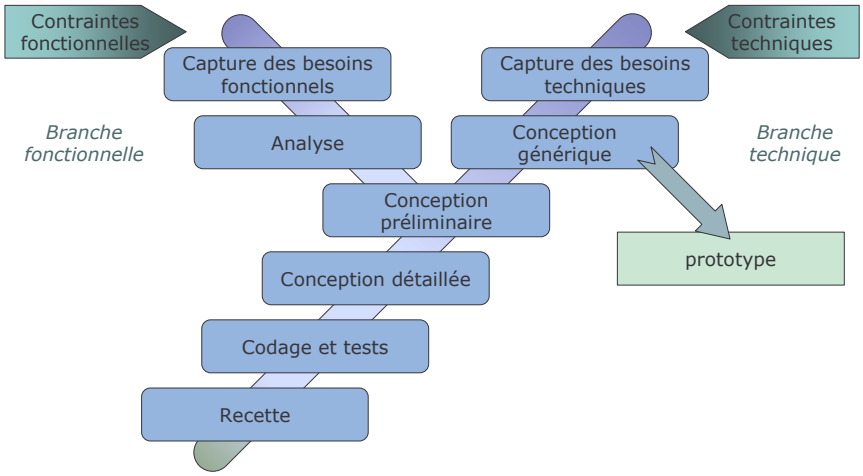
La méthode d'analyse

- 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

UML

Etude préliminaire

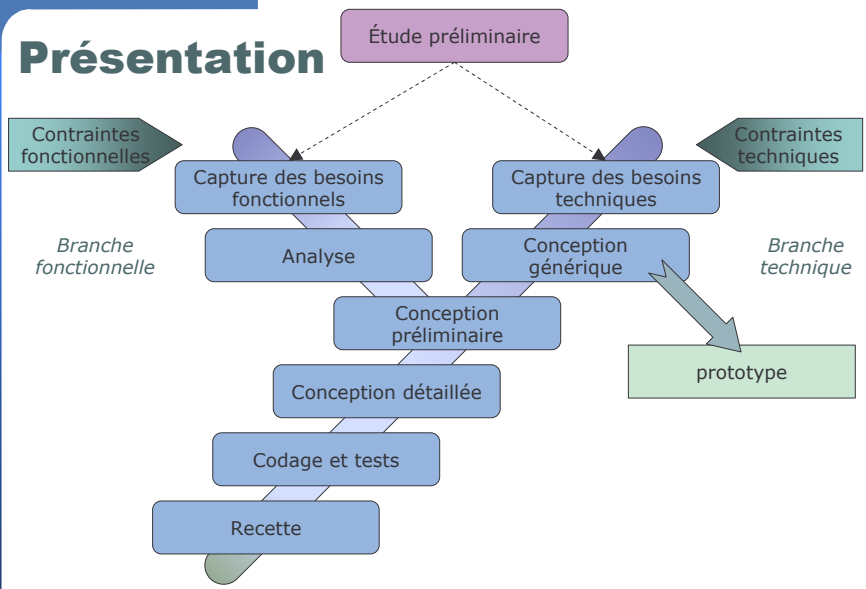
Cours de SI
Bordeaux I 2004

Présentation

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

Cours de SI
Bordeaux I

Présentation



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,
  - …

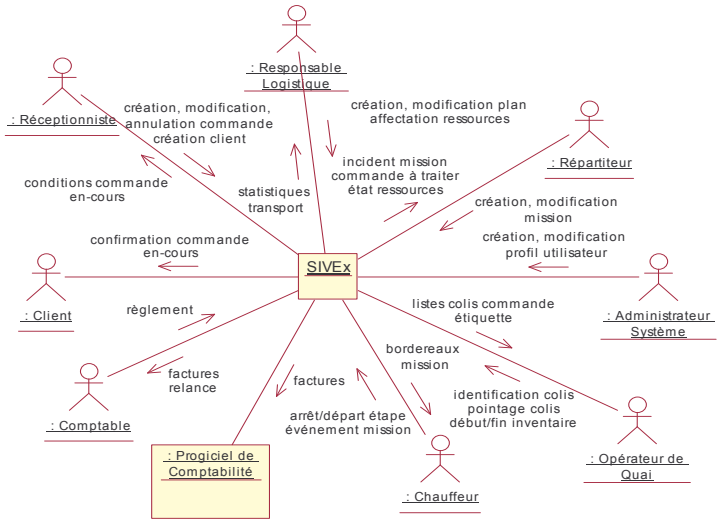
Messages du système SIVEx

- 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,
  - …

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.

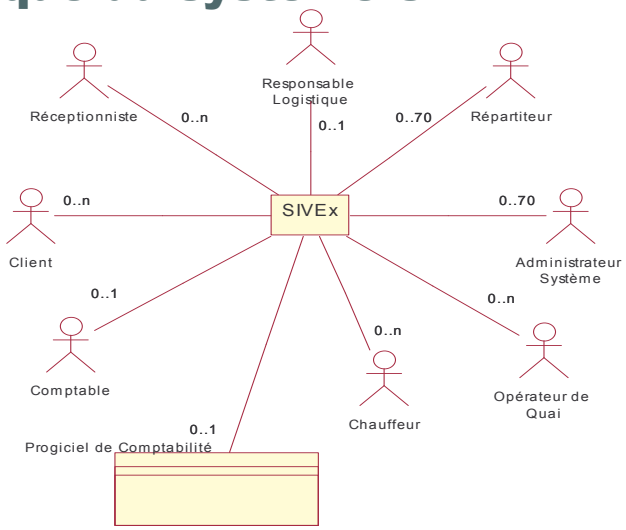
