Modèles et diagrammes de UML2
(Unified Modelling language)

Bernard ESPINASSE
Professeur à l'Université d'Aix-Marseille

2009

1. **Introduction :** Origine, objectifs, évolution, vues, modèles, diagrammes
2. **Diagrammes de cas d'utilisation**
3. **Diagrammes de classes**
4. **Diagrammes d'objets**
5. **Diagrammes d'interaction**
6. **Diagrammes d'états-transitions**
7. **Diagrammes d'activité**
8. **Diagrammes de composants**

1. Introduction

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 en précisant : le **comportement** des objets, les **différents états** par lesquels ils passent et les **événements** qui déclenchent ces changements d'états.
  - aspect **fonctionnel** : précise les fonctions réalisées par les objets par l'intermédiaire des méthodes

Méthodes orientées objet les plus connues :

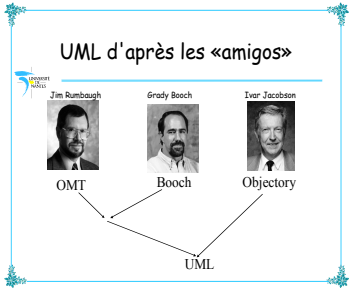
- **OMT** (Rumbaugh et al. 1995) puis **UML** (Rumbaugh et al. 1998)
- **OOA** (Object Oriented Analysis - Coad & Yourdon 1992)
- **BOOCH** (Booch 1991)
- **OOA** (Object Oriented Analysis - Shlaer & Mellor 1992)
- **Objectory-OOSE** (Jacobson & al. 19XX)
- **HOOD**
- **...**

De l'unification des méthodes objet à UML

- 1995 unification 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, ...) …



Les objectifs de UML

Objectifs poursuivis :

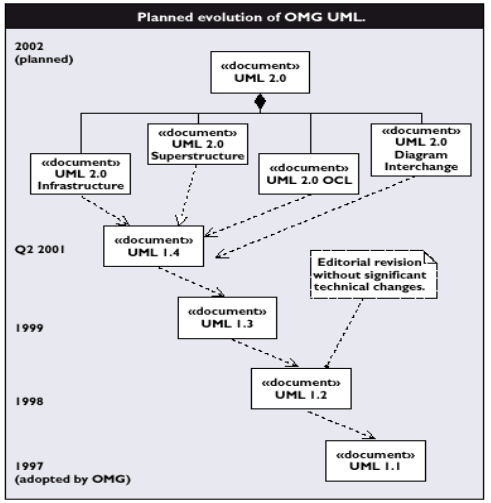
- **représenter des systèmes** entiers (au dela du seul logiciel) par des **concepts objets**,
- créer un **langage de modélisation** utilisable par les humains et les machines.
- établir un **couplage explicite** entre les **concepts** et les **artefacts exécutables**

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/

De UML 1.5 à UML 2.0

- ***UML 2.0 Infrastructure RFP*** **-- A UML 2.0 RFP issued September 15, 2000 that is primarily concerned with architectural alignment, restructuring and extension mechanisms.**
- ***UML 2.0 Superstructure RFP*** **-- A UML 2.0 RFP issued September 15, 2000 that is primarily concerned with the refinement and extension of UML 1.x semantics and notation.**
- ***UML 2.0 OCL RFP*** **-- A UML 2.0 RFP issued September 15, 2000 that is primarily concerned with with defining an OCL metamodel.**
- ***UML 2.0 Diagram Interchange RFP*** **-- A UML 2.0 RFP issued March 2, 2001 that is primarily concerned with with defining metamodel for diagram interchange using the XMI facility.**



Les Modèles d'UML

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

- UML défini **9 diagrammes** pour élaborer ces modèles
  - **Vues statiques du système**
  - **Vues dynamiques du système**

Diagrammes d'UML (1)

Vues statiques du système :

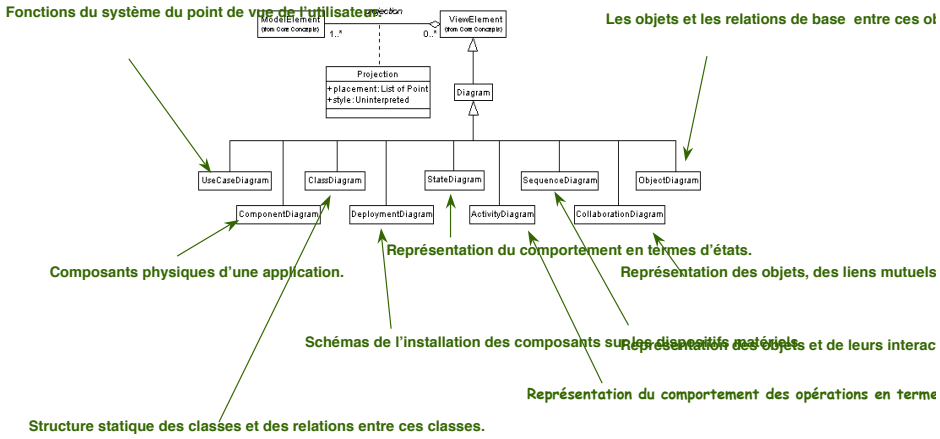
- **diag. de cas d'utilisation :** représent. des fonctions du système du point de vue de l'utilisateur,
- **diag. d'objets** : représent. des objets et leurs relations,
- **diag. de classes** : représent. de sa structure statique en termes de classes et relations,
- **diag. de composants** : représentation des composants physiques d'une application,
- **diag. de déploiement :** déploiement des composants sur les dispositifs materiels,

Vues dynamiques du système :

- **diag. d'interaction** :
- **diag.de collaboration :** représent. spatiale des objets, des liens et des interactions,
- **diag.de séquence :** représent. temporelle des objets et leurs interactions,
- **diag. d'états-transition** (Statecharts) : comportement d'une classe d'objet en termes d'états,
- **diag. d'activités** : représentation d'une opération en termes d'actions.

Diagrammes d'UML (2)

Vues multiples (aspects d'un système logiciel) :

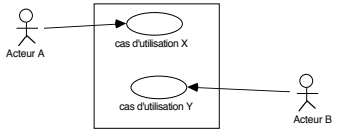


2. Diagrammes de Cas d'Utilisation

Les diagrammes de Cas d'Utilisation (Use Case Diagram)

Origine :

- Ivar Jacobson : "**Use Case**" in Object-Oriented Software Engineering (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 :

- permettent de **délimiter les frontières du système**
- constituent un moyen d'**exprimer les besoins** d'un système
- **utilisés** par les **utilisateurs finaux** pour exprimer leurs attentes et leurs besoins
- permettent d'**impliquer les utilisateurs** dès premiers stades du développement
- constituent une **base pour les tests fonctionnels**

Les diagrammes de cas d'utilisation

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)

Les diagrammes de cas d'utilisation

Acteurs et cas d'utilisations:

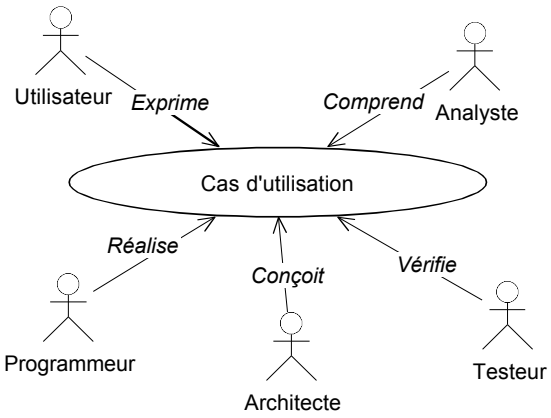
- Les cas d'utilisations représentent le dialogue entre l'acteur et le système de manière abstraite
- Ensemble de scénarios au sein d'une description unique
- Les cas d'utilisations doivent être vus comme des classes de scénarios

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 ?

