

Modélisation et Programmation 'Orienté Objet' De UML à C++

Jean-Luc Charles

jlc@lamef.bordeaux.ensam.fr

Laboratoire Matériaux Endommagement Fiabilité et Ingénierie des Procédés
ENSAM - Université de Bordeaux I

Modélisation et Programmation 'Orienté Objet' De UML à C++

Épisode I L'Approche "Orienté Objet" (Analyse et Conception)

Épisode II Modélisation "Orienté Objet" avec UML

Épisode III Le langage C++ - Les bases

Épisode IV Le langage C++ - Objet

Plan

Épisode I

Épisode II

Bibliographie

- 1 **Épisode I : L'Approche "Orienté Objet"**
 - Différentes Approches d'Analyse et de Conception
 - L'Approche Fonctionnelle
 - L'Approche Orientée Objet
 - Le concept d'Objet
 - Le concept de Classe
 - Les relations entre Classes
 - Le Polymorphisme
 - La Généricité

- 2 **Épisode II : Modélisation "Orienté Objet" avec UML**
 - Modélisation Objet UML
 - Diagramme de Classes
 - Diagramme d'Objets
 - Diagramme des cas d'utilisation
 - vers les langages OO

- 3 **Bibliographie**

v1.1 du 26/04/2007

3 / 78

Épisode I : L'Approche "Orienté Objet"

Épisode I

Différentes Approches

Approche Fonctionnelle

Approche Orientée
Objet

Concept d'Objet

La Classe

Les relations

Polymorphisme

Généricité

Épisode II

Bibliographie

" S'il n'y a pas de solution, c'est qu'il n'y a pas de problème."
– Logique Shaddock

Les 3 phase (idéales) dans la conduite d'un projet (informatique) :

- **Analyse** "QUOI faire?" (investigation du problème posé et des besoins : construire le bon Système)
- **Conception** -> "COMMENT faire?" (identifier les concepts de la solution : bien construire le Système)
- **Réalisation** : "réaliser" la solution (informatique : programmer).

✓ Plusieurs approches sont possibles pour chacune de ces phases en fonction de la nature du problème traité (organisationnel, informatique, production, ...).

✓ Par exemple pour les phases d'Analyse et de Conception :

- L' **Approche Fonctionnelle** et les méthodes associées
 - Analyse Fonctionnelle (AF), SADT (Analyse Fonctionnelle descendante)
 - APTE (APplication aux Techniques d'Entreprise)
 - FAST (Function Analysis System Technique) ...
- L' **Approche 'Orienté Objet'** et les méthodes associées
 - RUP (Rational Unified Process), XP (Extreme Programming) ...
 - techniques de conception (design pattern, programmation par contrat, ...)

v1.1 du 26/04/2007

4 / 78

I - L'Approche Fonctionnelle (années 1960-80)

Épisode I

Différentes Approches

Approche Fonctionnelle

Approche Orientée
Objet

Concept d'Objet

La Classe

Les relations

Polymorphisme

Généricité

Épisode II

Bibliographie

Concepts clef

- Les **fonctions** du Système (objet matériel, programme, service, ...)
 - identification des fonctions du Système ...
 - découpages en sous-fonctions ...
- Analyse Fonctionnelle : Fonctions Principales, Fonctions Contraintes ...

Inconvénients :

- Modification donnée => impacts multiples (tous les blocs fonctionnels utilisant la donnée)
- Évolutivité et maintenabilité parfois problématique
- Informatique :
 - données séparées des traitements
 - peu ou pas de niveau d'abstraction des données.

Donne de "bons résultats" pour des problèmes aux fonctions bien identifiées, stables dans le temps (cahier des charges fixé, ne bouge pas).

I - L'Approche 'Orienté Objet' (années 1980-90)

Épisode I

Différentes Approches

Approche Fonctionnelle

Approche Orientée
Objet

Concept d'Objet

La Classe

Les relations

Polymorphisme

Généricité

Épisode II

Bibliographie

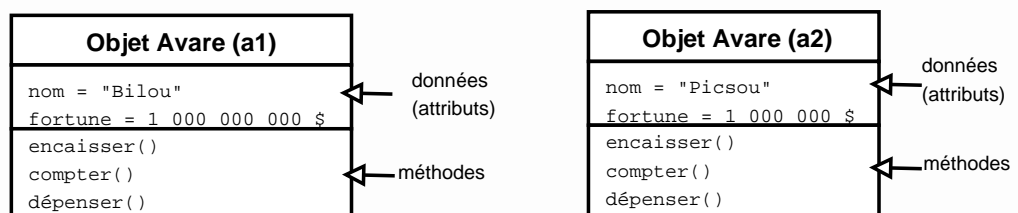
Idée de l'approche objet :

S'inspirer du monde réel ... fait d'**Objets**, **communiquant** entre eux, possédant :

- des **propriétés**, intrinsèques ou dépendant d'éléments extérieurs
- des **comportements**, intrinsèques ou dépendant d'éléments extérieurs.

Un Objet =

- des **DONNÉES** (propriétés)
- des **MÉTHODES** manipulant ces données (comportements)
- des échanges de **MESSAGES** avec d'autres objets.



Deux objets de type Avare

I - L'Approche 'Orienté Objet'

Épisode I

Différentes Approches
Approche Fonctionnelle
**Approche Orientée
Objet**
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Analyse/Conception "Orienté Objet" d'un Système :

Le comportement global du Système repose sur :

- l'ensemble des objets identifiés
- les relations et les communications possibles entre ces objets.

⇒ L'aspect dynamique du Système correspond à des envois de messages entre objets qui déclenchent des traitements divers ...

Système vu par l'approche 'Orienté Objet' :

Ensemble d'objets **interagissant entre eux** pour réaliser les **fonctions** du Système

⇒ L'approche "Orienté Objet" est applicable pour l'Analyse, la Conception et la Programmation !

I - L'Approche 'Orienté Objet'

Épisode I

Différentes Approches
Approche Fonctionnelle
**Approche Orientée
Objet**
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Les paradigmes de l'Orienté Objet :

- **Objet**
- **Classe**
- **Encapsulation**
- **Héritage**
- **Polymorphisme**
- **Généricité**

I - Le concept d'Objet

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

L'Objet

C'est une **unité atomique** possédant :

- une **identité**
- un **état**, défini par un ensemble de données (attributs, ou données membres)
- un **comportement**, défini par un ensemble de fonctions (méthodes, ou fonctions membres).

Un Objet peut correspondre à :

- Un **objet concret** du monde réel, ayant une réalité physique (une personne, une voiture, un outil, un système mécanique, ...)
- Un **concept abstrait** (un compte bancaire, une contrainte mécanique, une vitesse, ...)
- Une **activité** produisant des effets observables (un calcul numérique, un pilote d'impression, ...)

I - Le concept d'Objet

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Le principe d'**Encapsulation** permet aux objets de se présenter sous deux vues possibles :

la vue externe (celle de l'*utilisateur* de l'objet)

- Définit l'**interface** de l'objet (comment on l'utilise)
- Fournit la liste des services accessibles aux utilisateurs de l'objet
- Ne fournit aucun accès aux mécanismes internes de l'objet (données et méthodes)
- UML/C++ : correspond aux déclarations qualifiées **public**.

la vue interne (celle du *concepteur* de l'objet)

- Donne les détails de constitution interne de l'objet (comment il est construit)
- UML/C++ : correspond aux déclarations qualifiées **protected** ou **private**.

⇒ L'encapsulation (des attributs et des méthodes) permet de dissimuler à l'utilisateur d'un objet des détails susceptibles d'évoluer avec le temps, ou ne présentant pas d'intérêt pour l'utilisation externe de l'objet.

⇒ L'interface proposée par la vue externe est la seule partie de l'objet qui **DOIT** être **pérenne** dans le temps, tout en proposant suffisamment d'interaction avec l'objet.

I - Le concept d'Objet

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Persistence

C'est la possibilité de stocker les valeurs des attributs d'un objet sur un support externe pour pouvoir les re-charger ultérieurement

⇒ les objets deviennent immortels !

Un mécanisme de persistance offre une **fermeture persistante** lorsqu'il propose de sauvegarder automatiquement un objet et toutes ses dépendances (les autres objets auxquels il est lié).

I - Le concept d'Objet

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

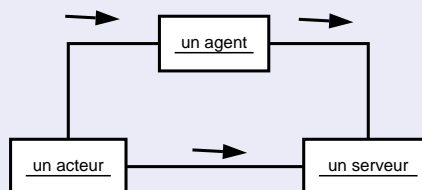
Épisode II

Bibliographie

Objets et Messages

Dans les échanges des messages entre objets, on distingue :

- **Acteur** : objet actif, à l'origine de l'envoi de messages
- **Serveur** : objet passif destinataire des messages (jamais à l'origine d'un échange)
- **Agent** : objet à la fois Acteur et Serveur peut interagir avec les autres objets, de sa propre initiative ou suite à un message.