Modéliser le contexte statique

- 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.

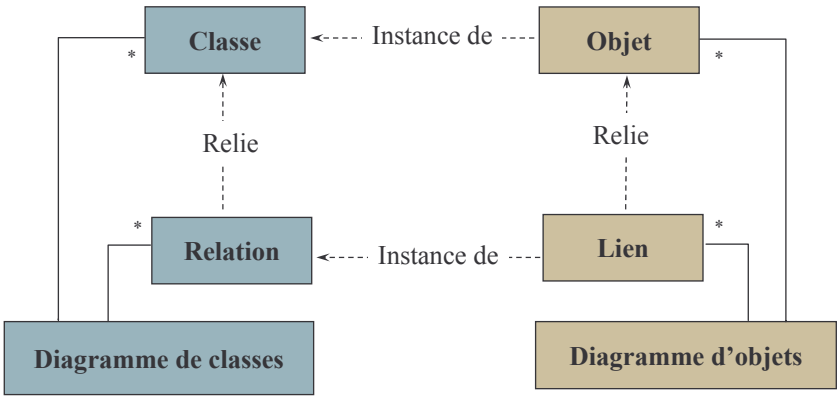
Conclusion

- 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

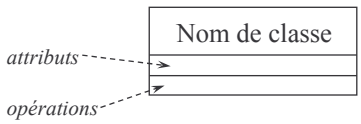
Classes et Objets



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

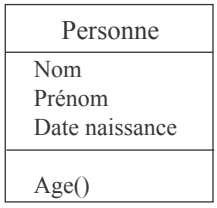
Nom du paquetage::Nom de classe

Exemples :

Personne

Poste de travail

Département

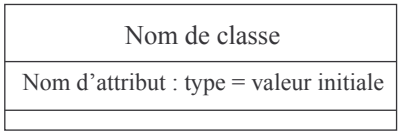


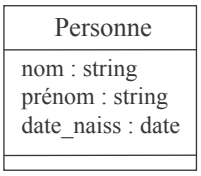
Personne
{auteur = Jean,
état = vérifié,
par = Julie}