Les diagrammes de cas d'utilisation

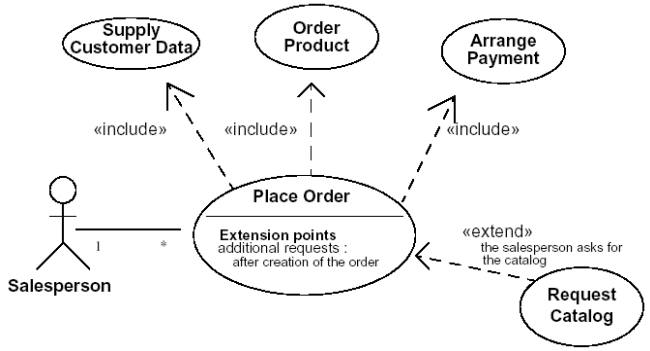
... leurs usage dans un projet ...



Les diagrammes de cas d'utilisation

Relations entre cas d'utilisation

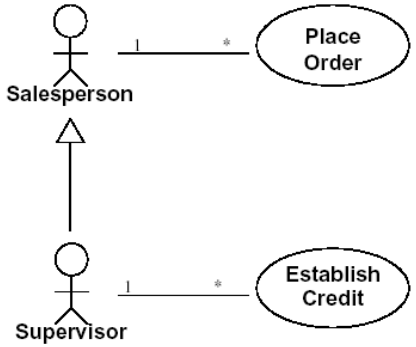
•Association, Extension, Utilisation, Généralisation :



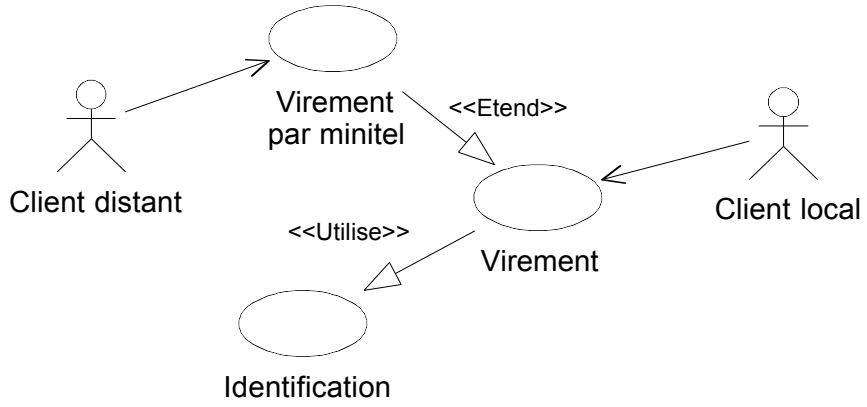
Les diagrammes de cas d'utilisation

Relations entre cas d'utilisation

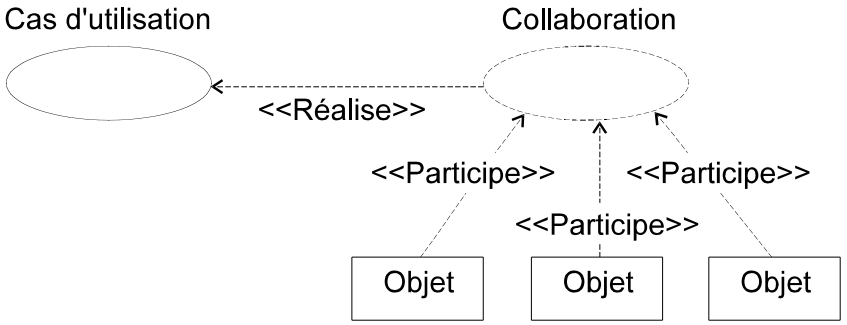
• Association, généralisation :



Exemple de relations entre cas d'utilisation :



Un cas d'utilisation réalisé par une collaboration d'objets :

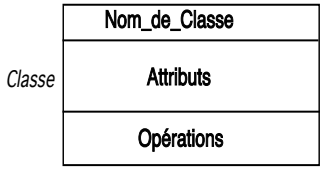


3. Diagrammes de Classes

- ils expriment structure statique du système en terme de classes et de relations entre ces classes
- ce sont des **extension du modèle Entité-Association** (« français ») par l'introduction de :
  - l'**agrégation**,
  - la **généralisation**
  - la spécification d'**opérations** et **contraintes** au niveau des entités, nommées ici "**Classes**"

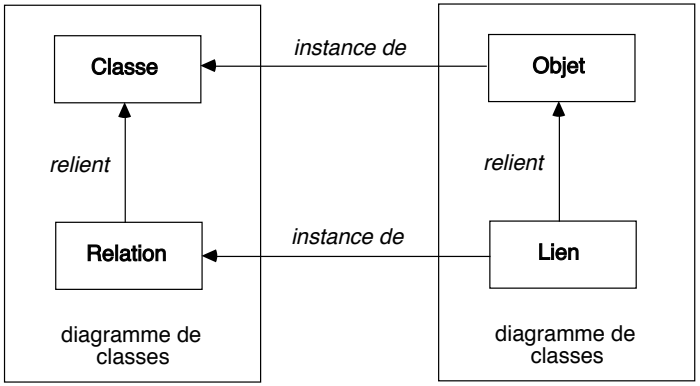
La classe :

- Une classe d'objets modélise un **ensemble d'objets** ayant :
  - des **caractéristiques** similaires (Attributs),
  - des **comportements** similaires (Opérations)
- Elle décrit le domaine de définition d'un ensemble d'objets
- C'est une description conceptuellement séparée en 2 parties :
  - La **spécification** d'une classe qui décrit le domaine de définition et les propriétés des instances de cette classe (type de donnée)
  - La **réalisation** qui décrit comment la spécification est réalisée
- **Conventions graphiques adoptées** : rectangle à 3 compartiments



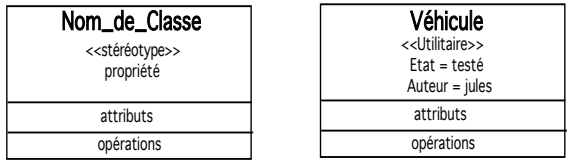
Classes / Objets

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



Classes : stéréotypes et propriétés

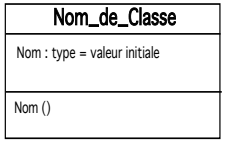
- une classe peut contenir un des **stéréotypes** suivants :
  - **<<signal>>** : occurrence remarquable déclenchant une transaction dans un automate,
  - **<<interface>>** : description des opérations visibles,
  - **<<métaclasse>>** : classe d'une classe,
  - **<<utilitaire>>** : classe réduite au 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

