Présentation générale du langage de modélisation objet :

U.M.L. (Rumbaugh, Booch & al.)

Bernard ESPINASSE

Professeur à l'Université d'Aix-Marseille

2004

- **Introduction**
- **Les modèles d'UML**
- **Les Diagrammes d'UML**

Les méthodes Orientées Objet

- influencées par le développement du langage **Ada** et des langages de programmation basés sur les objets **C++**
- la plupart aborde l'étude d'un problème est réalisée suivant 3 aspects :
  - aspect **statique** : identifie les propriétés des objets et leurs liaisons avec les autres objets,
  - aspect **dynamique** définit le cycle de vie des objets : **comportement** des objets, les **différents états** par lesquels ils passent et **événements** déclenchant ces changements d'états.
  - aspect **fonctionnel** : précise les fonctions réalisées par les objets par l'intermédiaire des méthodes
- **méthodes orientées objet les plus connues :**
  - **OMT** (Rumbaugh et al. 1991)
  - **BOOCH** (Booch 1991)
  - **Objectory-OOSE** (Jacobson & al. 1992)
  - OOA (Object Oriented Analysis - Shlaer & Mellor 1992)
  - OOA (Object Oriented Analysis - Coad & Yourdon 1992)
  - HOOD …

-> Unification des méthodes

De l'unification des méthodes objet à UML (Unified Modeling Language)

- 1995 unification rendue possible par l'expérience acquise par les diverses méthodes

-> 1995 - Méthode Unifiée 0.8 : rapprochement OMT (Rumbaugh &al.) OOD (Booch)

- 1996 - UML 0.9 : association de OOSE (Jacobson &al.)
- 1997 - UML 1.0 : soumission à l' OMG (janvier 97 - standardisation en cours)
- 1997 - ouverture du partenariat (DEC, HP, IBM, Microsoft, Oracle, ...)

- **Objectifs poursuivis :**
  - **représenter des systèmes** entiers (au dela du seul logiciel) par des **concepts objets**,
  - créer un **langage de modélisation** utilisable par les humains et les machines.
  - établir un **couplage explicite** entre les **concepts** et les **artefacts exécutables**,
- **Bibliographie complémentaire :**
  - **OMT** : Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W., "Object-Oriented Modeling and Design", Prentice Hall, 1991 - traduction française : "OMT : Modélisation et conception orientées objet", Masson - Prentice Hall, 1994.
  - **UML** : Muller P-A, Modélisation objet avec UML, Eyrolles, 1997.
  - Internet (**spécifications UML**) : http://www.rational.com/uml/

Modèles d'UML

- UML défini **6 modèles** pour la représentation des systèmes :
  - **Modèle des classes** : capture la structure statique,
  - **Modèle des états** : exprime le comportement dynamique des objets,
  - **Modèle des cas d'utilisation** : décrit les besoins de l'utilisateur,
  - **Modèle d'interaction** : représente les scénarios et les flots de messages,
  - **Modèle de réalisation** : montre les unités de travail,
  - **Modèle de déploiement** : précise la répartition des processus,
- UML défini **9 diagrammes** pour élaborer ces modèles

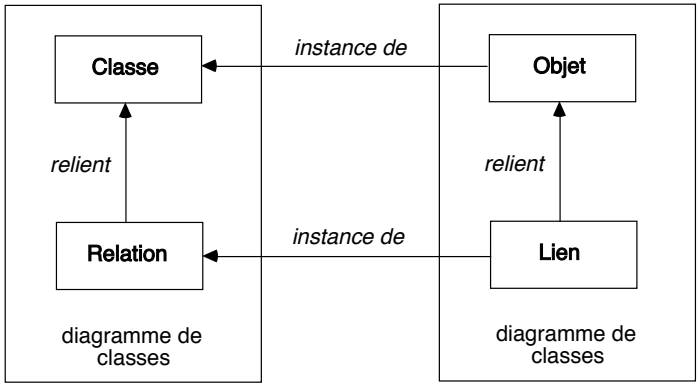
les notations graphiques UML pour ces diagrammes sont différentes de celles d'OMT, Booch, OOSE mais représentent les mêmes concepts objet