I - Le concept d'Objet

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

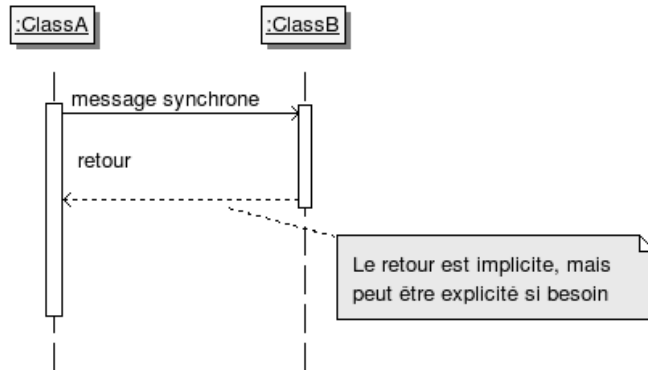
Épisode II

Bibliographie

Message Synchrone

Lorsqu'un objet A envoie un **message synchrone** à l'objet B :

- A attend que B accepte de prendre le message
- Une fois le message accepté par B, A est "bloqué" tant que B n'a pas fini le traitement correspondant.



I - Le concept d'Objet

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

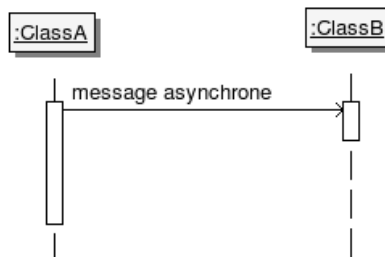
Épisode II

Bibliographie

Message Asynchrone

Un objet A envoie un **message asynchrone** à l'objet B :

- A n'attend pas que B accepte le message
- A n'attend pas que B ait fini le traitement correspondant
- A passe directement à autre chose (sans savoir si B a accepté le message, ni si B a fini le traitement correspondant !).



I - Le concept d'Objet

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Récapitulons :

- Un **Objet** possède une **identité**
- Un ensemble d'**attributs** (données) caractérise l'**état** de l'objet
- Un ensemble d'**opérations** (méthodes) définit le **comportement** de l'objet
- L'**encapsulation** permet aux objets de fournir une interface aux autres objets, tout en dissimulant ses détails internes
- Un objet peut être en **relation** avec d'autre(s) objet(s)
- Un objet peut échanger des **messages** avec d'autre(s) objet(s)
- Un objet possède un **cycle de vie** (construction ... destruction)
- Un objet peut être **persistant**.

I - Le concept de Classe

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

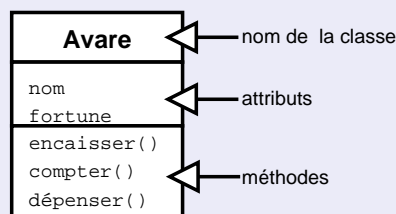
Épisode II

Bibliographie

Classe

La Classe est un **modèle** représentant une famille d'objets ayant :

- la même **structure** de données (même liste d'attributs)
- les mêmes **méthodes**
- les mêmes **relations**.



**La Classe par elle-même ne contient pas les valeurs des données :
c'est un type de données abstrait.**

- ✓ La notion de **Classe** est fortement liée à la notion de **type** des langages informatiques. En C++ :
- un nom de classe devient un nouveau type du langage
 - les opérateurs du langage (*, +, /, ++, -, ...) peuvent être re-définis (mécanisme de **Surcharge**) pour s'appliquer aux nouveaux types définis par les classes.

I - Le concept de Classe

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

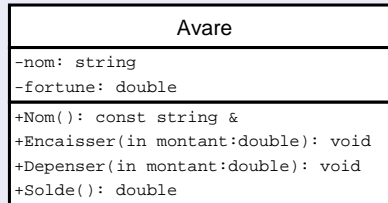
Bibliographie

Encapsulation

La déclaration d'une classe permet de préciser complètement son **interface** :

- la partie **publique**, utilisable par n'importe quelle entité
- la partie **privée**, utilisable uniquement au sein de la classe
- la partie **héritée**, utilisable par la classe ou des classe dérivées.

Notation UML :



- attribut ou méthode privée (**private**)
- # attribut ou méthode protégée (**protected**)
- + attribut ou méthode (**public**).

I - Le concept de Classe

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Intérêts de l'encapsulation :

⇒ Facilite l'**évolution** du modèle :

- on peut modifier des détails internes d'une classes (nombre, nature et noms des attributs, détails des traitements) sans modifier son interface.
- la façon de s'en servir ne change pas pour les utilisateurs (modèles, programmes, ...) de la classe.

⇒ Favorise l'**intégrité des données** :

- permet d'interdire l'accès direct aux détails internes
- permet de contrôler et garantir la cohérence des données de la classe en imposant l'utilisation de méthodes publiques d'accès aux données de la classe (lecture ou écriture).

I - Le concept de Classe

Épisode I

Différentes Approches

Approche Fonctionnelle

Approche Orientée
Objet

Concept d'Objet

La Classe

Les relations

Polymorphisme

Généricité

Épisode II

Bibliographie

Exemple de déclaration de la classe Avare en C++ :

```
class Avare
{
public:
// Tout ce qui est public est accessible à tout le monde
    const string& Nom();
    void Encaisser(double somme);
    void Depenser(double somme);
    double Solde();

protected:
// Tout ce qui est protected est transmis par héritage

private:
// Tout ce qui est private est interne à la classe
    double fortune;
    string nom;
};
```

I - Le concept de Classe

Épisode I

Différentes Approches

Approche Fonctionnelle

Approche Orientée
Objet

Concept d'Objet

La Classe

Les relations

Polymorphisme

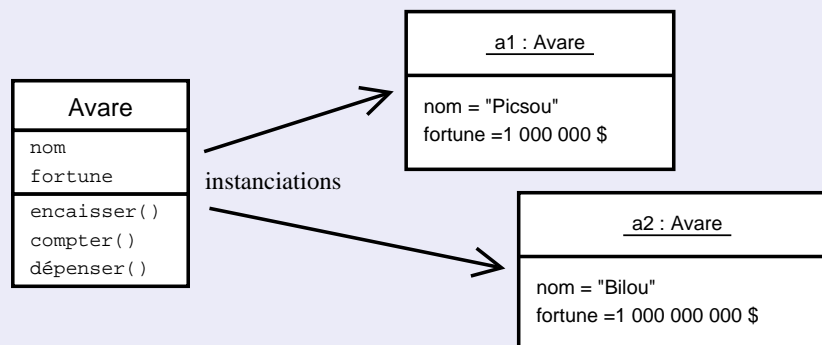
Généricité

Épisode II

Bibliographie

L'Objet comme Instance d'une Classe

- La création d'un **objet** en tant qu'exemplaire concret (contenant des données) d'une classe s'appelle une **INSTANCIATION**.
- Chaque objet (instance d'une classe) donne des valeurs aux attribut de la classe.



I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

L'analyse des Systèmes (réels ou conceptuels) se fait souvent selon deux approches :

- la **Classification** (hiérarchique)
- la **(Dé)Composition** (structurale).

"un pommier *est un* arbre *composé* d'un tronc, de branches, de feuilles et de fleurs"
"un élément triangle T6 *est un* élément fini *composé* de 6 noeuds"

✓ L'approche OO propose plusieurs type de relations entre classes :

- l'**Héritage** (traduit souvent une classification hiérarchique)
- l'**Aggrégation**, la **Composition** (traduit qu'un contenant contient des agrégats/composants : décomposition structurale)
- l'**Association** simple, permettant à deux classes de se connaître (=> d'échanger des messages)
- l'**Utilisation/Dépendance**, traduit le fait qu'une classe se sert d'une autre.

✓ Les **relations** entre classes modélisent les interactions (les **liens**) entre objets.

I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Héritage

traduit la relation "est un"

Exemples :

"Un roman est un livre"
"Une voiture est un Véhicule"
"Une MatriceSymétrique est une Matrice"

Principe de substitution de **Liskov** :

- notion de **sous-type** pour les classes dérivées (sous-classes).
- Permet de déterminer si une relation d'héritage est bien employée comme classification

Principe : si la classe S est un sous-type de la classe T, alors on peut substituer des objets de type S à des objets de type T sans altérer les propriétés désirables d'un modèle.

I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Héritage

- L'**héritage** est **LE** mécanisme de transmission des propriétés (attributs et méthodes) d'une classe à une autre
- La **classe de base** (classe mère, parente) transmet toutes ses propriétés **transmissibles** (protected) aux **classes dérivées** (classe fille, enfant)
- La classe dérivée : - ne peut pas accéder aux membres privés de la classe de base
 - possède ses propres attributs et méthodes, que la classe de base ne connaît pas
 - peut redéfinir (améliorer, spécialiser,) les méthodes transmises par la classe de base.
- Tout ce que la classe de base sait faire, la classe dérivée sait le faire ;
Éventuellement elle sait le faire "mieux" ou "différemment".

I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Héritage

- peut être **simple** au **multiple** : une classe peut dériver de une ou plusieurs classes de base
- souvent **arborescent** : une classe dérivée peut à son tour être une classe de base pour une autre dérivation (-> **hiérarchie de classes**)
- n'est pas réflexif : une classe ne peut pas hériter d'elle-même
- est **LE** mécanisme fondamental pour faire évoluer un modèle (un programme).