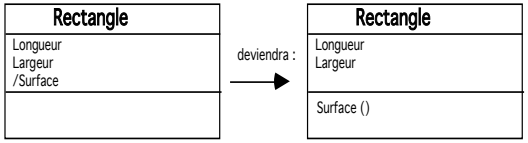
- **attributs** et **opérations** d'une classe peuvent être **explicités** :
  - syntaxe d'un **attribut** :

Nom_attribut : Type_Attribut = Valeur_Initiale



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

/surface

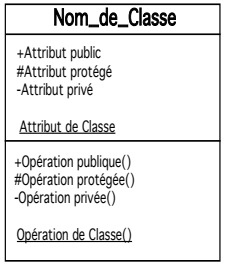


Classes : Attributs et Opérations

Syntaxe d'une **opération** :

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

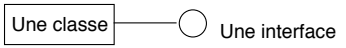
- **3 niveaux de visibilité** des **attributs** et des **opérations** :
  - **public :** élément visible à tous les clients de la classe (+),
  - **protégé :** élément visible aux sous-classes de la classe (#),
  - **privé :** élément visible à la classe seule (–).



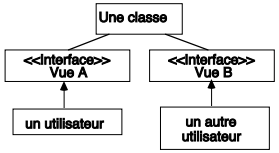
- **attributs et opération soulignés : visibles globalement** dans **toute la portée lexicale de la classe**

Classe : les Interfaces

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

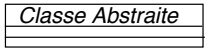


- peuvent être représentées au moyen de **classes stéréotypés** :



Classes abstraites

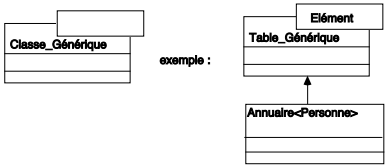
- pas directement instanciables (ne donnent pas naissance à des objets)
- sont utilisées pour une spécifications de typage plus générale pour manipuler les objets instances d'une (ou plusieurs) de leurs sous-classes :



Classes paramétrables

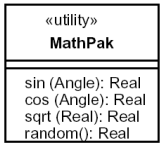
= **modèles de classes** (classes générique de Eiffel ou template de C++)

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

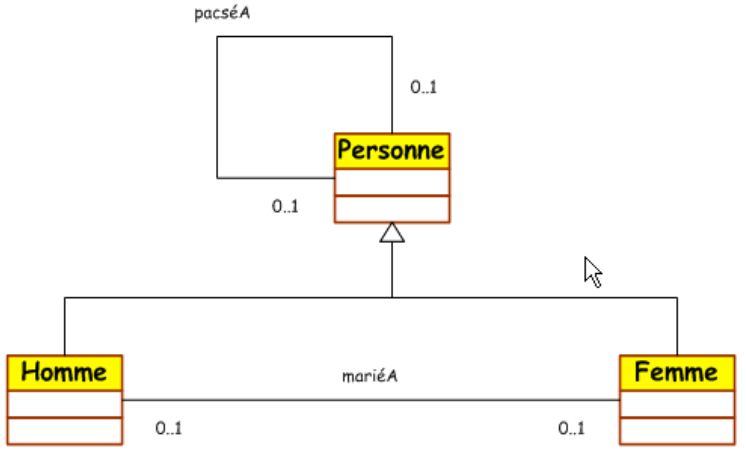


Classes utilitaires

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



Associations entre classes

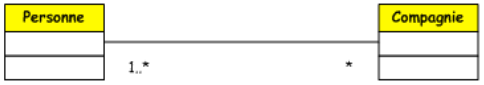


Associations binaires : nommage, rôle et multiplicités

• **Nommage :**

• **Rôles :**

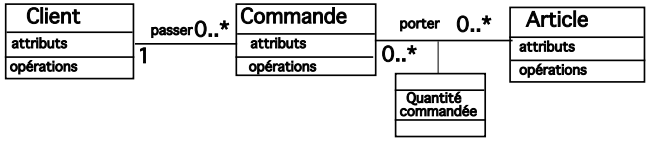
• **Multiplicités** :



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)

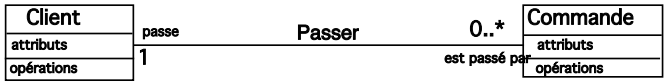


*l'association "porter" est **attribuée** (attribut "Quantité commandée")*

• 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**"

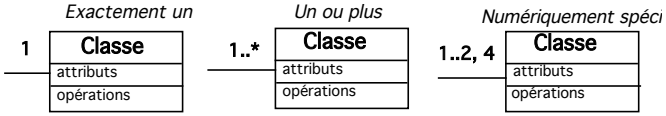
• le nommage des rôles est possible (lourd si cummulé à celui des associations)



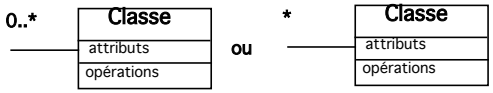
Multiplicités

• caractérisent les **liens associatifs**

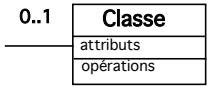
• spécification des multiplicités à l'autre bout de la ligne (**inverse cardinalités Merise**)



Plusieurs (zéro ou plus)



Optionnel (zéro ou un)

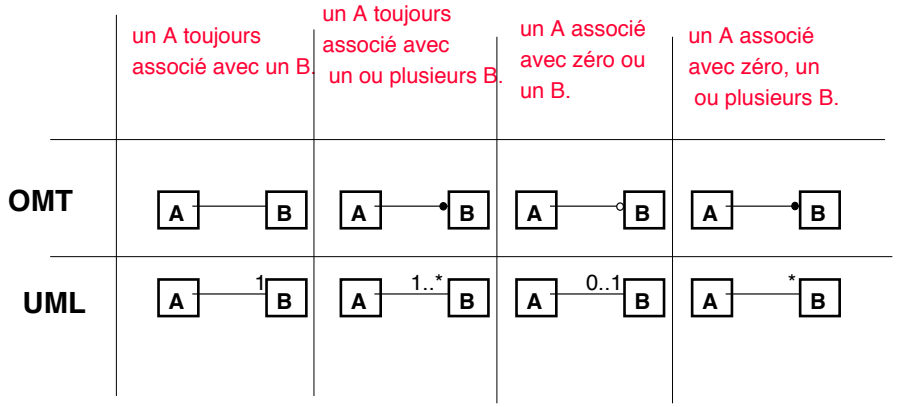


Remarque :