Diagrammes d'UML

- UML défini **9 diagrammes** pour élaborer les 6 modèles :
  - **diagrammes de classes** : représentation de la structure statique en termes de classes et relations,
  - **diagrammes d'interaction** :
    - **diag.de séquence** : représentation temporelle des objets et leurs interactions,
    - **diag.de collaboration** : représentation spatiale des objets, des liens et des interactions,
  - **diagrammes d'objets** : représentation des objets et leurs relations (diag. de collaboration simplifiés sans envois des messages),
  - **diagrammes d'états-transition** (Statecharts) : représentation du comportement d'une classe d'objet en termes d'états,
  - **diagrammes d'activités** : représentation d'une opération en termes d'actions,
  - **diagrammes de cas d'utilisation :** représentation des fonctions du système du point de vue de l'utilisateur,
  - **diagrammes de composants** : représentation des composants physiques d'une application,
  - **diagrammes de déploiement** : représentation du déploiement des composants sur les dispositifs materiels,

Classes / Objets

- **classe** = décrit un ensemble **d'objets**
- **association** = un ensemble de **liens**

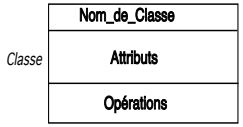


Les Diagrammes de Classes

- expriment la structure statique du système en termes de classes et de relations entre elles
- **extension du modèle Entité-Association** par l'introduction de :
  - l'**agrégation**,
  - la **généralisation**
  - la spécification d'**opérations** et **contraintes** au niveau des entités, nommées ici "**Classes**"

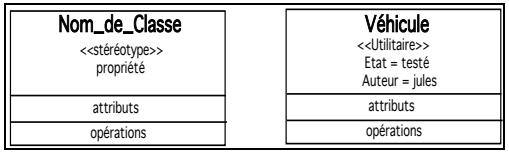
Classes

- une classe d'objets modélise un **ensemble d'objets** ayant :
  - des caractéristiques similaires (Attributs),
  - des comportements similaires (Opérations)
- diagramme de classes : présente **classes**, leurs **hiérarchies** et leurs **relations** entre elles.
- **conventions** graphiques adoptées : rectangle à **3 compartiments**



Classes : stéréotypes et propriétés

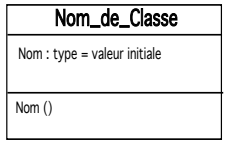
- une classe peut contenir un des **stéréotypes** suivants :
  - `<<signal>>` : occurrence remarquable déclenchant une transaction dans un automate,
  - `<<interface>>` : description des opérations visibles,
  - `<<métaclasse>>` : classe d'une classe,
  - `<<utilitaire>>` : classe réduite au concept de module et ne peut être instanciée.
- tout compartiment peut avoir des **propriétés** représentées par des **étiquettes**
- **étiquette** = paire (attribut, valeur) définie par l'utilisateur :



Classes : Attributs et Opérations

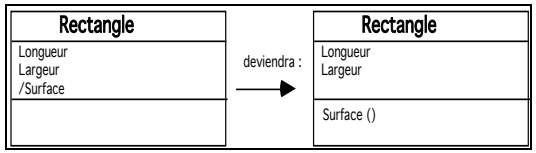
- **attributs** et **opérations** d'une classe peuvent être **explicités** :
  - syntaxe d'un **attribut** :

```
Nom_attribut : Type_Attribut = Valeur_Initiale
```



  - syntaxe d'un **attribut dérivé** :

```
/surface
```

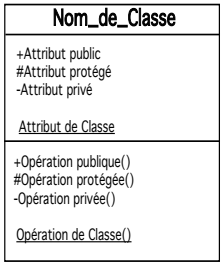


Classes : Attributs et Opérations

- syntaxe d'une **opération** :

```
Nom_Opération
(Nom_Argument : Type_Argument = Valeur_Par_Défaut,...)
: Type_Retourné
```

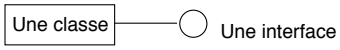
- **3 niveaux de visibilité** des **attributs** et des **opérations** :
  - **`public :`** élément visible à tous les clients de la classe (+),
  - **`protégé :`** élément visible aux sous-classes de la classe (#),
  - **`privé :`** élément visible à la classe seule (-).



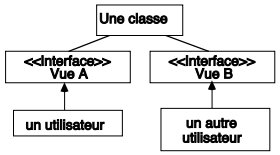
- **attributs et opération <u>soulignés</u> : visibles globalement** dans **toute la portée lexicale de la classe**

Classe : les Interfaces

- **interface** : fournit une vue totale ou partielle d'un ensemble de services offerts par un ou plusieurs éléments:

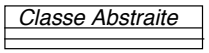


- peuvent être représentées au moyen de **classes stéréotypés** :