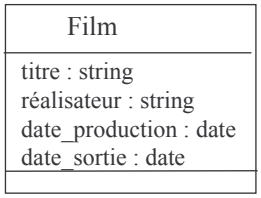
Une classe avec des propriétés

Attributs d'une classe

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

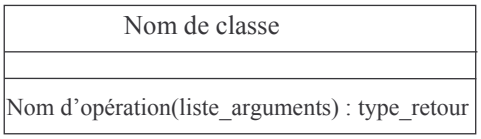


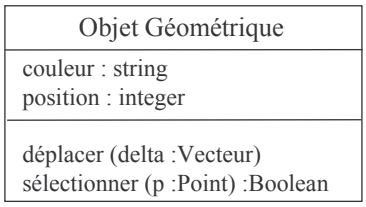




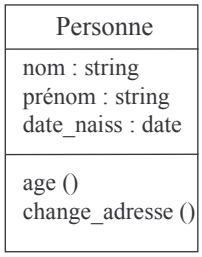
Opérations d'une classe

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





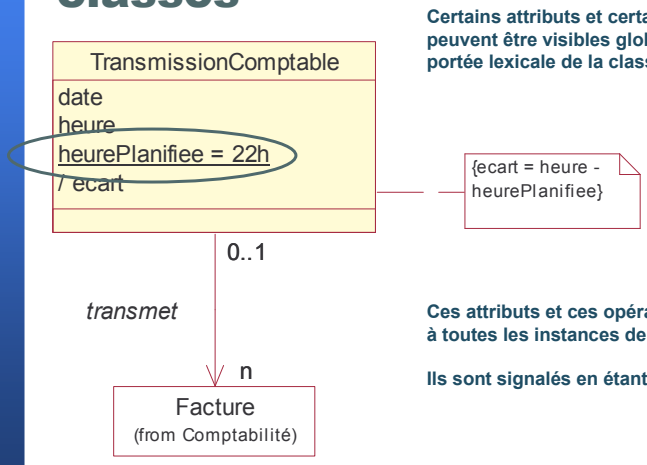
Opérations avec leur signature



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

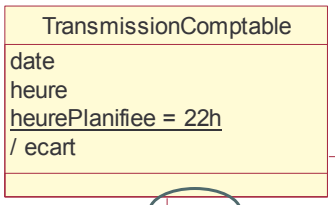


Certains attributs et certaines opérations peuvent être visibles globalement, dans toute la portée lexicale de la classe.

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

Ils sont signalés en étant soulignés.

Multiplicité



{ecart = heure - heurePlanifiee}

0..1

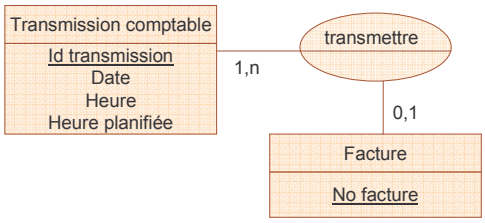
transmet

n

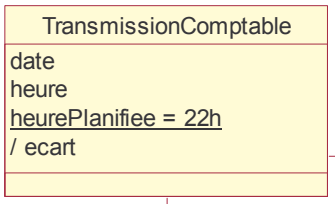
Facture
(from Comptabilité)

Se lit :
1 transmission comptable transmet n factures

ATTENTION : Avec Merise on obtiendrait



Notes et étiquettes



{ecart = heure - heurePlanifiee}

0..1

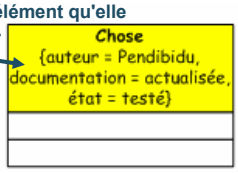
transmet

n

Facture
(from Comptabilité)

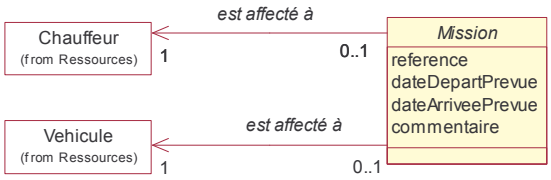
Note
commentaire attaché à un ou plusieurs éléments de modélisation
par défaut, une note ne véhicule pas de contenu sémantique