L'héritage évite la DUPLICATION et encourage la RÉUTILISATION

I - Les relations entre Classes

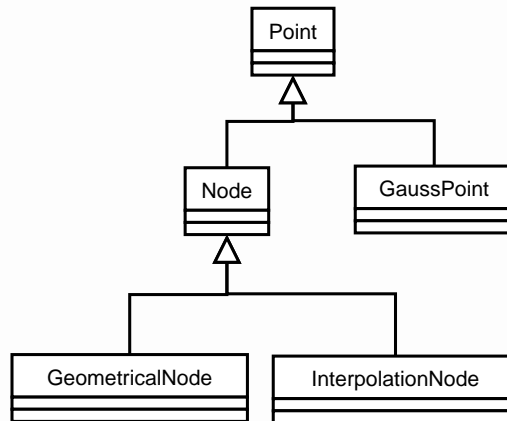
Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Exemple d'héritage simple :



Modèle mécanique de type "éléments finis" :

- un noeud (classe **Node**) "est un" point (classe **Point**)
- un point de Gauss (classe **GaussPoint**) "est un" **Point**
- un noeud géométrique (**GeometricalNode**) *est un* noeud
- un noeud d'interpolation (**InterpolationNode**) *est un* noeud.

I - Les relations entre Classes

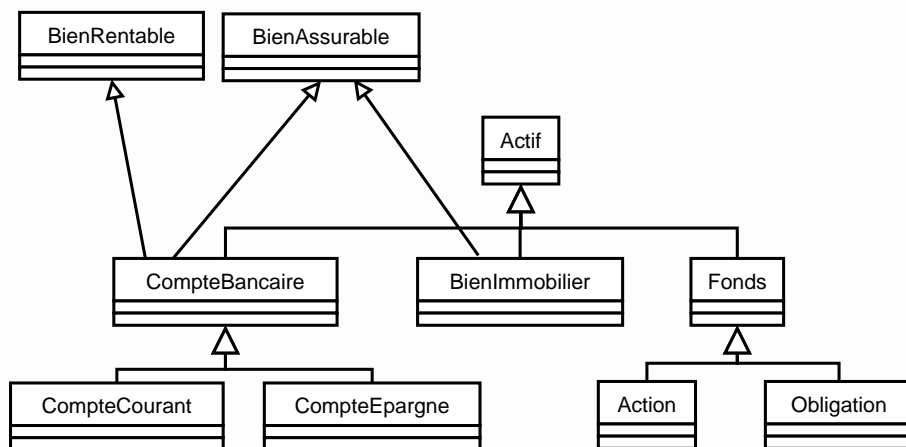
Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Exemple d'héritage multiple :



Un **CompteBancaire** "est un" **BienRentable**,
mais c "est aussi un" **BienAssurable** et "un" **Actif**.

I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Héritage par **spécialisation** (descendant) :

- une classe de base est dérivée pour produire une ou plusieurs classes dérivées **spécialisées** dans certains traitements, ou possédant des attributs particuliers ...
- démarche très utilisée dans les phases initiales de conception car elle permet :
 - de ré-utiliser tout ou partie des attributs et des méthodes d'une classe de base
 - de ne rajouter dans la classe dérivée que les attributs et les méthodes qui la spécialisent.

Héritage par **généralisation** (ascendant) :

- **factorisation** d'un ensemble d'attributs ou de méthodes que l'on regroupe dans une classe de base
- démarche utilisée pour imposer un modèle commun à une hiérarchie de classes
- démarche fréquente en fin de phase de conception (après mûrissement), quand on s'aperçoit que certaines classes possèdent des attributs ou des comportements communs factorisables dans une classe de base.

v1.1 du 26/04/2007 27 / 78

I - Les relations entre Classes

Épisode I

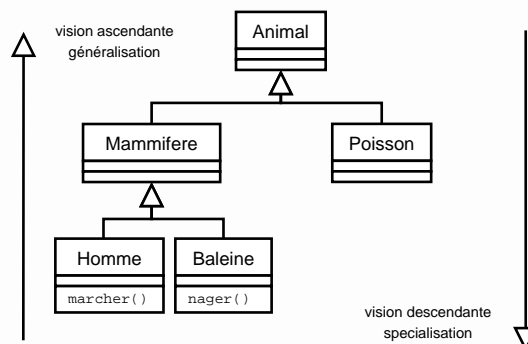
Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Héritage Ascendant :

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



Héritage descendant :

permet d'étendre ou de spécialiser une classe existante, sans la modifier.

v1.1 du 26/04/2007 28 / 78

I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

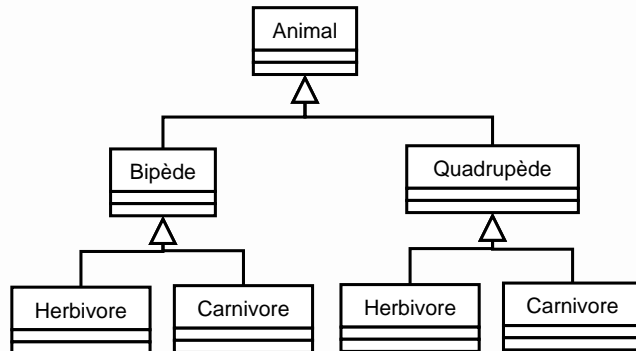
✓ Classifier par hiérarchie n'est pas toujours une opération triviale et peut présenter des pièges.

Exemple : classification des animaux selon plusieurs critères tels que :

- la station (bipède, quadrupèdes),
- le régime (type de nourriture),
- la protection (à plume, à écaille, ...).

✓ La détermination des critères pertinents et de leur ordre peut conduire à des casse-têtes ...

Exemple de classification utilisant le critère 'station' avant le critère 'régime'



I - Les relations entre Classes

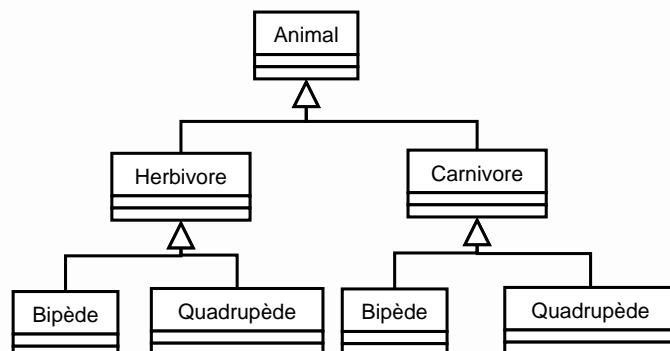
Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

✓ On peut aussi spécialiser les animaux d'abord par leur régime :



⇒ En fait les 2 solutions précédentes présentent le même défaut de **redondance**.

I - Les relations entre Classes

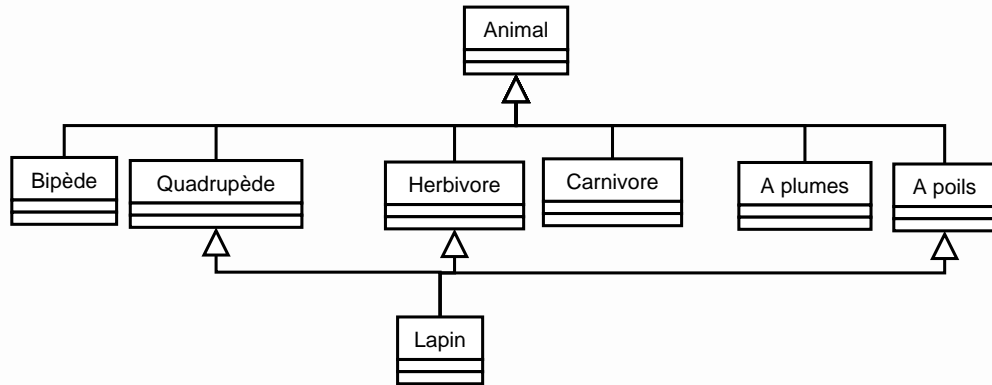
Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

⇒ Solution plus élégante possible : l'héritage multiple



I - Les relations entre Classes

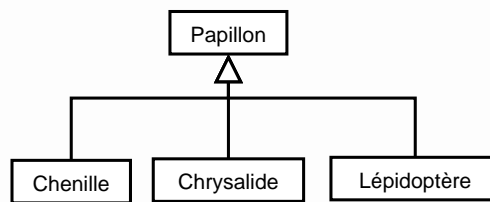
Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

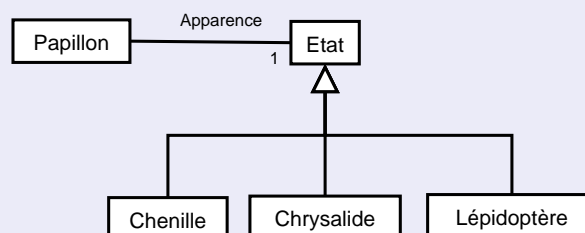
Bibliographie

✓ La relation d'héritage entre classes introduit un couplage statique très fort dans le modèle, non mutable => pas adaptée à la modélisation d'une métamorphose (changement dynamique de type)



Délégation d'héritage

Une solution consiste à "sortir" la caractéristique mutable de la classe, pour le déléguer à une classe qui devient alors la base d'une hiérarchie :



l'apparence du papillon est traduite par une association entre la classe Papillon et la classe Etat, base d'une classification d'états (hiérarchie)

I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

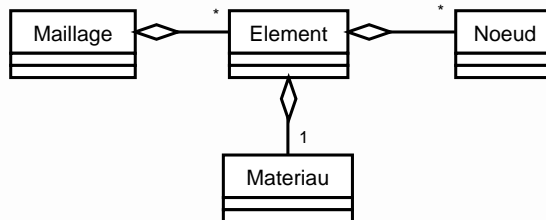
Aggrégation

traduit la relation "possède", "est constitué de", ou "est fait de"

L' **Aggrégation** est une relation bi-directionnelle dissymétrique exprimant un couplage plus fort que la simple association :

- traduit des relations de type **maître-esclave**, ou **contenant-contenu**,
- relation transitive, non symétrique et réflexive
- l'une des classes est un agrégat (le tout), composé de l'aggrégation des autres classes (les parties).