Classes abstraites

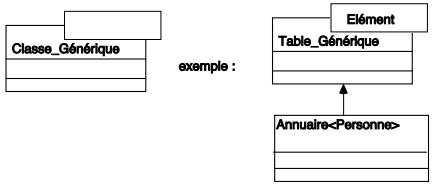
- pas directement instanciables
- ne donne pas naissance à des objets
- sont utilisées pour une spécifications de typage plus générale pour manipuler les objets instances d'une (ou plusieurs) de leurs sous-classes :



Classes paramétrables

= **modèles de classes** (classes générique de Eiffel ou template de C++)

- **doivent être instanciées pour être utilisées** : des paramétres personnalisent la classe réelle
- permettent de construire des **collections universelles, typées** par **paramétres effectifs**

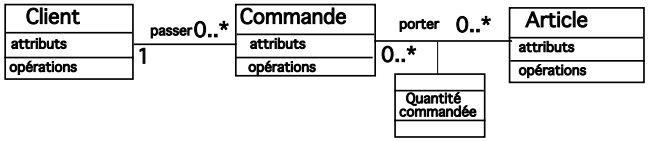


Classes utilitaires

- ne sont pas des types de données, elles **représentent des modules** (fonctions d'une bibliothèque mathématique par ex.),
- **ne peuvent pas être instanciées**,

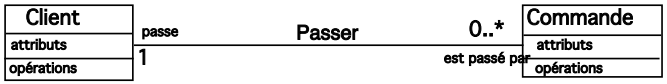
Associations binaires

- **peuvent relier des classes non nécessairement distinctes**
- représentées par un lien qui peut être **nommé** (forme active/forme passive/infinitif et en italique)



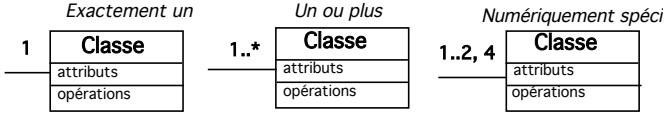
*l'association "porter" est **attribuée** (attribut "Quantité commandée")*

- lorsque l'association est caractérisée par un attribut, une nouvelle classe, **classe-association** sans nom est crée (ici pour l'attribut Quantité_commandée)
- l'extrémité d'un lien associatif est appelé "**rôle**"
- le nommage des roles est possible (lourd si cummulé à celui des assciations)

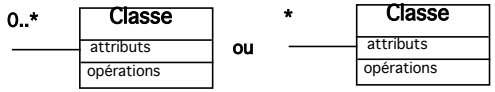


Multiplicités

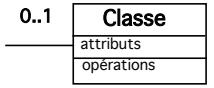
- caractérisent les **liens associatifs**
- spécification des multiplicités à l'autre bout de la ligne (**inverse cardinalités Merise**)



Plusieurs (zéro ou plus)



Optionnel (zéro ou un)

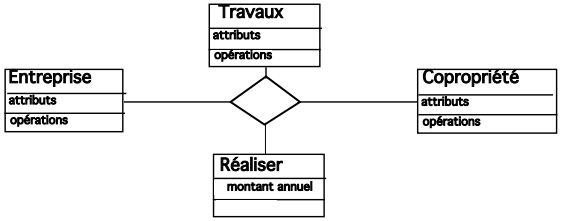


Remarque : ?

la notion de multiplicité **ne fonctionne plus** avec les associations **n-aires**.

Associations n-aires

- 2 façons de représenter une association n-aire :
- 1) par une **classe-association** ordinaire (avec losange):



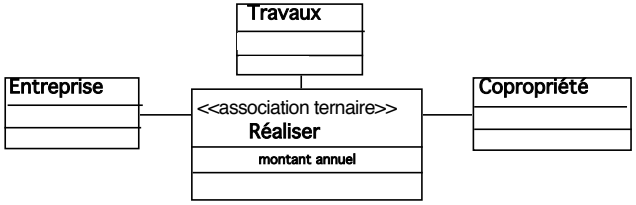
*ici **classe-association** REALISER reliée par un losange*

elle possède un attribut propre "montant annuel"

- à ces classes-associations peuvent être rattachés des **attributs** et/ou des **opérations** propres

Associations n-aires

- 2) par une **classe-association stérotypée** :



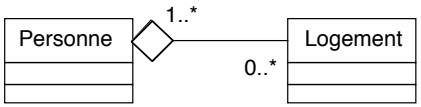
*ici **classe-association** Réaliser reliée par un losange*

elle possède un attribut propre "montant annuel"

- à ces classes-associations peuvent être rattachés des **attributs** et/ou des **opérations** propres