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

Multiplicités

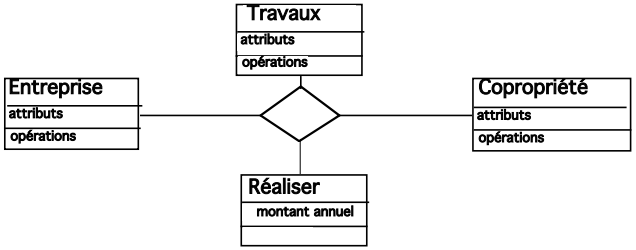
Comparaison OMT / UML :



Associations n-aires

2 façons de représenter une association n-aire :

• 1) par une **classe-association** ordinaire (avec losange):

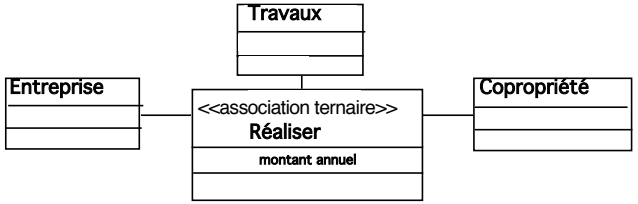


*ici **classe-association** REALISER reliée par un losange*
elle possède un attribut propre "montant annuel"

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

Associations n-aires

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

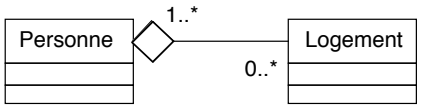


*ici **classe-association** Réaliser reliée par un losange*
elle possède un attribut propre "montant annuel"

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

L'Agrégation

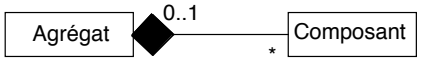
• association **non symétrique** dans laquelle une extrémité joue un rôle prédominant sur l'autre :
  • **une classe fait partie d'une autre classe,**
  • les **valeurs d'attributs** d'une classe **se propagent** dans les valeurs d'attributs d'une autre classe
  • une **action** sur une classe **implique** 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**



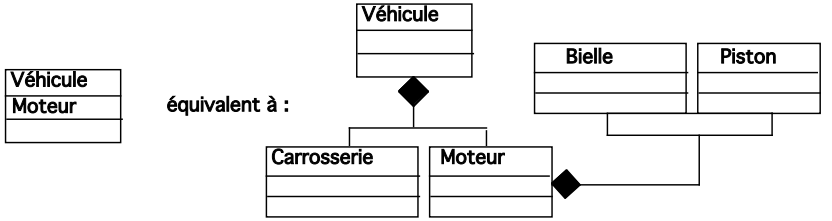
des Personnes peuvent être copropriétaires des mêmes Logements

La Composition

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



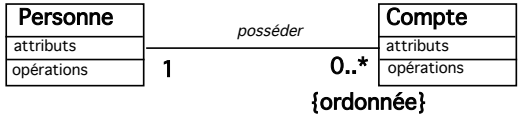
• implique une contrainte sur la valeur de la multiplicité du coté de l'agrégat
• composition et attributs sont **sémantiquement équivalents**:



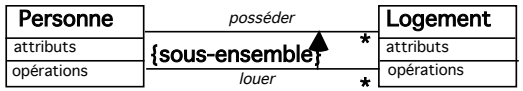
une classe Véhicule est composée par exemple de 2 autres classes, la Carrosserie et le Moteur qui a son tour est composé des classes Bielle et Piston

Contraintes sur les associations

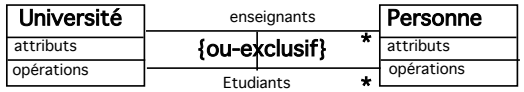
• **contrainte** `{ordonnée}` **:** une relation d'ordre décrit les objets placés dans la collection : l'ordre doit être maintenu,



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



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

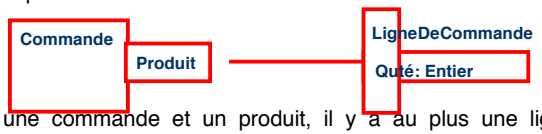


Association qualifiée

Une association qualifiée met en relation 2 classes sur la base un attribut spécifique appelé "clé" :

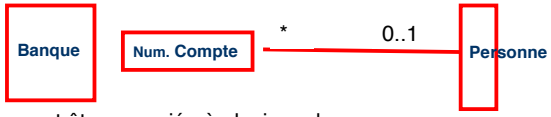
- cette clé est représentée sur le rôle de départ
- elle appartient pleinement à l'association et non aux classes associées

Exemple 1:



- Etant donnés une commande et un produit, il y a au plus une ligne de commande associée

Exemple 2 :



- Une personne peut être associée à plusieurs banques
- Etant donnés une banque et un numéro de compte, il y a au plus une personne ayant ce numéro de compte à cette banque

La Navigation ou la spécification des chemins

• 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**

La Navigation ou la spécification des chemins

Exemple :

l'association +**propriétaire** entre les classes **Utilisateur** et **MotDePasse** est seulement navigable de **Utilisateur** vers **MotDePasse**

Ainsi :

- étant donné un utilisateur, on désire pouvoir accéder à ses mots de passe
- étant donné un mot de passe, on ne souhaite pas pouvoir accéder à l'utilisateur correspondant

Syntaxe de spécification des chemins :

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

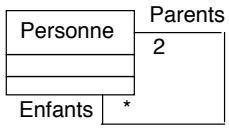
La Navigation : règle R1

```
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

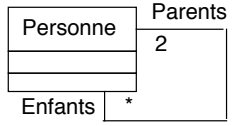
La Navigation : règle R2

```
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

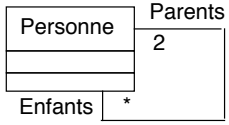
La Navigation : règle R3

```
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

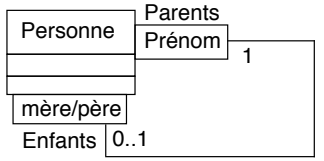
La Navigation : règle R4