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}



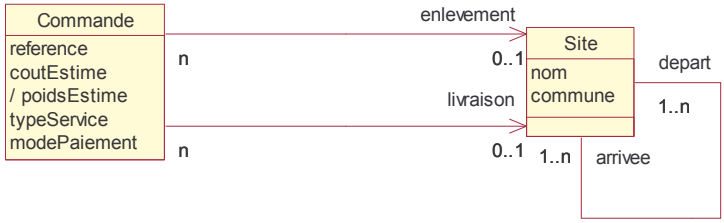
Associations

- 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

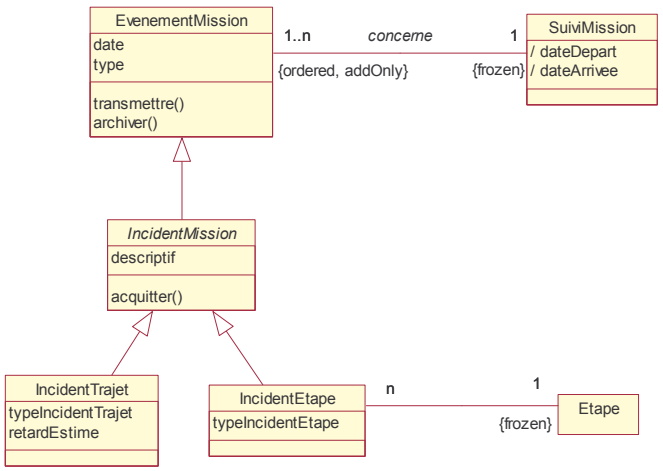
- Chaque association binaire possède 2 rôles.
- Le rôle décrit comment une classe voit une autre classe au travers d'une association.
- Le nommage des associations et le nommage des rôles ne sont pas exclusifs l'un de l'autre.



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



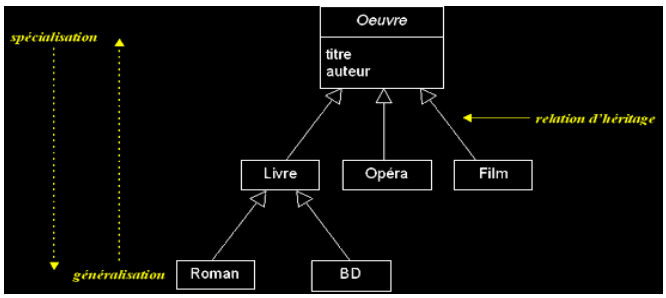
Formalisation de l'héritage

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.

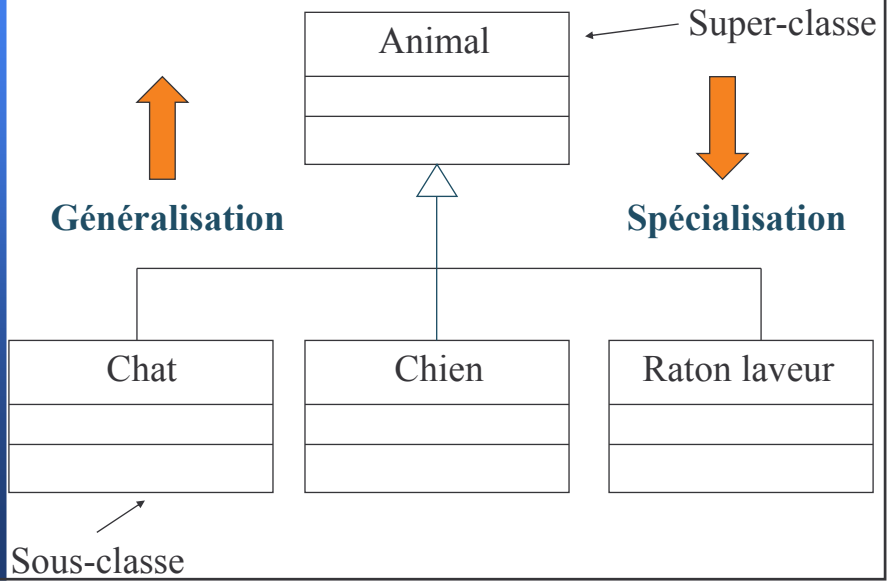
La spécialisation et la généralisation permettent de construire des hiérarchies de classes.
L'héritage peut être simple ou multiple.
L'héritage évite la duplication et encourage la réutilisation.