L'Agregation

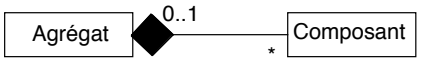
- association **non symétrique** dans laquelle une extrémité joue un rôle prédominant sur l'autre :
  - **une classe fait partie d'une autre classe,**
  - les **valeurs d'attributs** d'une classe **se propagent** dans les valeurs d'attributs d'une autre classe
  - une **action** sur une classe **implique** une action sur une autre classe,
  - les **objets** d'une classe sont **subordonnés** aux objets d'une autre classe.
- **ne peut concerner qu'un seule rôle de l'association**



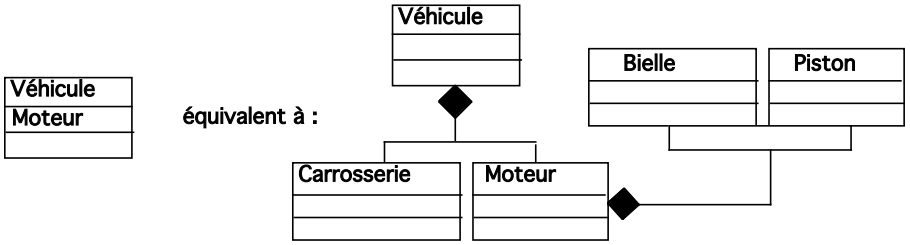
des Personnes peuvent être copropriétaires des mêmes Logements

La Composition

- cas particulier d'agrégation : **contenance physique**
- relation **transitive** du type **Composé-Composant** entre une classe "**agrégat**" et une classe "**composant**" :



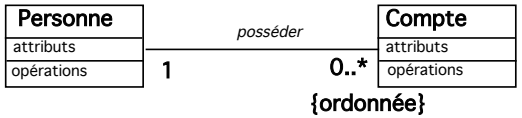
- implique une contrainte sur la valeur de la multiplicité du coté de l'agrégat
- composition et attributs sont **sémantiquement équivalents** (notation par attribut ou par composition) :



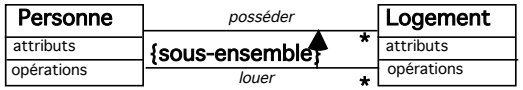
une classe Véhicule est composée par exemple de 2 autres classes, la Carrosserie et le Moteur qui a son tour est composé des classes Bielle et Piston

Contraintes sur les associations

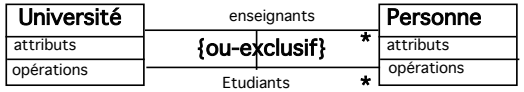
- contrainte **`{ordonnée}` :** une relation d'ordre décrit les objets placés dans la collection : l'ordre doit être maintenu,



- contrainte **`{sous-ensemble}` :** une collection est incluse dans une autre collection,

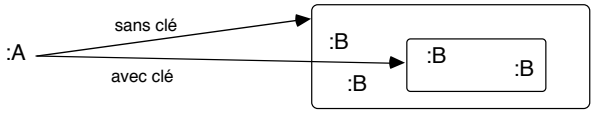


- contrainte **`{ou-exclusif}` :** pour un objet donné, une seule association est valide,

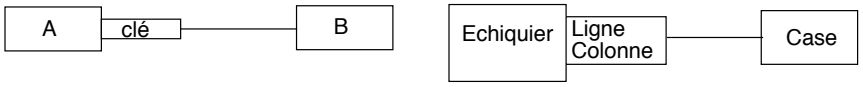


Restriction des associations (qualification)

- soit une association entre 2 classes A et B, une qualification consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association:



- est réalisée par un attribut spécifique appelé "clé" :
  - représentée sur le rôle de départ
  - appartient pleinement à l'association et non aux classes associées



combinaison d'une ligne et d'une colonne pour identifier une case de l'échiquier

La Navigation

• les **associations** :

  • décrivent un réseau de relation structurelles entre les classes
  • donnant naissance aux liens entre les objets, instances de ces classes

• les liens constituent des "**canaux de navigation**" entre les objets

• par **défaut** les associations sont **"navigables" dans les 2 sens**

• dans certains cas **un seul sens** est possible : celui de la **flèche** :

l'association entre les classes A et B est seulement navigable de A vers B

La Navigation : spécification des chemins

• **pseudo-langage** pour spécifier les **chemins** au sein des diagrammes de classes

• ce langage définit des expressions servant à préciser notamment des contraintes

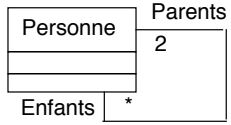
• **règle syntaxique R1 :**

```
cible ::= ensemble '.' sélecteur
```

avec :