```
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]}
```

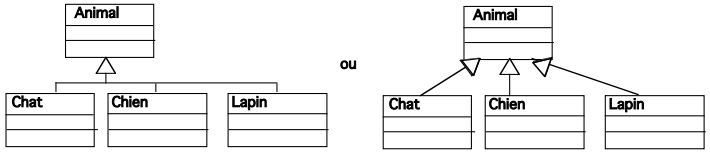
alors :

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

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

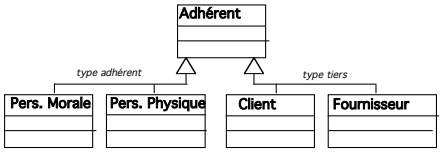
La Généralisation

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



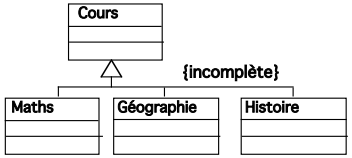
Chat, Chien, Lapin sont dites sous-classes de la super-classe Animal

- 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 (contr. **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 :



4. Diagrammes d'objets

3 types de diagrammes avec des objets

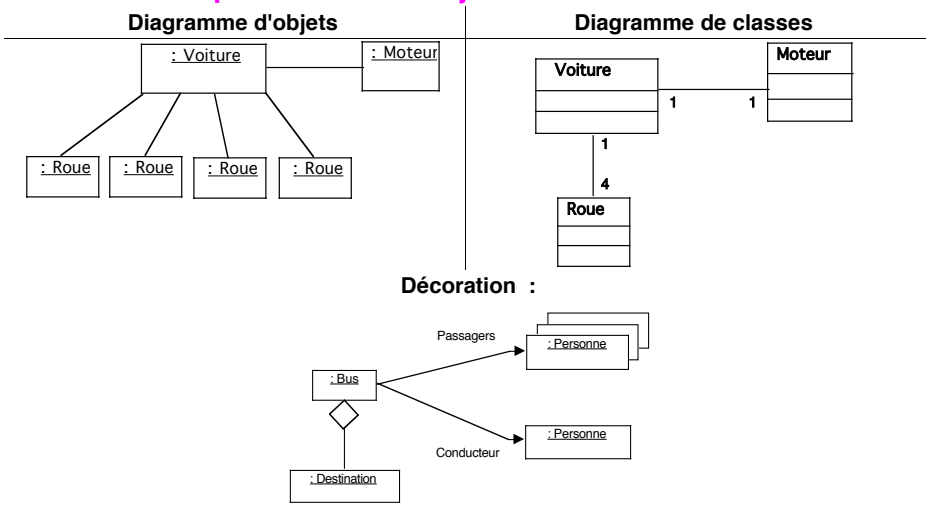
- **Diagrammes d'objets (point de vue statique)**
- **Diagrammes d'interaction (point de vue dynamique) :**
  - Diagramme de séquence
  - Diagramme de collaboration
- **2 niveaux de représentation des collaborations :**
  - Niveau Spécification (des rôles et des messages)
  - Niveau Instance (des instances et des stimuli)

Les diagrammes d'objets

- représentent un ensemble **d'objets** et leurs **liens**
- sont des **vues statiques des instances** des éléments qui apparaissent dans les **diagrammes de classes**
- Ils présentent la **vue de conception d'un système**, exactement comme les diagrammes de classes, mais à partir de cas réels ou de prototypes.
- est **une instance d'une diagramme de classes**
- les **diagramme de classes peuvent aussi contenir des objets** (objets de classes, au sens variables de classes)

Les diagrammes d'objets

Représentation d'objets et de leurs liens :



5. Diagrammes d'interaction

- Diagrammes de séquences
- Diagrammes de collaboration

Les diagrammes d'interaction (interaction diagrams)

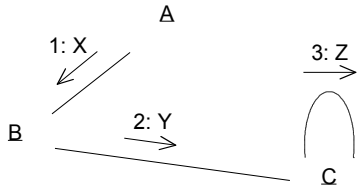
- représentent une interaction, c.a.d. un ensemble d'objets et leurs relations, y compris les messages qu'ils peuvent échanger. Ils présentent une vue dynamique d'un système.
- 2 types de diagrammes :
  - **les diagrammes de séquence** : ils mettent l'accent sur le **classement chronologique des messages** de collaboration d'instances
  - **les diagrammes de collaboration d'instances : ils mettent l'accent sur l'organisation structurelle des éléments qui envoient et reçoivent des messages**.

Les diagrammes de séquence et les diagrammes de collaboration d'instances sont **isomorphes** : c.a.d. l'un peut être transformé en l'autre.

Les diagrammes de collaboration (collaboration diagrams)

- concernent des **objets** reliés par des **liens** et qui se connaissent dans une situation donnée