U M L Symbolismes de base

Généralisation simple

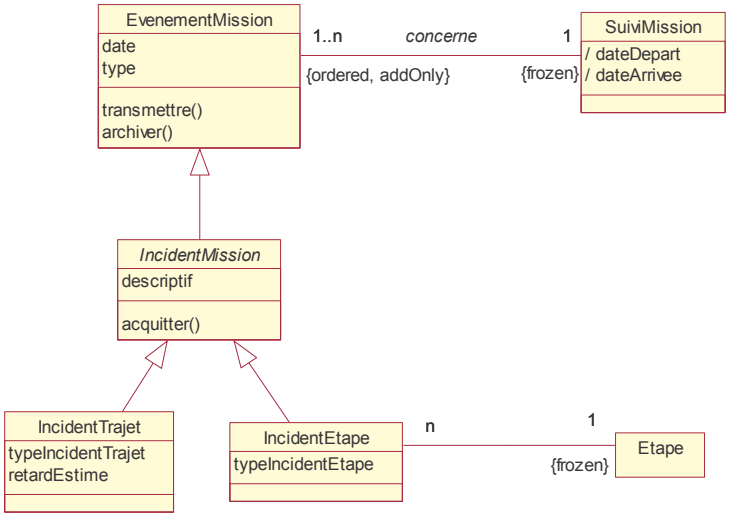
Formalisation de l'héritage



Cours de SI
Bordeaux I

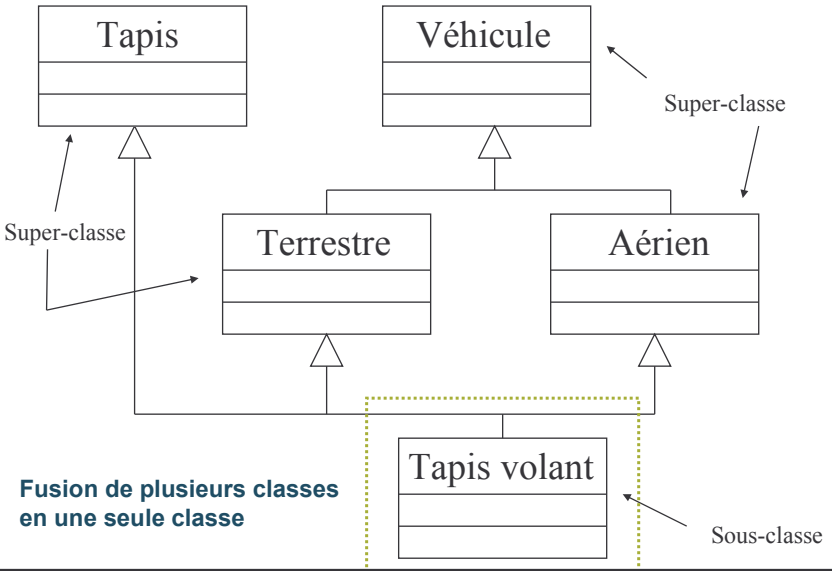
Généralisation simple

Formalisation de l'héritage



Généralisation multiple

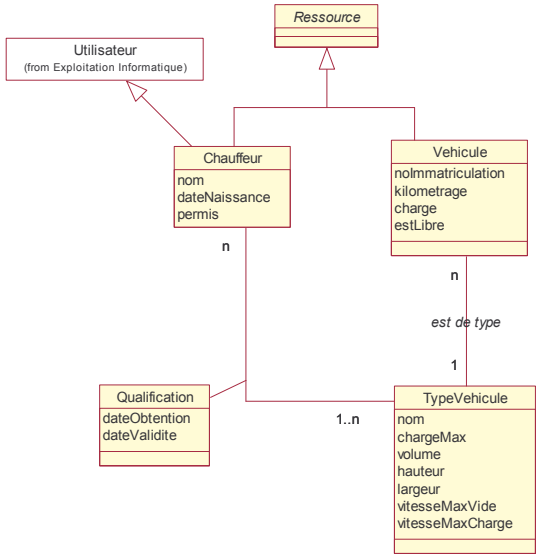
Formalisation de l'héritage



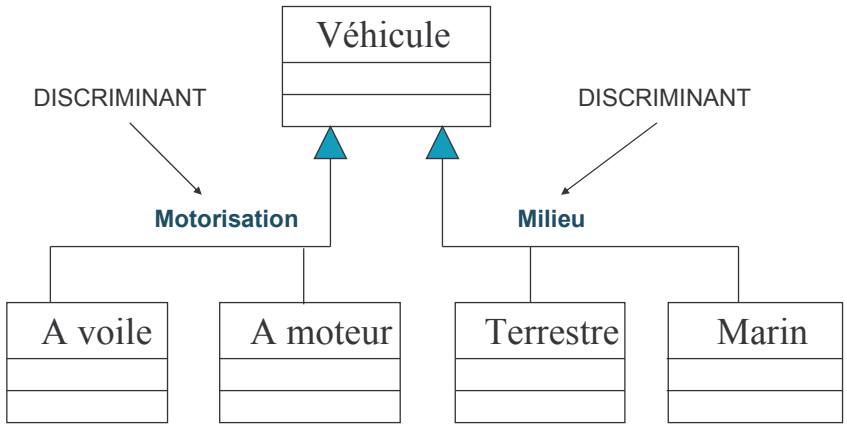
Fusion de plusieurs classes en une seule classe

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*** ».

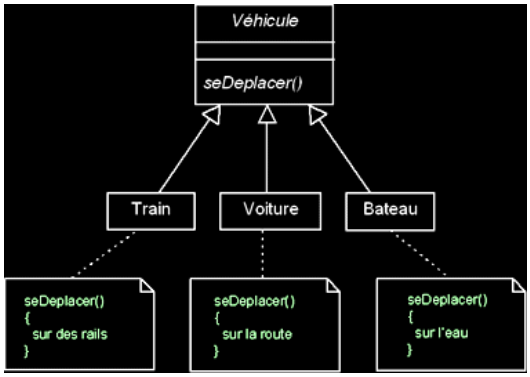
Formalisation du polymorphisme

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.

Polymorphisme, exemple :

```
Vehicule convoi[3] = {
  Train("TGV"),
  Voiture("twingo"),
  Bateau("Titanic")
};

for (int i = 0; i < 3; i++)
{
  convoi[i].seDeplacer();
}
```