  • **cible** = ensemble d'objet (une classe désigne {objets instances de la classe})
  • **sélecteur** = nom d'attribut /d'association /de rôle (opposé sur un lien) des objets de l'ensemble

exemple :



`UnePersonne.Enfants` désigne tous les enfants d'une personne donnée

La Navigation : spécification des chemins

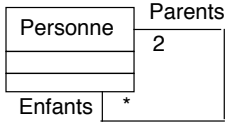
• **règle syntaxique R2 :**

```
cible ::= ensemble '.' '~'sélecteur
```

avec :

  • **sélecteur** = nom de rôle placé du coté de l'ensemble
  • **cible** = {d'objet} obtenu par navigation dans le sens opposé au nom de rôle

  • exemple :



`UnePersonne.~Enfants` désigne les parents d'une personne donnée

La Navigation : spécification des chemins

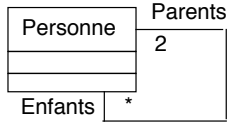
• **règle syntaxique R3 :**

```
cible ::= ensemble '[' expression_booléenne ']'
```

avec :

  • **expression booléenne** : construite à partir des objets de l'ensemble et des liens et valeurs accessibles par ces objets
  • **cible** = {d'objets} vérifiant cette expression booléenne = sous-ensemble de l'ensemble de départ

exemple :



l'expression booléenne `UnePersonne.Enfants[âge>=20]` désigne tous les enfants de plus de 20 ans d'une personne donnée

La Navigation : spécification des chemins

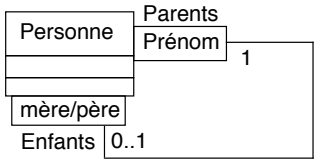
• **règle syntaxique R4 :**

```
cible ::= ensemble '.' sélecteur '[' valeur_de_clé ']'
```

avec :

- **sélecteur** = association qui partitionne l'ensemble à partir de la valeur de la clé
- **cible** = sous-ensemble de l'ensemble défini par l'association
- **expression booléenne** : construite à partir des objets de l'ensemble et des liens et valeurs accessibles par ces objets
- **cible** = {d'objets} vérifiant l'expression booléenne = sous-ensemble de l'ensemble de départ

exemple : restriction par l'ajout de clés sur l'association, ce qui réduit la multiplicité :



l'expression `UnePersonne.Enfants[UnPrénom]` identifie un enfant de façon non ambigue (chaque enfant d'une même famille a un prénom différent)

La Navigation : spécification des chemins

• expression de pré ou post conditions attachées aux spécifications des opérations :

1 - association correcte des parents et des enfants :

```
{UnePersonne = UnePersonne.Enfant[UnPrénom].(Parent[Mère] ou
 Parent[Père])}
```

2 - ou en navigant en sens inverse :

```
{UnePersonne = UnePersonne.Enfant[UnPrénom].(~Enfant[Mère] ou
 ~Enfant[Père])}
```

3 - ou enfin, en posant :

```
{papa = UnePersonne.Parent[Père]}
```

et

```
{papi = UnePersonne.(Parent[Père] ou (Parent[Mère]}.Parent[Père]}
```

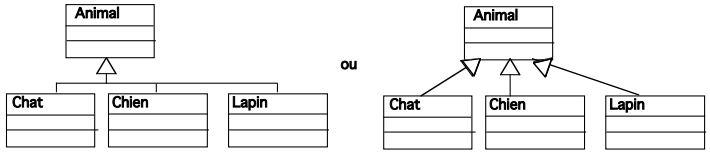
alors :

```
{papi.Parent[Père] = papa.Parent[Père].Parent[Père]}
```

... le papa de papi est le papi de papa !!!!

La Généralisation

• relation de classification **transitive** entre une classe plus générale - **super-classe** - et une ou plusieurs autres classes plus spécifiques - **sous-classes** -

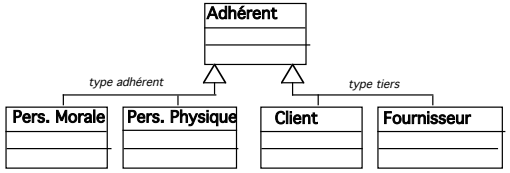


Chat, Chien, Lapin sont dites sous-classes de la super-classe Animal

• les attributs, opérations, relations et contraintes définies dans les super-classes sont **hérités** intégralement dans les sous-classes

• une classe peut avoir plusieurs super-classes : **généralisation multiple**

• une généralisation peut être faite selon des **critères indépendants** :