- **Représentation spatiale** d'une **interaction** :
  - Mise en avant de la structure
  - Représentation des structures complexes (récursives par ex)
- 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**

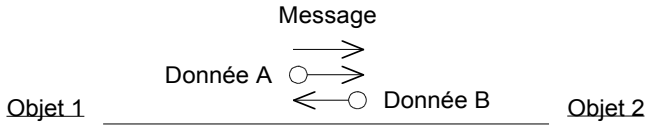
Exemple :



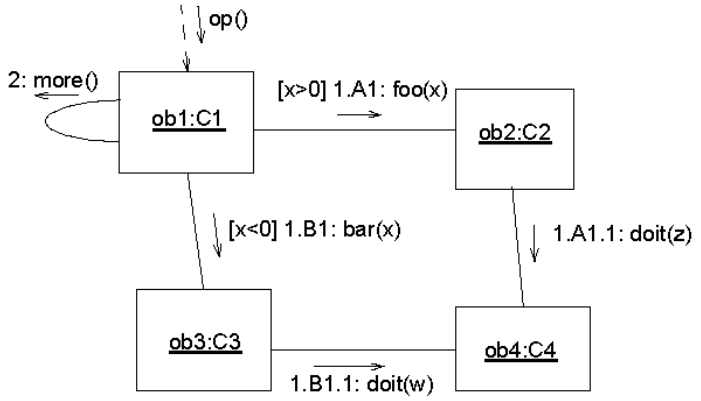
- *Un objet **A** envoie un message **X** à un objet **B**,*
- *puis l'objet **B** envoie un message **Y** à un objet **C**,*
- *et enfin **C** s'envoie un message **Z**.*

Les diagrammes de collaboration : messages

- unité de communication entre rôles
- Regroupe les flots de contrôle et les flots de données
- Représentation des messages :

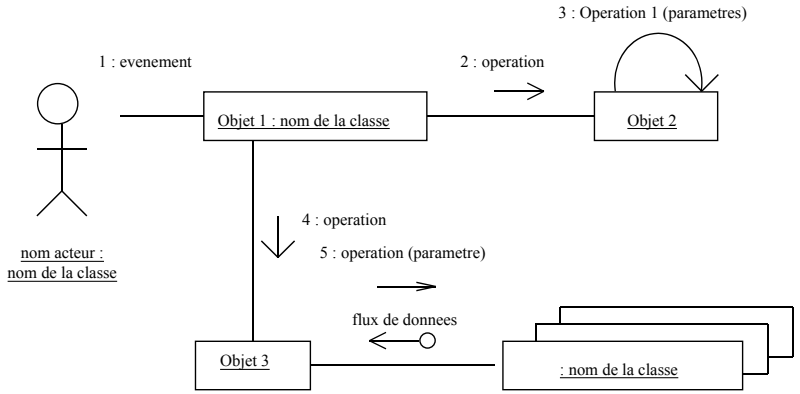


Les diagrammes de collaboration avec condition



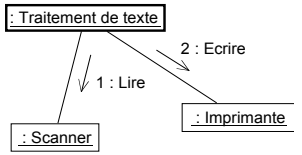
Les diagrammes de collaboration

Exemple 1 :

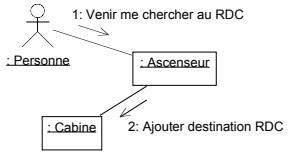


Les diagrammes de collaboration : exemples

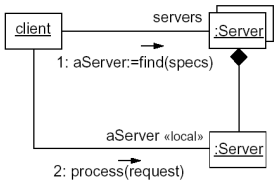
Entre objets actifs



Entre objets actifs avec acteur

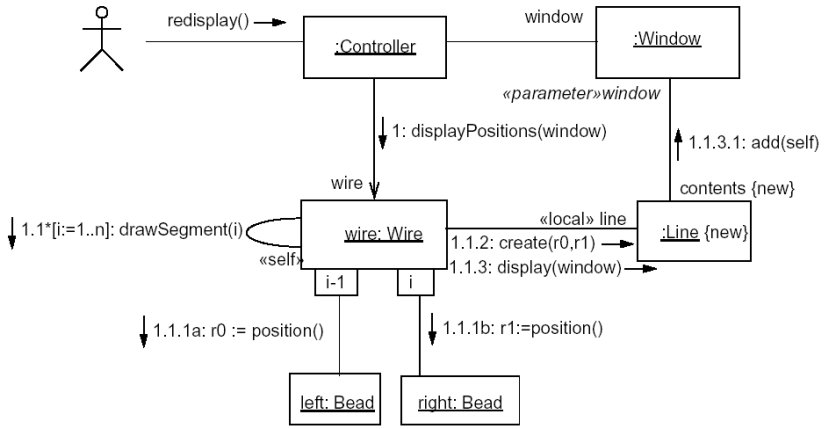


Multi-objets



Les diagrammes de collaboration (niveau instances)

Exemple :



Les diagrammes de collaboration : règles de nommage

ObjectName '/' ClassifierRoleName ':' ClassifierName

syntax	explanation
: C	un-named Instance originating from the Classifier C
/ R	un-named Instance playing the role R
/ R : C	un-named Instance originating from the Classifier C playing the role R
O / R	an Instance named O playing the role R
O : C	an Instance named O originating from the Classifier C
O / R : C	an Instance named O originating from the Classifier C playing the role R
O	an Instance named O
/ R	a role named R
: C	an un-named role with the *base* Classifier C
/ R : C	a role named R with the *base* Classifier C

Les diagrammes de séquences (sequence diagrams)

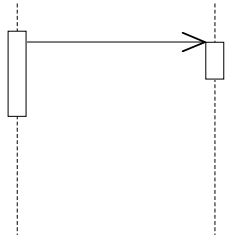
- Représentation des interactions entre acteurs et objets
- Vision temporelle d'une interaction :
  - Chaque objet est symbolisé par une barre verticale
  - Le temps s'écoule de haut en bas, de sorte que la numérotation des messages est optionnelle.
  - Diagramme dual du diagramme de collaboration
- Souvent utilisé pour représenter une instance de cas d'utilisation
- De manière plus générale, représentation temporelle d'une interaction
  - Bien adapté pour de longues séquences
  - Ne visualise pas les liens
  - Complémentaire du diagramme de collaboration

Les diagrammes de séquences

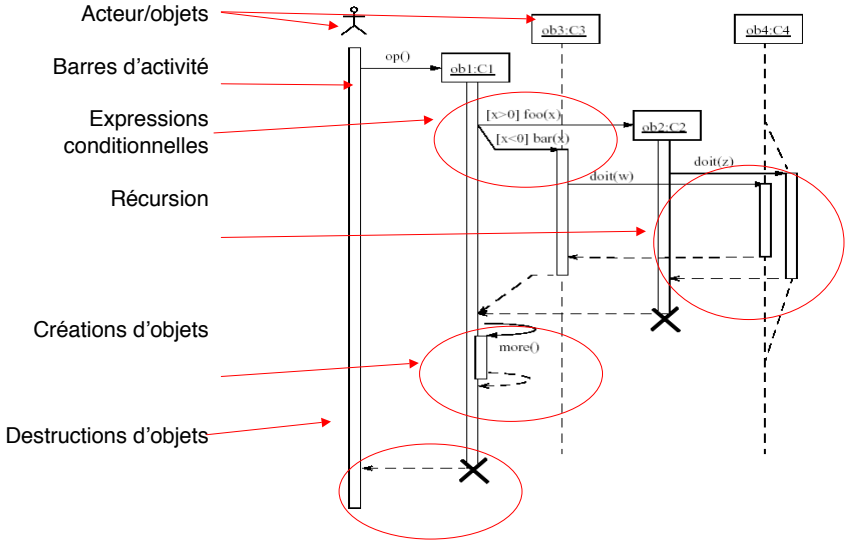