Exemple d'aggrégation :



Un maillage éléments finis "est constitué" d'éléments.
Chaque élément "est fait" d'un matériau et "possède" des noeuds

I - Les relations entre Classes

Épisode I

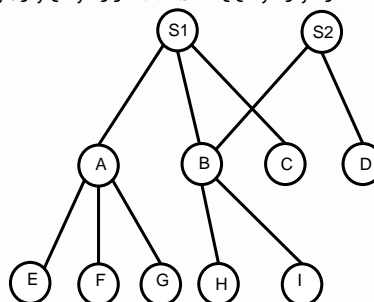
Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

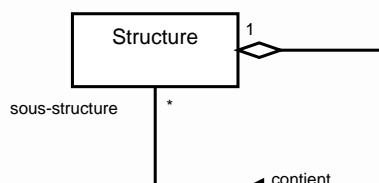
Bibliographie

La réflexivité de l'aggrégation permet de traduire des structures arborescentes :

soit $S1=\{A,B,C\}$, $S2=\{B,D\}$, $A=\{E,F,G\}$ et $B=\{H,I\}$
-> $S1=\{\{E,F,G\},\{H,I\}\}$ et $S2=\{\{H,I\},D\}$



L'arborescence peut être modélisée par une aggrégation réflexive :



```
Structure S1,S2,A,B,C,D,E,F,G,H,I;  
A.add(E,F,G); B.add(H,I);  
S2.add(B,D);  
S1.add(A,B,C);
```

I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

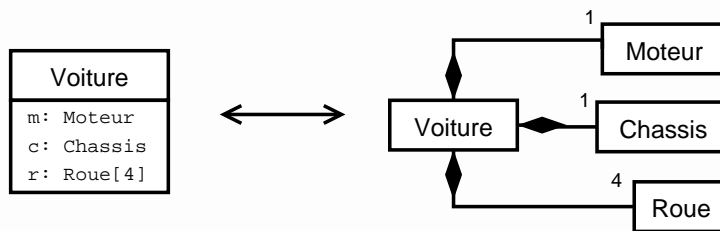
Bibliographie

Composition

forme d'aggrégation avec un couplage encore plus fort

- Les composants ne sont pas partageables entre plusieurs agrégats
- La destruction de l'agrégat entraîne la destruction des composants agrégés
- La composition agit comme un conteneur exclusif.

Les attributs d'une classe peuvent être vus comme des composants, au sens de la relation de composition :



I - Les relations entre Classes

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

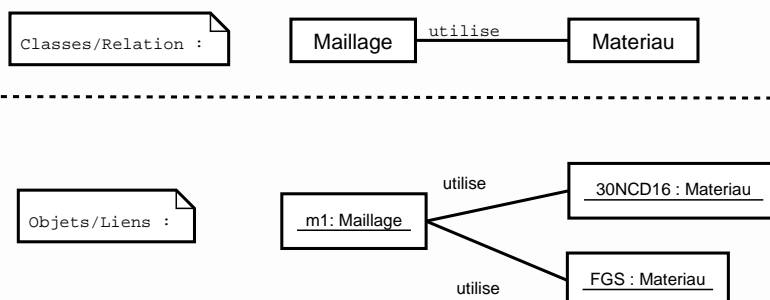
Épisode II

Bibliographie

Association

relation simple bi-directionnelle entre classes

- Les classes associées ont simplement conscience de l'existence les unes des autres
- L'association est un couplage faible entre classes
- Le fait que 2 classes soient associées suffit pour qu'elles puissent échanger des messages.



I - Les relations entre Classes

Épisode I

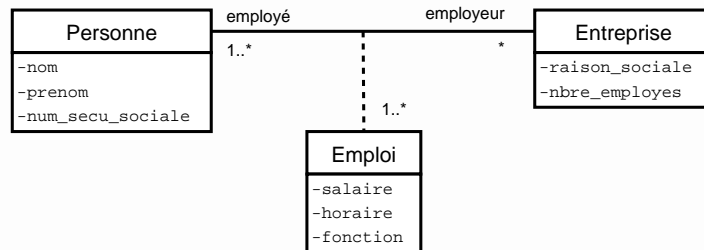
Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Classe portée par une association

- Une association entre 2 classes peut porter des données ou un comportement
- Une classe **portée par l'association** permet de représenter les données et le comportement de l'association.



- une entreprise emploie une ou plusieurs personnes (ses employés)
- chaque personne travaille dans zéro ou plusieurs entreprises (employeur)
- l'association (personne,entreprise) est un emploi de la personne par l'entreprise
- le modèle intègre le cas où une personne exerce plusieurs emplois chez le même employeur (-> cardinalité "1..*" placée près de la classe Emploi)
- les données et méthodes de la classe Emploi sont celles du contrat de travail liant employé et employeur.

I - Le Polymorphisme

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Polymorphisme

Mécanisme fondamental qui permet d'attribuer plusieurs significations à un même nom.

(C++) Polymorphisme de **Surcharge** (*overloading*) :

- Un même nom d'opérateur, de fonction ou de méthode peut réaliser des opérations différentes en fonction du **nombre** et du **type** des arguments passés
- La Résolution est faite par le compilateur => résolution **statique** des surcharges.

Polymorphisme d'**héritage** :

- Le même nom de méthode peut se retrouver dans une classe de base et une(des) classe(s) dérivée(s)
- La Résolution peut être **dynamique**, à l'exécution, en fonction du **type dynamique** des objets mis en oeuvre (C++ : mot clef `virtual`)
- Le polymorphisme d'héritage augmente la généralité des programmes
- Il n'est plus nécessaire d'écrire des lignes du style `"if (type objet == ...) then ... else ..."`

I - Le Polymorphisme

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Polymorphisme de Surcharge (+/- supporté par les langages) :

⇒ Surcharge des **opérateurs** (C++) :

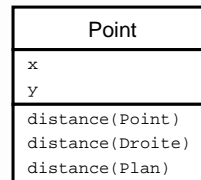
- On peut redéfinir par surcharge les opérateurs du langage
- Un opérateur peut exécuter du code différent en fonction du type des opérandes.
Matrice m1(...), m2(...);
Matrice m3 = m1 + m2; // + redéfini pour les objets Matrice

⇒ Surcharge des **fonctions** (C++) :

- On peut définir par surcharge plusieurs fonctions de même nom, réalisant des opérations différentes.
fonction retirer(carte bancaire, distributeur) : argent
fonction retirer(cheque, banque) : argent

⇒ Surcharge des **méthodes** d'une classe

- On peut définir par surcharge plusieurs méthodes de même nom au sein d'une classe, réalisant des opérations différentes.



I - Le Polymorphisme

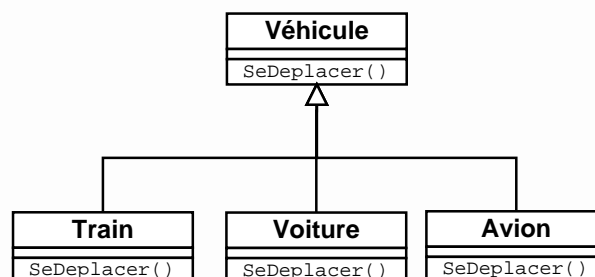
Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

Épisode II

Bibliographie

Exemple de conception et de programmation code C++ utilisant le polymorphisme dynamique :



```
Vehicule * convoi[3]; // le type statique de convoi est : tableau
                    // de 3 pointeurs sur Vehicule
convoi[0] = new Train("TGV"); // le type dynamique de convoi[0]
                             // est "pointeur sur Train"

convoi[1] = new Voiture("twingo"); // le type dynamique de convoi[1]
                                   // est "pointeur sur Voiture"

convoi[2] = new Bateau("Titanic"); // le type dynamique de convoi[2]
                                   // est "pointeur sur Titanic"

for (int i = 0; i < 3; i++)
{
    convoi[i]->seDéplacer(); // se déplace en fonction du typedDynamique !
}
```

I - La Généricité

Épisode I

Différentes Approches
Approche Fonctionnelle
Approche Orientée
Objet
Concept d'Objet
La Classe
Les relations
Polymorphisme
Généricité

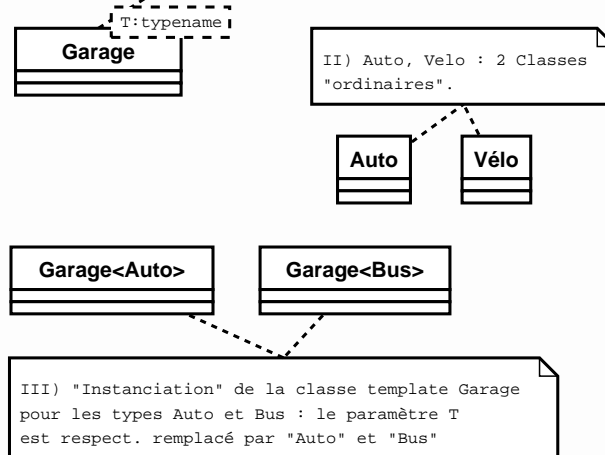
Épisode II

Bibliographie

Généricité

En Orienté Objet, correspond à la possibilité de paramétrer une Classe, une fonction, ou une méthode d'une classe par un paramètre générique formel représentant un type arbitraire.