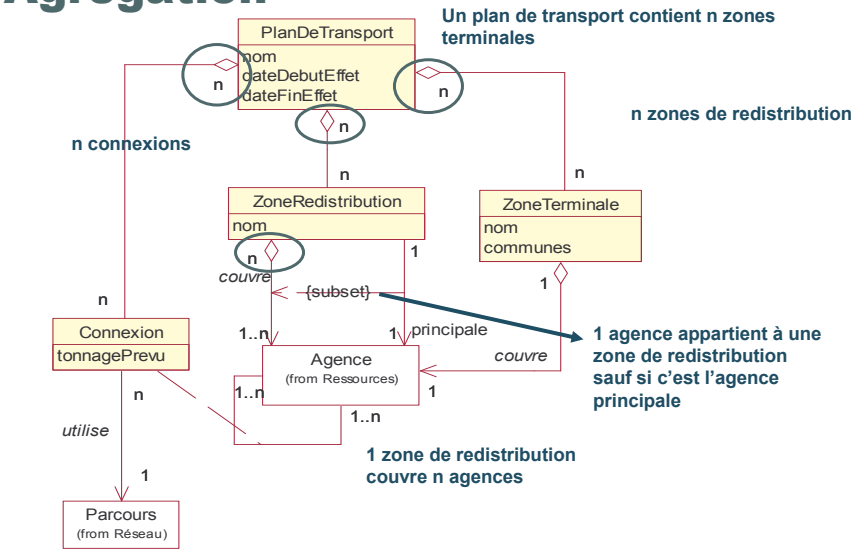
Agrégation

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.

U M L Symbolismes de base

Agrégation



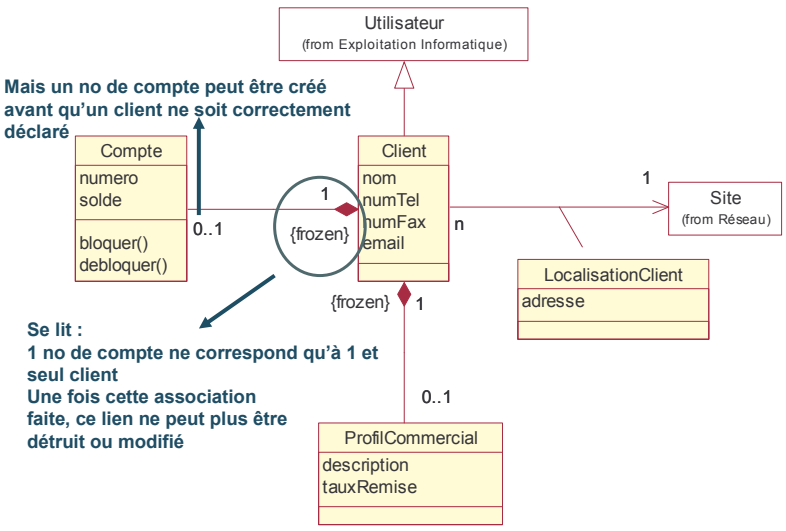
Cours de SI
Bordeaux I

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.