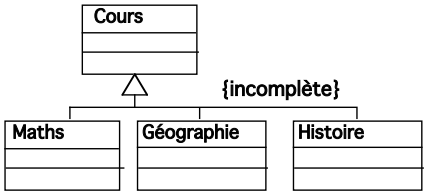
La Généralisation

• chaque sous-classe **hérite** des caractéristiques (attributs et opérations) de la surclasse

• toute sous-classe peut modifier et/ou ajouter de **nouveaux attributs** / **opérations** par rapport à ceux hérités

• possibilité de **contraintes** sur hiérarchies de classes (contraintes **d'intersection/disjonction):**

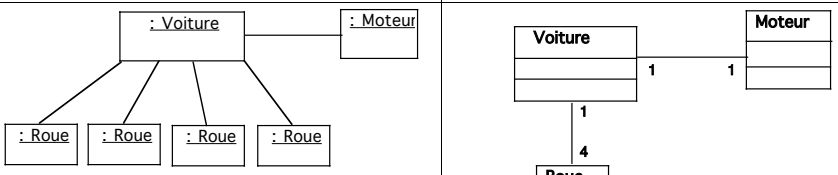
- **contrainte `{Disjoint}` ou `{Exclusif}`** : une sous-classe ne peut être sous-classe que d'une super-classe,
- **contrainte `{Chevauchement}` ou `{Inclusif}`** : une sous-classe peut être sous-classe de plusieurs super-classes (généralisation multiple possible),
- **contrainte `{Complete}`** : la généralisation est terminée, plus possible de rajouter de nouvelles sous-classes,
- **contrainte `{Incomplete}`** : il est possible de rajouter de nouvelles sous-classes :



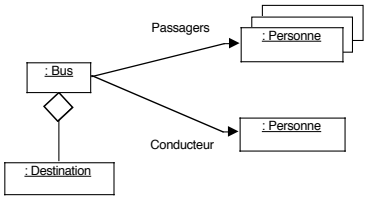
Les diagrammes d'objets

Représentation d'objets et de leurs liens :

diagramme d'objets | **diagramme de classes**



Décoration :

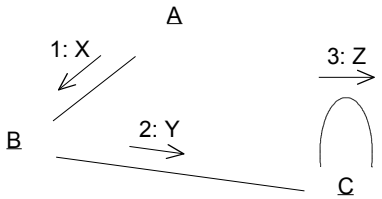


Les diagrammes de collaboration

soit :

- des **objets** reliés par des **liens** et qui se connaissent dans une situation donnée
- les **messages** échangés par les objets sont représentés le long de ces liens
- l'ordre d'**envoi des messages** est matérialisé par un **numéro de séquence**

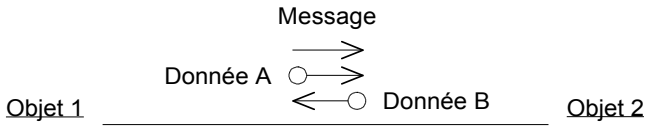
Exemple:



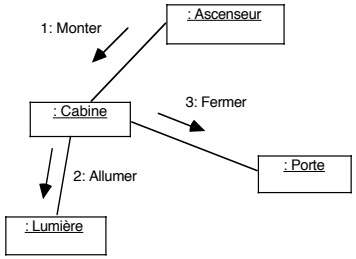
*Un objet **A** envoie un message **X** à un objet **B**,*
*puis l'objet **B** envoie un message **Y** à un objet **C**,*
*et enfin **C** s'envoie un message **Z**.*

Les diagrammes de collaboration

- **Représentation des messages :**



Exemple :

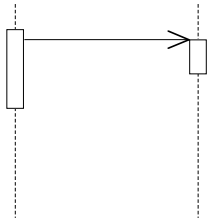


Les diagrammes de séquence

- L'accent est mis sur la communication, au détriment de la structure spatiale
- Chaque objet est représenté par une barre verticale
- Le temps s'écoule de haut en bas, de sorte que la numérotation des messages est optionnelle

Exemple:

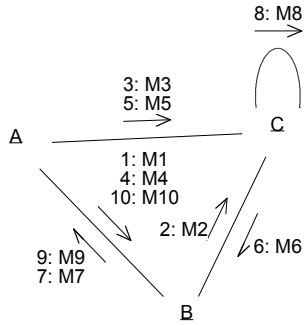
Un client Un serveur



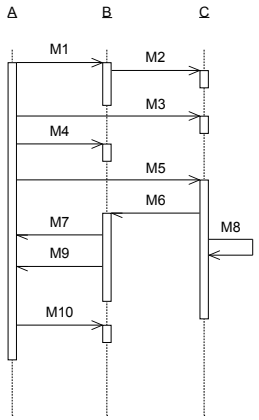
Diagrammes de collaboration / Diagrammes de séquence