I) La classe template peut utiliser le paramètre t qui représente un type arbitraire dans la définition de ses membres (attributs ou méthodes)



v1.1 du 26/04/2007 41 / 78

Épisode II : Modélisation "Orienté Objet" avec UML

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes
Diagramme d'Objets
Diagramme des cas
d'utilisation
langages OO

Bibliographie

"Tout ce qui est simple est faux,
tout ce qui ne l'est pas est inutilisable."
- Paul Valéry

Les premières approches objet

L'émergence des **approches 'objet'** (1990-1995) :

- Prise de conscience de l'importance d'une approche spécifiquement objet : comment structurer un Système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements (mais sur les deux) ?
- Plus de 50 *méthodes* objet sont apparues durant cette période (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE...)

3 méthodes ont émergé du lot :

- **OMT** (Object Modelling Technique), par James Rumbaugh, centre de R&D de General Electric
- **OOD** (Object Oriented Design), par Grady Booch
- **OOSE** (Object Oriented Software Engineering), par Ivar Jacobson, centre de développement d'Ericsson, en Suède.

v1.1 du 26/04/2007 42 / 78

II - OMG et UML

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

OMG (Unification et normalisation des approches objet)



OMG : Objet Management Group
<http://www.omg.org>

- Organisme à but non lucratif, créé en 1989 à l'initiative de grandes sociétés (HP, Sun, Unisys, American Airlines, Philips...)
- Fédère aujourd'hui plus de 850 acteurs du monde informatique
- Rôle : promouvoir des standards garantissant l'**interopérabilité** entre applications orientées objet

UML



UML : Unified Modeling Language
"langage de modélisation unifié"

fusion des 3 méthodes : OMT, Booch et OOSE.

- **Approche objet** pour **modéliser** la structure et le comportement d'un Système, indépendamment de toute méthode ou langage de programmation
- UML est un **langage de modélisation** et non une **méthode**.

v1.1 du 26/04/2007 43 / 78

II - UML

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

UML possède plusieurs facettes :

- C'est un langage **formel** et **normalisé** résultat d'un large consensus (industriels, méthodologistes...)
- C'est un **langage de modélisation objet**
 - puissant (couvre toutes les phases d'un cycle de développement)
 - ouvert (indépendant du domaine d'application et des langages d'implémentation).
- C'est un support de communication
- C'est un *cadre méthodologique*.

Mais

- La mise en pratique d'UML nécessite l'apprentissage du langage (encore un!).

v1.1 du 26/04/2007 44 / 78

II - Les diagrammes UML

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Principaux diagrammes proposés par UML pour modéliser les différents aspects du Système étudié (x : abordé dans ce support) :

Diagrammes Structurels

- x Diagramme de **Classes** : un des plus importants en conception OO, fournit la modélisation des entités du Système
- x Diagramme d'**Objets** : illustre le diagramme de classes en terme d'Objets (utile dans des cas complexes)

Diagrammes Comportementaux

- o Diagramme des **Cas d'utilisation** : structure des fonctions nécessaires aux utilisateurs du Système
- o Diagramme de **Séquences** : séquençement temporel des échanges entre les objets du Système
- o Diagramme d'**Activités** : règles d'enchaînement des activités du Système ("organigramme" amélioré)
- o Diagramme d'**États-transition** : cycle de vie commun aux objets d'une classe
- o Diagramme de **Collaboration** : organisation structurelle des objets qui envoient et reçoivent des messages

v1.1 du 26/04/2007

45 / 78

II - Les diagrammes UML

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

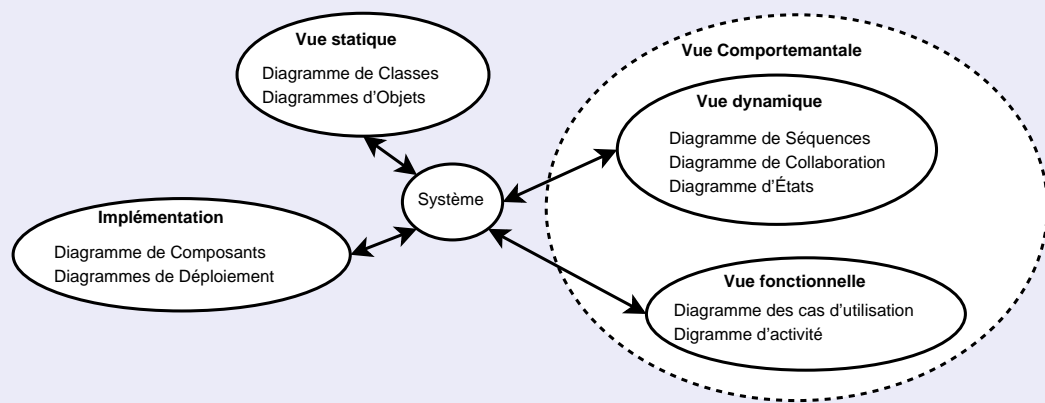
langages OO

Bibliographie

Diagrammes d'implémentation

- o Diagramme de **Déploiement** : structure du réseau informatique en charge du Système, installation des composants d'exploitation
- o Diagramme de **Composants** : structure des composants d'exploitation (bibliothèques dynamiques, bases de données, ...)

Les diagrammes UML permettent de proposer plusieurs vues du Système



v1.1 du 26/04/2007

46 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Diagramme de Classes

Utilité :

- C'est le diagramme le plus important de la modélisation orientée objet
- Montre la structure des Classes qui constituent le modèle orienté objet du Système étudié
- C'est une vue **statique** qui ne prend en compte que les dimension structurelle et fonctionnelle, pas la dimension temporelle.

Les éléments du diagramme sont :

- Les **Classes**
- Les **Relations** entre Classes.

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

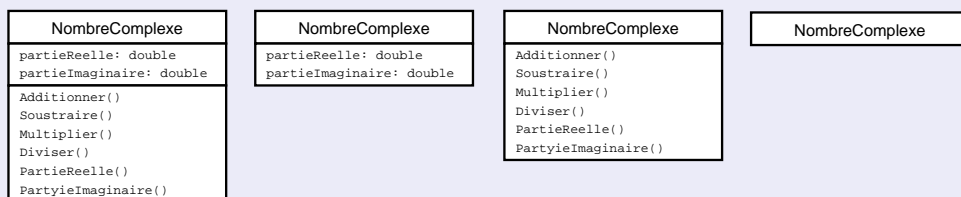
Bibliographie

Classe

⇒ UML propose la notion de **classificateur** : "élément de modèle qui décrit des caractéristiques structurelles et comportementales"

⇒ Une Classe est un classificateur représenté par un rectangle pouvant être divisé en trois compartiments :

- premier compartiment : **nom** de la classe
- second compartiment : liste des **attributs**, et éventuellement leur type
- troisième compartiment : liste des **méthodes** de la classe.



En fonction du contexte (Analyse, Conception préliminaire, détaillée, ...) :

- Il est utile ou pas de présenter tous les détails des attributs et des méthodes (type de valeur retournée, liste et type des arguments, ...)
- On peut même omettre de représenter tout ou partie des compartiments.

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

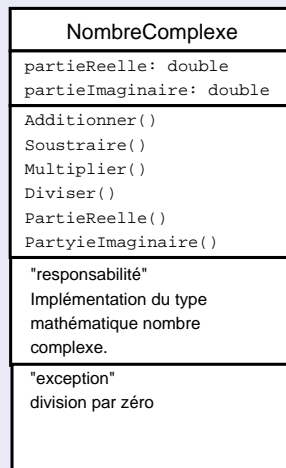
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Compartiments supplémentaires

responsabilité de la classe, exceptions, ...



II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

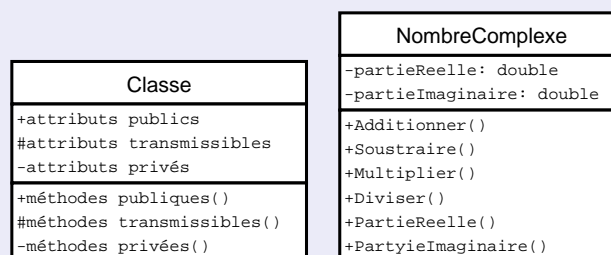
langages OO

Bibliographie

Visibilité des membres

Un caractère placé devant le nom des membres d'une classe traduit leur visibilité :

- le caractère '+' traduit la visibilité public
- le caractère '#' traduit la visibilité protected
- le caractère '-' traduit la visibilité private.