- L'accent est mis sur la communication, au détriment de la structure spatiale
- Chaque objet est représenté par une barre verticale
- Le temps s'écoule de haut en bas, de sorte que la numérotation des messages est optionnelle

Exemple:

Un client Un serveur

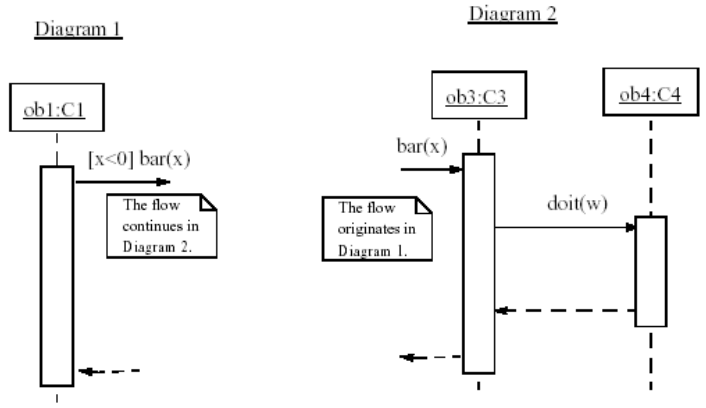


Les diagrammes de séquences : syntaxe générale



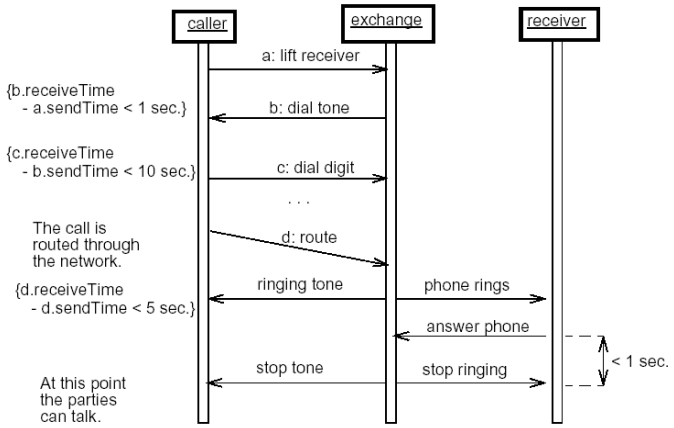
Les diagrammes de séquences : sous-séquence

Représentation d'une sous-séquence :



Les diagrammes de séquences : exemples

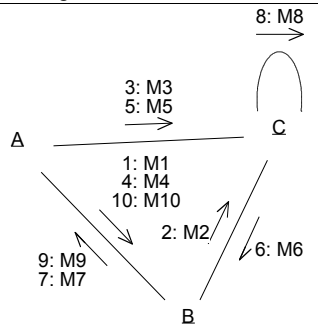
Avec des objets actifs :



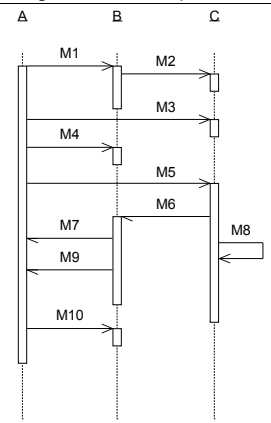
Diagrammes de collaboration / Diagrammes de séquence

Comparaison :

Diagrammes de collaboration



Diagrammes de séquence



6. Diagrammes d'états-transitions

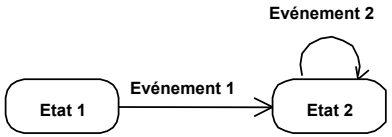
Les diagrammes d'état-transitions (statechart diagrams)

- ils présentent la **vue dynamique d'un système**
- ils sont particulièrement importants dans la **modélisation du comportement** d'une **interface**, d'une **classe** ou d'une **collaboration**
- le comportement d'un objet est **ordonnancé par les événements**, ce qui est particulièrement utile dans la modélisation des **systèmes réactifs**.
- ce sont des **automates à états**, composés d'**états**, de **transitions**, d'**événements** et d'**activités**.
  - **Etat** : Condition dans laquelle se trouve un objet
  - **Transition** : Chemin entre deux états
  - **Evénement** : Occurrence qui survient dans le domaine

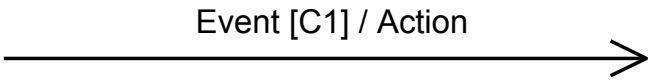
Les diagrammes d'états-transitions

Concepts majeurs

Evénement et états :



Evénement et action :

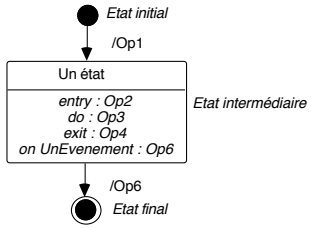


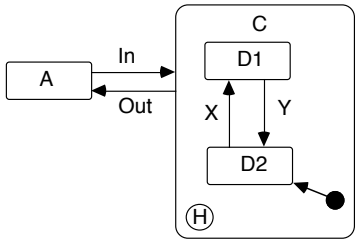
L'action est exécutée instantanément

Les diagrammes d'états-transitions : syntaxe

Syntaxe issue de la représentation des automates (statecharts de David HAREL) :

Exemples :





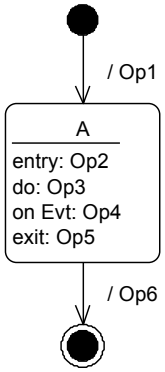
Les diagrammes d'états-transitions : syntaxe

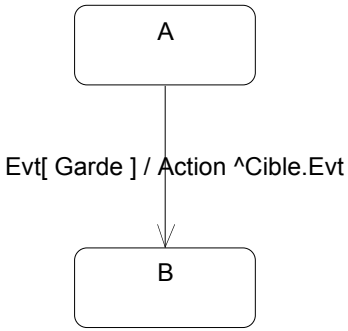
Notation des Statecharts de David HAREL autorisant :

- **gardes sur transitions,**
- **propagation de transition,**
- **actions sur transitions,**
- **actions sur entrée d'état,**
- **activités apparaissant tant que l'état est actif,**
- **actions sur sortie d'état,**
- **imbrication d'états,**
- **concurrence.**

Les diagrammes d'états-transitions : syntaxe

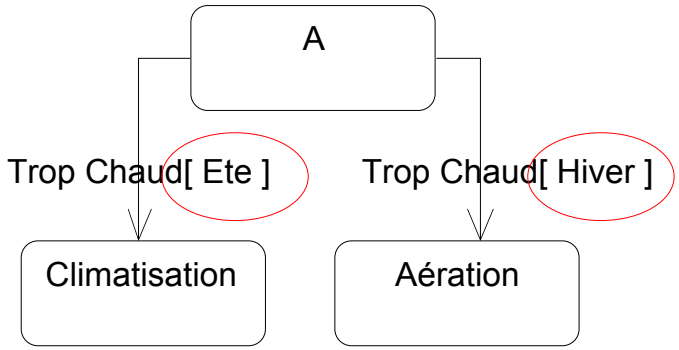
Les actions et les activités :





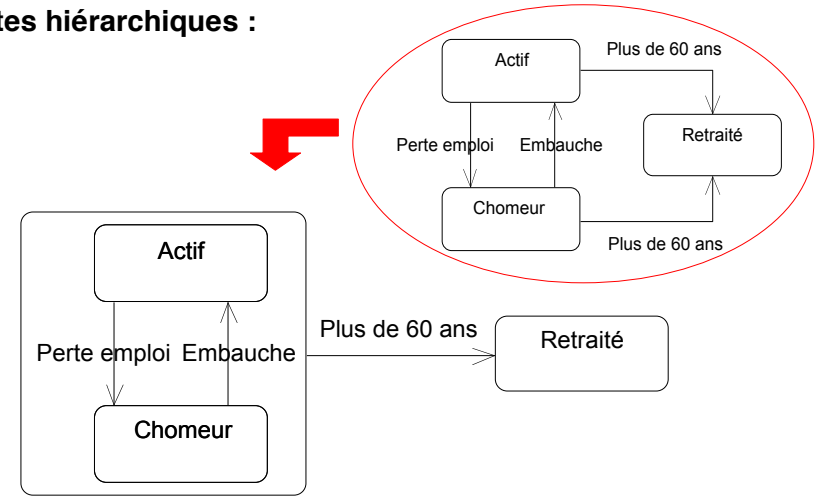