Comparaison :

Diagrammes de collaboration

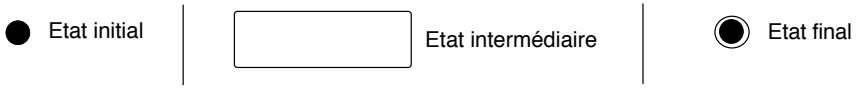


Diagrammes de séquence

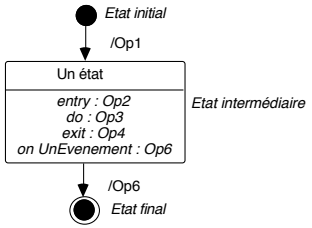


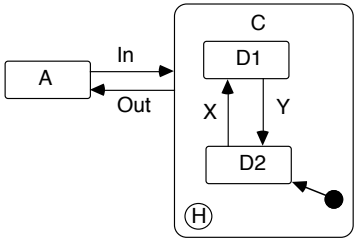
Les diagrammes d'états-transitions

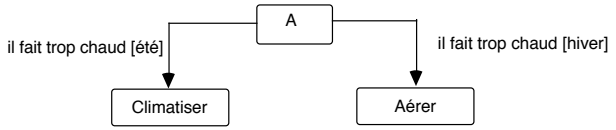
• issus de la représentation des automates (statecharts de David HAREL) :



Exemples :

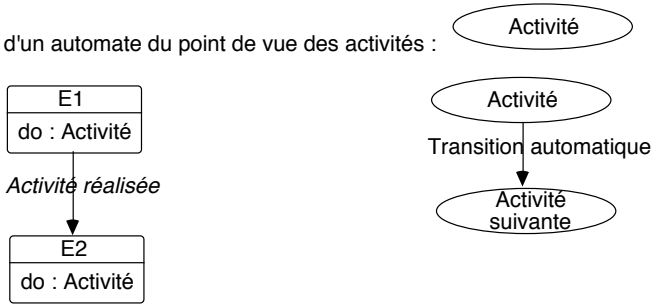




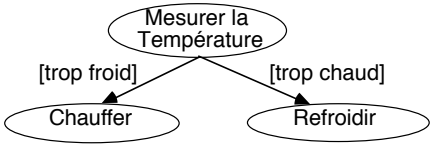


Les diagrammes d'activités

• représentation d'un automate du point de vue des activités :

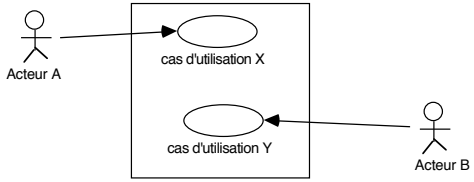


Exemple :



Les diagrammes de cas d'utilisation

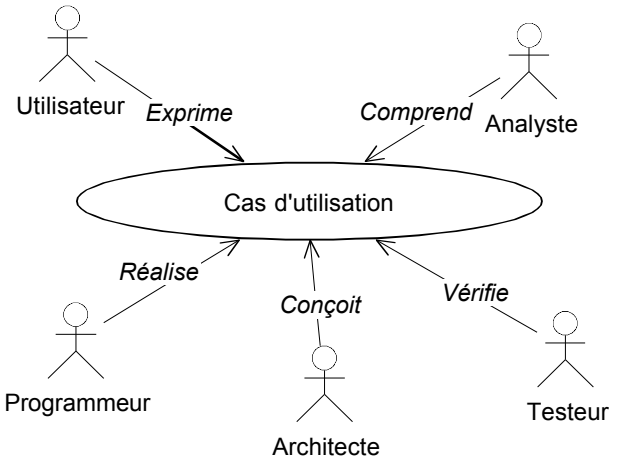
- **Origine :**
  - formalisés par Ivar Jacobson = "**Use Case**" : Object-Oriented Software Engineering (Addison-Wesley, 1992)
  - expression du **comportement du système** (actions et de réactions), selon le **point de vue de l'utilisateur**
  - décrivent le **système** et les **relations** entre le système et **l'environnement**
- **Intérêt :**
  - constituent un moyen d'**exprimer les besoins** d'un système
  - **utilisés** par les **utilisateurs finaux** pour exprimer leur attentes et leur besoins
  - permettent d'**impliquer les utilisateurs** dès les premiers stades du développement
  - constituent une **base pour les tests fonctionnels**