II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Association

- L'Association est la plus simple des relations pouvant exister entre deux classes (ou entités UML)
- Elle exprime simplement que les deux classes ont connaissance l'une de l'autre
- Chaque classe peut mettre en œuvre les membres publics (attribut ou méthode) de l'autre (-> envoi de messages).

Relation bi-directionnelle entre 2 classes :



v1.1 du 26/04/2007

51 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Décoration des Relations

Des attributs graphiques (**décoration** UML) permettent de préciser des caractéristiques des relations :

- relation unidirectionnelle :



- relation nommée (flèche optionnelle pour le sens de lecture) :

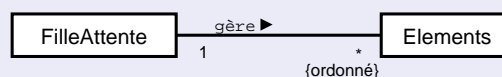


- indication des cardinalités :



- un maillage utilise 1 ou plusieurs matériaux
- un matériau est utilisé par aucun ou plusieurs maillages.

- indication d'une contrainte



une file d'attente gère zéro ou plusieurs éléments ordonnés.

v1.1 du 26/04/2007

52 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Cardinalité des relations

Chaque extrémité d'une relation peut recevoir une cardinalité :

- placée à proximité de la classe concernée
- précise le nombre d'instances de la classe qui participent à la relation

<i>cardinalité</i>	<i>signification</i>
1	exactement un
0..1	zéro ou 1
M..N	De M à N (entiers)
*	Zéro ou plusieurs
0..*	Zéro ou plusieurs
1..*	Un ou plusieurs

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

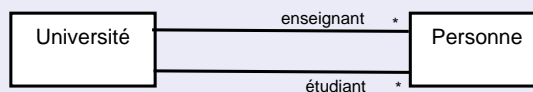
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Rôles

- Quand une classe participe à une association, elle y joue un rôle spécifique ;
- ce rôle peut être indiqué à l'extrémité du trait qui symbolise l'association.



une personne peut être ("jouer le rôle de") un enseignant ou un étudiant.

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

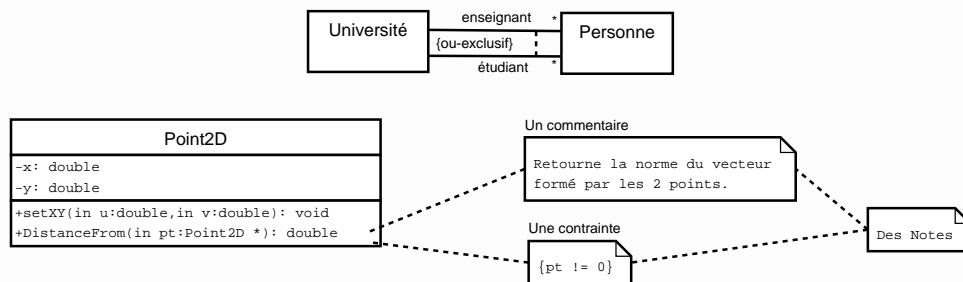
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Contraintes

- Relations entre éléments de modélisation
- Définissent des propositions qui doivent être vraies pour garantir la validité du modèle
- Peuvent être spécifiées :
 - en langage naturel
 - en pseudo-code
 - en langage formel en utilisant le langage OCL (Object Constraint Language).
- Sont indiquées entre accolades et placée près de l'élément concerné.



v1.1 du 26/04/2007

55 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

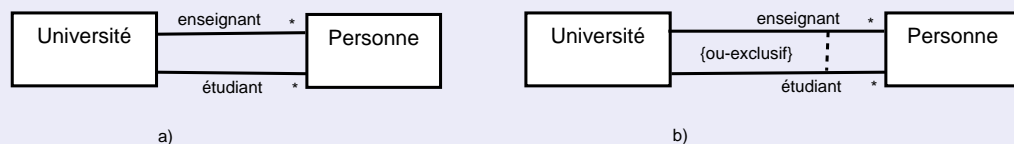
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Contraintes

Exemple de contrainte portant sur les relations entre classes :



La différence entre les 2 points de vue exprimés par les 2 modèles vient de la présence ou pas de contraintes :

- une même personne peut être à la fois un enseignant et un étudiant :
 - ⇨ ce modèle rend compte de la possibilité pour un enseignant d'être aussi inscrit comme étudiant dans son université.
- la contrainte `{ou exclusif}` est posée entre les 2 associations possibles :
 - une personne est soit un étudiant, soit un enseignant, pas les 2 à la fois.
 - ⇨ Ce modèle s'applique à une université où un enseignant ne peut pas s'inscrire comme étudiant dans son université.

v1.1 du 26/04/2007

56 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

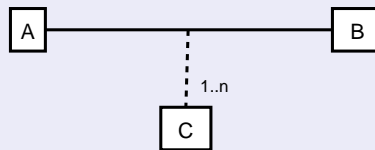
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Classe-Association

- Si l'association entre deux classes A et B génère des données et/ou des méthodes, on peut les modéliser par une classe association C, "portée" par cette association
- La cardinalité placée du côté de la classe C indique le nombre d'instances de C qui doivent être créées à chaque occurrence d'une association entre A et B.



v1.1 du 26/04/2007

57 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

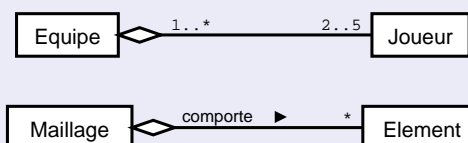
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Aggrégation

- Relation bidirectionnelle dissymétrique qui exprime un couplage fort entre des classes
- L'une des classes est un agrégat (le tout)
- L'agrégat est composé de l'aggrégation des autres classes (les parties)
- L'aggrégation correspond à la phrase "est fait de", ou "possède"
- Remarque : l'aggrégation ne lie pas les durées de vie de l'agrégat et des parties.



v1.1 du 26/04/2007

58 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Composition

- Traduit un couplage entre des classes encore plus fort que l'agrégation
- L'une des classes contient les autres
- La composition correspond à la phrase "est composé de", avec une dépendance forte entre les classes
- Si un objet de la classe maître disparaît, les objets contenus disparaissent aussi
- Un objet de la composition ne peut pas appartenir à une autre composition.



v1.1 du 26/04/2007

59 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Dépendance

- Relation unidirectionnelle qui exprime qu'un élément de modélisation (le plus souvent une classe) dépend d'un autre élément
- La dépendance est souvent utilisée quand une classe utilise une autre classe comme argument dans la signature d'une méthode par exemple
- Une dépendance est représentée par une flèche en pointillés dirigée vers la dépendance.



La classe A dépend de la classe B.

v1.1 du 26/04/2007

60 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

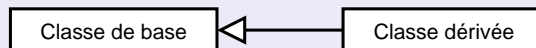
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Héritage

- Relation bidirectionnelle asymétrique qui permet de traduire la relation "est un"
- Elle est représentée par une flèche se terminant par un triangle fermé dirigée de la **classe dérivée** (enfant) vers la **classe de base** (parent).



v1.1 du 26/04/2007

61 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

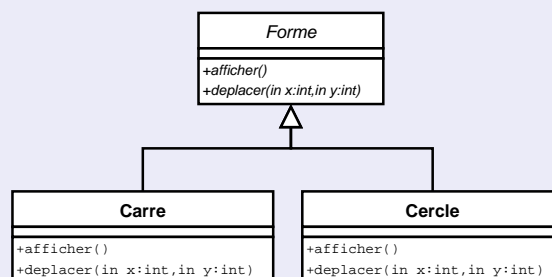
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Classe **Abstraite** (par opposition à une classe **concrète**)

- **Ne peut pas être instanciée**
- Fournit un moyen d'imposer un modèle qui devra être suivi par toute classe dérivée concrète
- Possède des méthodes **abstraites** qui **doivent** être définies par toute classe dérivée concrète
- Le caractère abstrait est représenté graphiquement par des noms (classe, méthode) en *italique*.



La classe abstraite *Forme* spécifie 2 méthodes abstraites *afficher()* et *deplacer()* qui **doivent** être définies dans les classes dérivées concrètes *Carre* et *Cercle*.

v1.1 du 26/04/2007

62 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

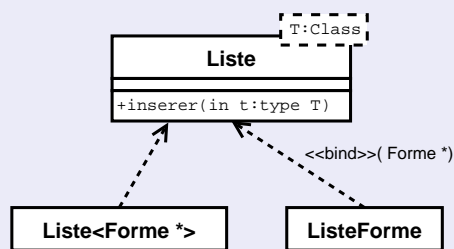
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Classe Template

- Dépend d'un paramètre template exprimant un type arbitraire
- Sert de base pouvant générer toute une famille de classes obtenues en fournissant un type particulier acomme paramètre template
- Doit être "instanciée" (au sens du compilateur) : on fournit un type donné au paramètre template pour obtenir une "vraie" classe
- Permet une définition **générique** des membres de la classes
- Très utilisées pour les classes conteneurs génériques, contenant "n'importe quoi".