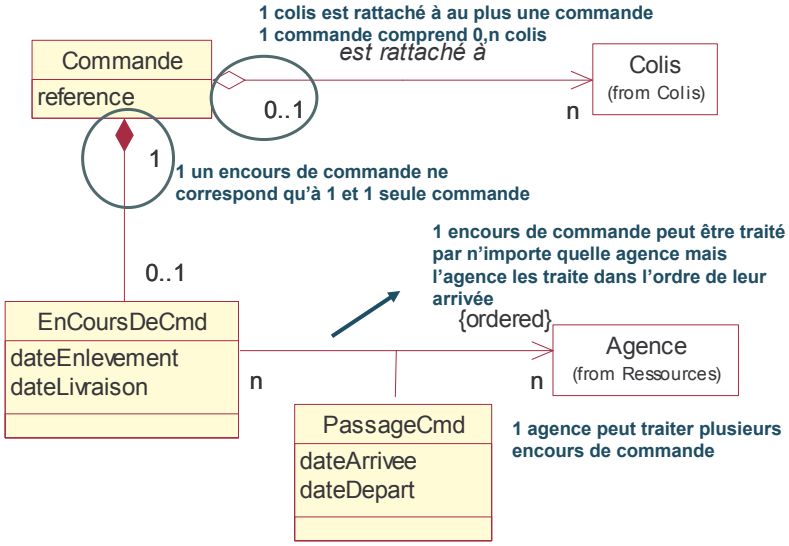
U M L Symbolismes de base

Composition



Cours de SI
Bordeaux I

Agrégation / composition

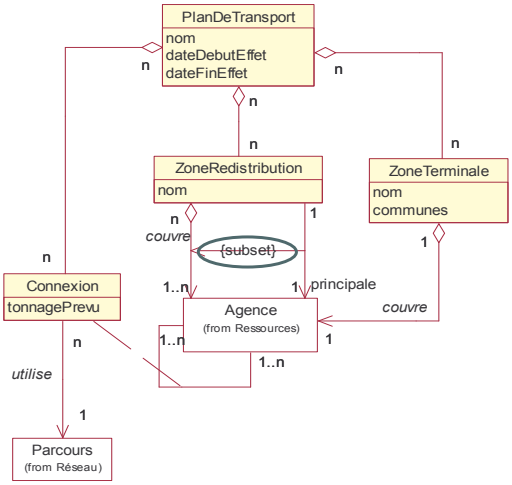


Contraintes

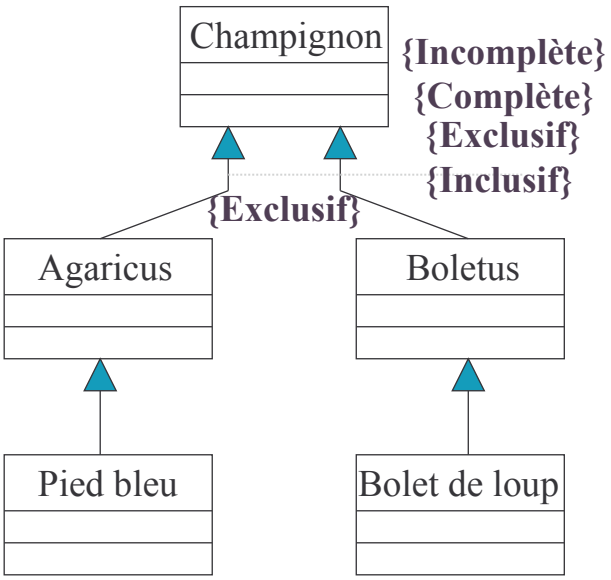
- relation sémantique (quelconque) entre éléments de modélisation
- *exemple :* `{incomplet}` sur une généralisation

Contraintes entre associations

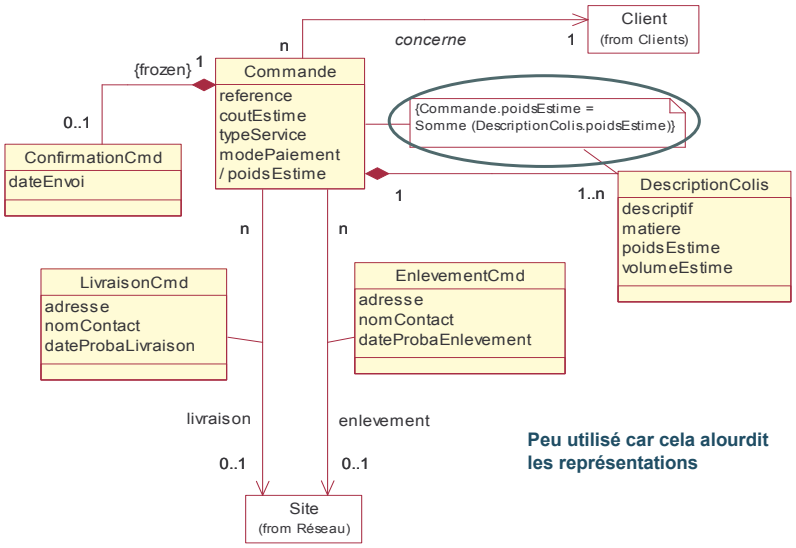
- UML définit notamment les contraintes « {subset} » et « {XOR} ».



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