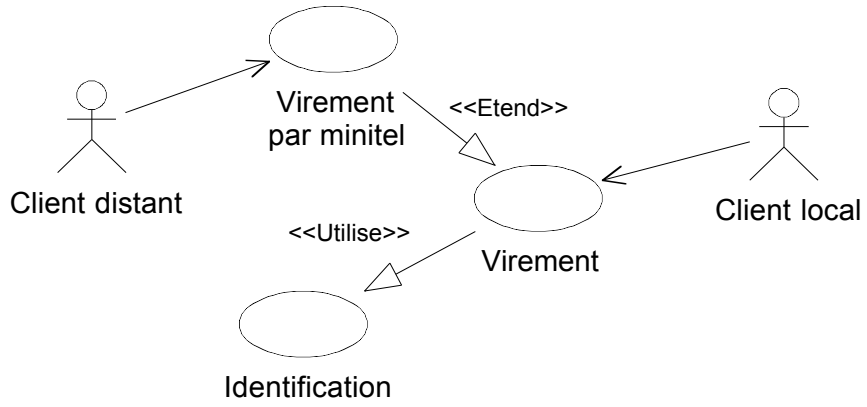
- **Acteur :**
  - = **personne** ou **système** qui **interagit** avec un système considéré, en échangeant de **l'information** (en entrée et en sortie)
  - sont **identifiés** en observant :
    - les utilisateurs directs du système, ceux qui sont responsable pour sa maintenance, ainsi que
    - les autres systèmes qui interagissent avec le système
- **Utilisateur :**
  - Un **acteur** représente un rôle joué par un **utilisateur** qui interagit avec le système
  - La même personne physique peut jouer le rôle de **plusieurs acteurs** (vendeur, client)
  - D'autre part, plusieurs personnes peuvent également jouer le **même rôle**, et donc agir comme le même acteur (tous les clients)

- **Acteurs et cas d'utilisations:**
  - Les cas d'utilisations représentent le dialogue entre l'acteur et le système de manière abstraite
  - Ensemble de scénarios au sein d'une description unique
  - Les cas d'utilisations doivent être vus comme des classes de scénarios
- **Détermination des cas d'utilisations :**
  - Quelles sont les tâches de l'acteur ?
  - Quelles informations l'acteur doit-il créer, sauvegarder, modifier, détruire ou simplement lire ?
  - L'acteur devra-t-il informer le système de changements externes ?
  - Le système devra-t-il informer l'acteur de conditions internes au système ?

... leurs usage dans un projet ...

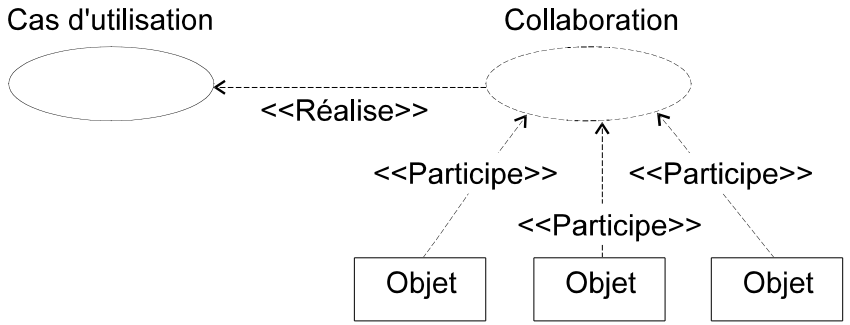


Exemple de relations entre cas d'utilisation :

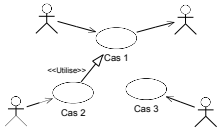


Les diagrammes de cas d'utilisation

Transition vers les objets :

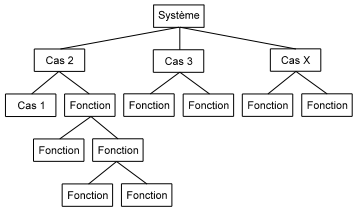


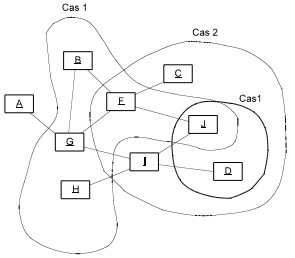
Les diagrammes de cas d'utilisation: le "virage" vers l'objet



Décomposition structurée

Décomposition objet





Les diagrammes de composants

• Représentation des éléments de réalisation :