La classe template Liste utilise comme paramètre le type T :
la classe Liste<Forme *> (ListeForme) est une instance qui correspond à une classe concrète
où le paramètre template est le type "Forme *" (pointeur sur Forme).

v1.1 du 26/04/2007

63 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

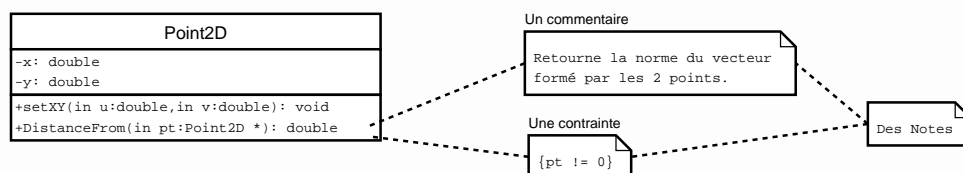
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Notes (tous diagrammes)

- Décorations graphiques importantes rattachées à un élément ou à un ensemble d'éléments
- Permettent de représenter des contraintes ou des commentaires textuels associés à des éléments
- Sont représentées par un rectangle écorné qui contient un texte.



v1.1 du 26/04/2007

64 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

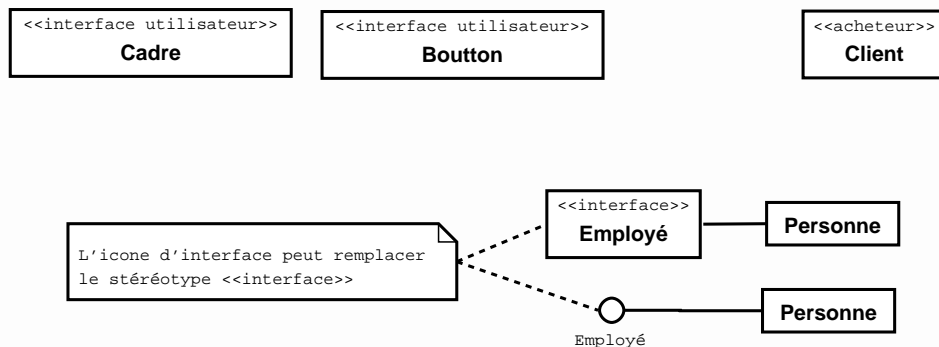
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Stéréotypes (tous diagrammes)

- Font partie des mécanismes d'extension d'UML
- Étendent le vocabulaire d'UML en permettant la création de nouvelles briques de base
- Sont représentés par un mot entre guillemets placé au-dessus du nom de l'élément stéréotypé
- Peuvent aussi être représentés par une icône symbole de l'élément stéréotypé (exple : l'icône "interface" proposée en standard par UML).



v1.1 du 26/04/2007

65 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Un diagramme de Classes exprime un Point de Vue

- Un diagramme de classes peut modéliser n'importe quel Système (matériel, service, logiciel)
- Un diagramme de classes est un modèle qui exprime un **point de vue**
- Pour un même Système, on peut construire des diagrammes de classes différents suivant le point de vue utilisé :
 - **Structuel** (de quoi le système est fait, point de vue technologique, CAO)
 - **Fonctionnel** (comment le système fonctionne)
 - **Organisationnel** (comment le système est organisé)
 - **Analyse** (que doit faire le système)
 - **Conception** (Comment faire ce que le système doit faire)
 - ...
- On construit des diagrammes plus ou moins riches en détails en fonction du point de vue :
 - les diagrammes d'Analyse montrent peu de détails sur les classes, mais s'attachent à lister toutes les classes et leurs relations
 - les diagrammes de Conception préliminaire rajoutent beaucoup de détails (et de classes) illustrant les caractéristiques de la solution proposée
 - les diagrammes de Conception détaillée fournissent tous les détails d'implémentation.

v1.1 du 26/04/2007

66 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

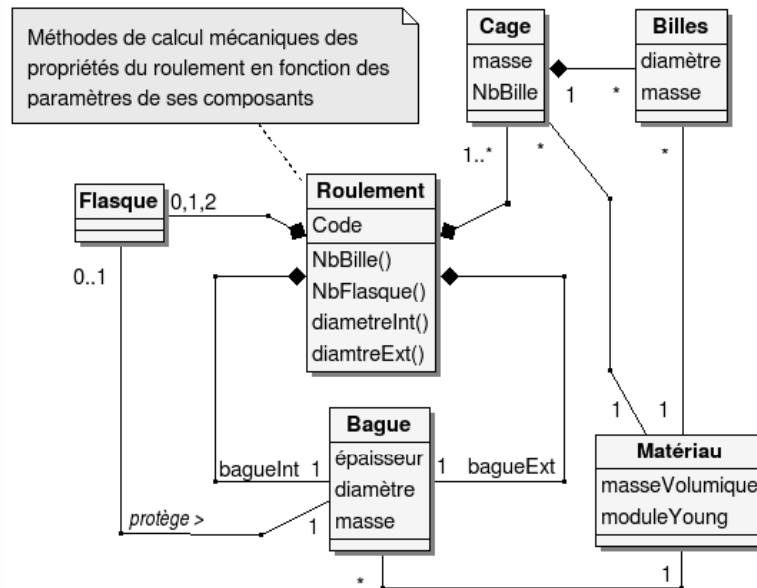
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Exemple de diagrammes de classes : Roulement

Point de vue du concepteur du roulement :



v1.1 du 26/04/2007

67 / 78

II - Diagramme de classes

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

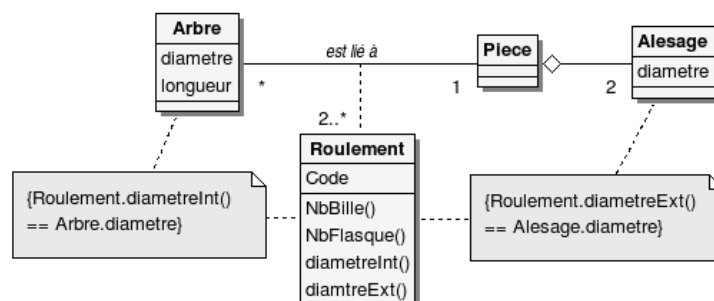
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Exemple de diagrammes de classes : Roulement

Point de vue de l'utilisateur du roulement :



v1.1 du 26/04/2007

68 / 78

II - Diagramme d'Objets

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

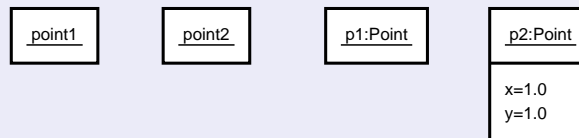
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Représentation des Objets

- Un objet est représenté par un **rectangle compartimenté** contenant le **nom de l'objet souligné**
- On peut également utiliser la syntaxe **nom_objet :nom_classe** pour insister sur le type de l'objet
- On peut représenter le compartiment des attributs, montrant les valeurs affectées pour un objet particulier ;



v1.1 du 26/04/2007

69 / 78

II - Diagramme des cas d'utilisation

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Diagramme des Cas d'utilisation

Utilité :

- Permet de recueillir, d'analyser et d'organiser les **besoins** des utilisateurs et de recenser les grandes **fonctionnalités** du Système
- Modélise le comportement d'un Système, d'un sous-système, d'une classe , tel qu'un utilisateur extérieur le voit
- Scinde les fonctions du Système en unités cohérentes : les **cas d'utilisation**.

Les éléments du diagramme sont :

- Les **Cas d'utilisation** : chacun représente un service rendu par le Système
- Les **Acteurs** : un acteur représente le rôle joué par une entité externe (personne, dispositif matériel, processus ...) qui interagit avec le Système
- Les **Relations** :
 - associations entre cas d'utilisation et acteurs
 - dépendances ou héritage entre cas d'utilisation
 - héritage entre acteurs.

v1.1 du 26/04/2007

70 / 78

II - Diagramme des cas d'utilisation

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

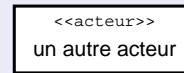
langages OO

Bibliographie

Acteurs

représenté graphiquement :

- soit sous la forme d'un petit personnage (*stick man*)
- soit sous la forme d'une classe incluant le mot clef «acteur».



✓ On distingue souvent quatre grandes catégories d'acteur :

- les **acteurs principaux** qui utilisent les fonctions principales du Système (utilisateurs d'un Système automatisé, d'un code de calcul ...)
- les **acteurs secondaires** qui effectuent en général des tâches administratives ou de maintenance
- le **matériel externe** qui regroupe les dispositifs matériels qui font partie du champ applicatif et qui sont donc utilisés par le Système
- les **autres systèmes** qui peuvent interagir avec le Système objet de l'étude.

v1.1 du 26/04/2007

71 / 78

II - Diagramme des cas d'utilisation

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

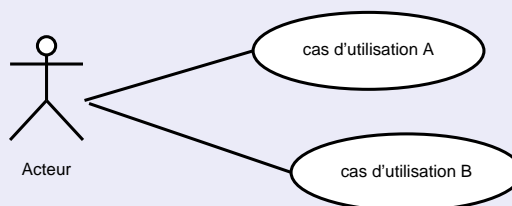
Diagramme des cas
d'utilisation

langages OO

Bibliographie

Cas d'utilisation

représenté graphiquement une ellipse contenant le nom du cas (verbe),
nécessairement relié à un acteur :



- Les cas peuvent être contenus dans un rectangle (limites du Système) ; les acteurs sont à l'extérieur du rectangle, car ils ne font pas partie du système
- La relation acteur-cas est une association représentée par une ligne.

Un cas d'utilisation :

- est l'image d'une fonctionnalité d'un Système ou d'une classe, déclenchée en réponse à la stimulation d'un acteur externe (utilisateur ou processus)
- correspond à un ensemble de séquences d'actions réalisées par le Système produisant un résultat observable intéressant pour un acteur.

v1.1 du 26/04/2007

72 / 78

II - Diagramme des cas d'utilisation

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

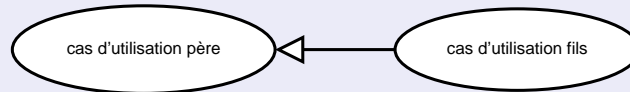
langages OO

Bibliographie

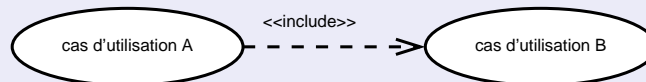
Relations entre cas d'utilisation :

UML prévoit trois types de relation entre cas d'utilisation :

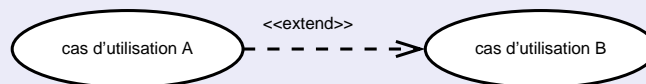
- **Héritage** (généralisation) : le cas d'utilisation dérivé est une spécialisation du cas d'utilisation parent



- **Inclusion** : un cas d'utilisation a besoin d'un autre cas d'utilisation pour réaliser sa tâche



- **Extension** : un cas d'utilisation A ajoute ses fonctionnalités à celles d'un autre cas d'utilisation B (on dit alors que A étend B)



v1.1 du 26/04/2007

73 / 78

II - Diagramme des cas d'utilisation

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Description textuelle des cas d'utilisation

- Comme la plupart des diagrammes UML, le diagramme des cas d'utilisation nécessite souvent une description narrative (textuelle) associée
- Décrire un cas d'utilisation consiste à définir son contexte, et à détailler la communication entre le cas et l'acteur.

La fiche de description textuelle d'un cas d'utilisation n'est pas normalisée par UML. Il est raisonnable de s'inspirer de la structure suivante :

- 1 **Sommaire d'identification** (obligatoire) : Titre, Objectif, Dates, Version, Responsable, Acteurs concernés (principaux et secondaires)
- 2 **Description des scénarios** (obligatoire) : Description du scénario nominal, des scénarios alternatifs, les scénarios d'exceptions (erreurs).
On peut structurer les scénarios selon la "programmation par contrat" (B. Meyer) :
 - **Préconditions** : conditions garantissant le démarrage correct du cas
 - **Processus** : description pas à pas des échanges acteur-cas d'utilisation
 - **Arrêt** : liste les fins possibles du cas
 - **Postconditions** : conditions devant être satisfaites à la fin, pour garantir que le Système est dans un état cohérent.
- 3 **Exigences fonctionnelles** (optionnel) : Informations diverses (Fréquence, volumétrie, ... interface Homee-Machine, règles d'ergonomie, charte graphique ...)

v1.1 du 26/04/2007

74 / 78

II - Diagramme des cas d'utilisation

Épisode I

Épisode II

Modélisation Objet

UML

Diagramme de Classes

Diagramme d'Objets

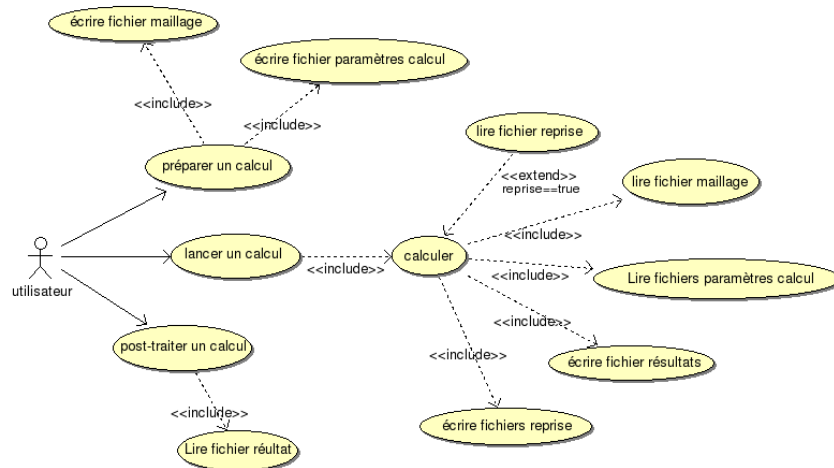
Diagramme des cas d'utilisation

langages OO

Bibliographie

Exemple de diagramme de cas d'utilisation pour un programme de simulation mécanique :

- L'utilisateur peut lancer 3 tâches principales : préparer, exécuter ou post-traiter un calcul
- Chacune de ces tâches est un cas d'utilisation qui inclut d'autres cas d'utilisation.



Le cas d'utilisation "lire fichier reprise" étend le cas d'utilisation "calculer", si on est dans une situation de calcul sur reprise.

v1.1 du 26/04/2007

75 / 78

II - Diagramme des cas d'utilisation

Épisode I

Épisode II

Modélisation Objet

UML

Diagramme de Classes

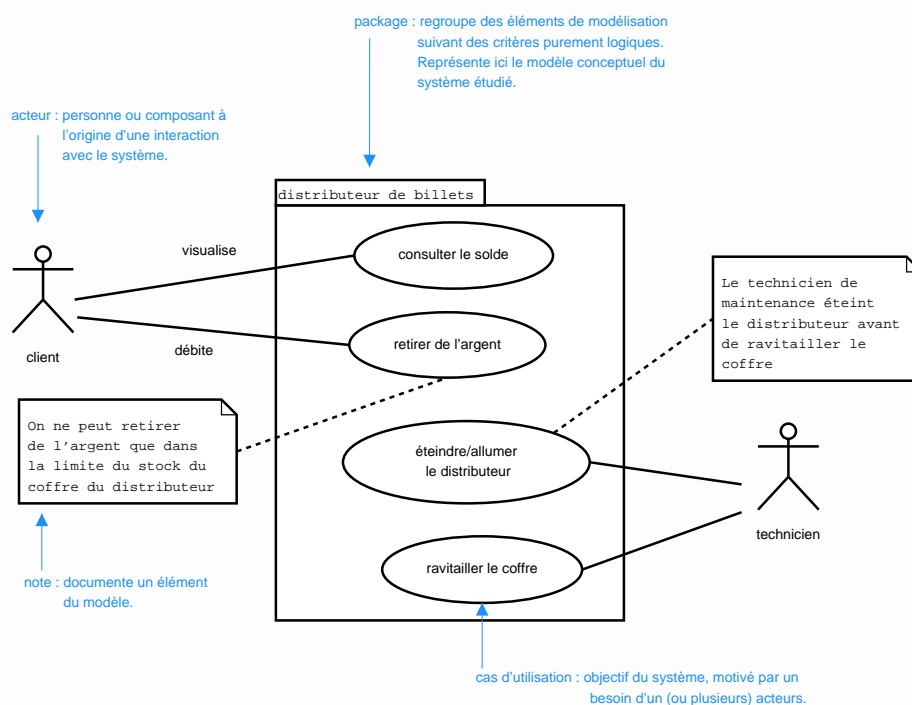
Diagramme d'Objets

Diagramme des cas d'utilisation

langages OO

Bibliographie

Exemple de diagramme de cas d'utilisation :



v1.1 du 26/04/2007

76 / 78

II - Vers les langages OO

Épisode I

Épisode II

Modélisation Objet
UML

Diagramme de Classes

Diagramme d'Objets

Diagramme des cas
d'utilisation

langages OO

Bibliographie

Génération automatique de code informatique

- La plupart des logiciels de création de diagrammes UML proposent la *génération automatique de code orienté objet* traduisant les informations contenues dans les diagrammes.
- Principaux langages informatiques orientés objet cibles :
 - C++
 - JAVA
 - Eiffel
 - SmallTalk
 - C#
 - PHP
 - Perl
 - ASP
 - Python
 - ...

Bibliographie

Épisode I

Épisode II

Bibliographie

OO

- *Conception et Programmation orientées objet*", Bertrand Meyer (Eyrolles)

UML

- *Modélisation objet avec UML*, Pierre-Alain Muller et Nathalie Gaetner (Eyrolles 2000, 2ème édition, ISBN 2-212-09122-2)
- *UML2 et les design pattern*, Graig Larman (Pearson Education 2005, 3ème édition, ISBN 2-7440-7090-4)
- *UML2 par la pratique, Étude de cas et exercices corrigés*, Pascal Roques (Eyrolles 2006, 5ème édition, ISBN 2-212-12014-1)
- *Le guide de l'utilisateur UML*, Grady Booch, James Rumbaugh et Ivar Jacobson ((Eyrolles 2003, 3ème édition, ISBN 2-212-09103-6)