Les diagrammes d'états-transitions : les gardes

Les gardes :



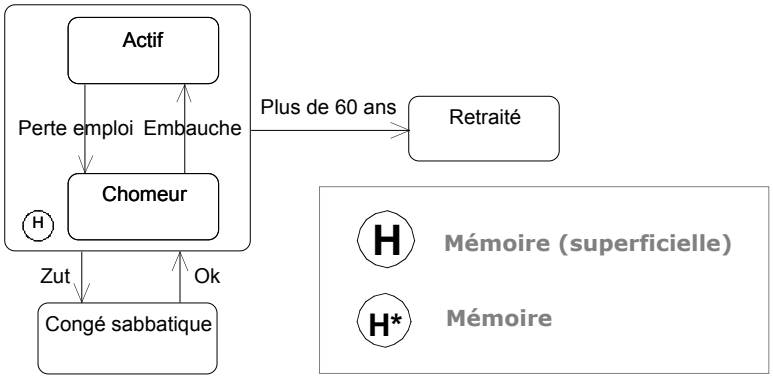
Les diagrammes d'états-transitions : Généralisation d'états

Automates hiérarchiques :



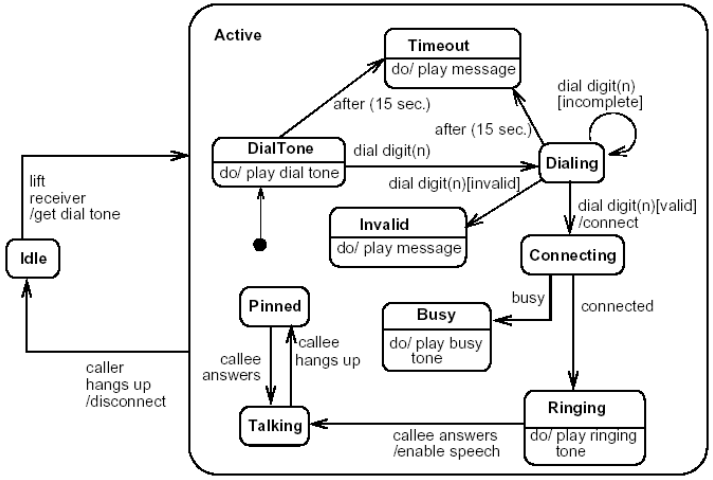
Les diagrammes d'états-transitions : mémoire

Automates à mémoire :



Les diagrammes d'états-transitions : exemple

Exemple :



7. Diagrammes d'activités

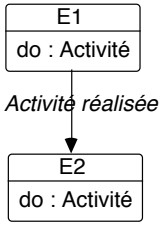
Les diagrammes d'activités (activity diagrams)

- ils sont un type particulier de diagrammes d'état décrivant la succession des activités au sein d'un système.
- Ils présentent la vue dynamique d'un système,
- Ils sont particulièrement importants dans la modélisation de la fonction d'un système et mettent l'accent sur le flot de contrôle entre objets.
- Le statut relatif des diagrammes d'activités et d'état est un changement de UML 1.5 en UML 2.0
- Utilisés pour décrire le comportement interne :
  - D'une **classe**
  - D'une **méthode**
  - D'un **cas d'utilisation**

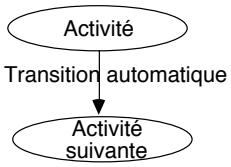
Les diagrammes d'activités

Représentation d'un automate du point de vue des activités :

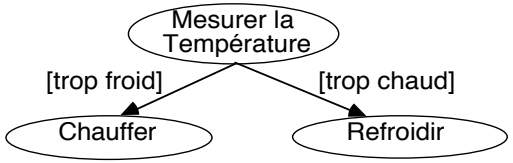
Point de vue des états :



Point de vue des activités :

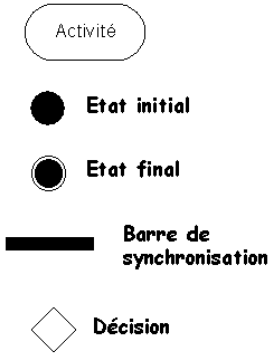


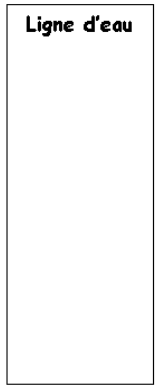
Exemple :



Les diagrammes d'activités : syntaxe

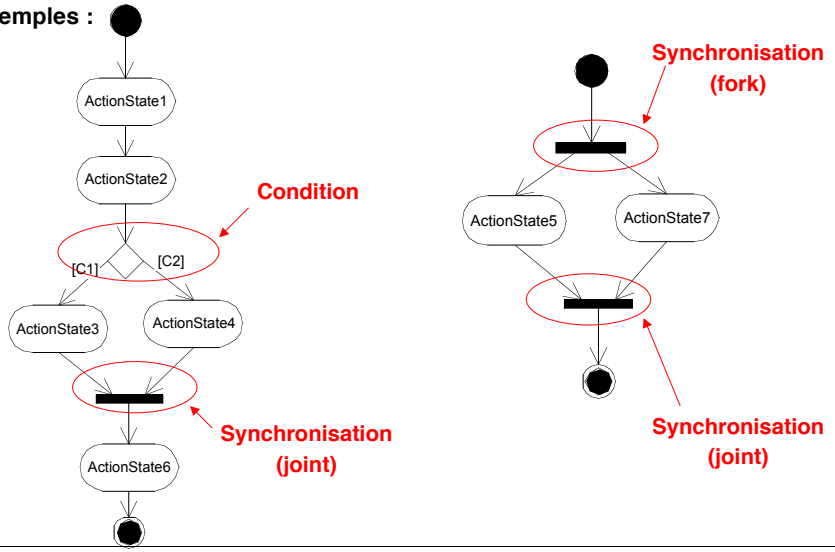
Syntaxe d'un type particulier de diagrammes d'état décrivant la succession des activités au sein d'un système :





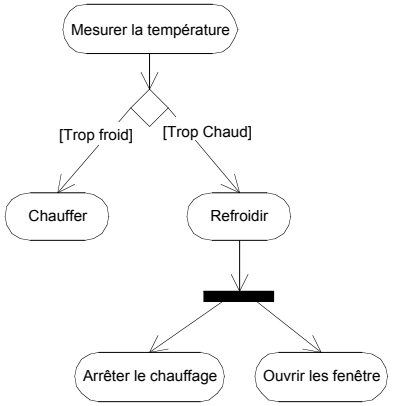
Les diagrammes d'activités : exemples 1

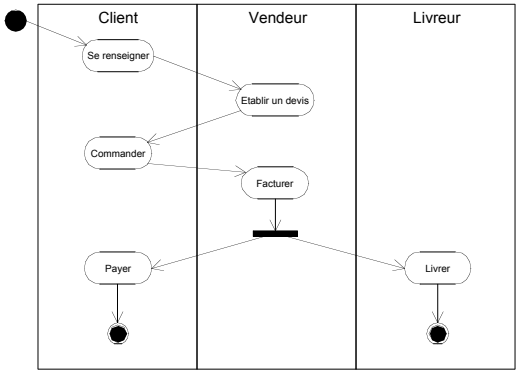
Exemples :



Les diagrammes d'activités : exemples 1

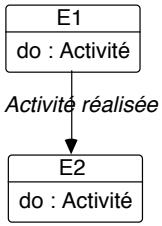
Exemples :

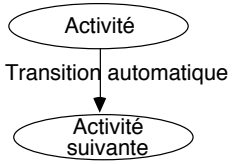




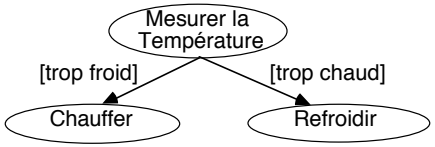
Les diagrammes d'activités

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





Exemple :



8. Diagrammes de composants

Diagrammes de composants (component diagrams)

- Ils représentent **l'organisation** et les **dépendances** au sein d'un **ensemble de composants**
- Ils présentent **la vue d'implémentation statique d'un système**
- Ils sont **liés aux diagrammes de classe** dans le sens où un composant correspond généralement à une ou plusieurs classes, interfaces ou collaborations

Les diagrammes de composants

Représentation de la structure du code :

- Fichiers source, binaires, exécutables, DLL
- Document d'organisation

Partitionnement de l'espace de réalisation :

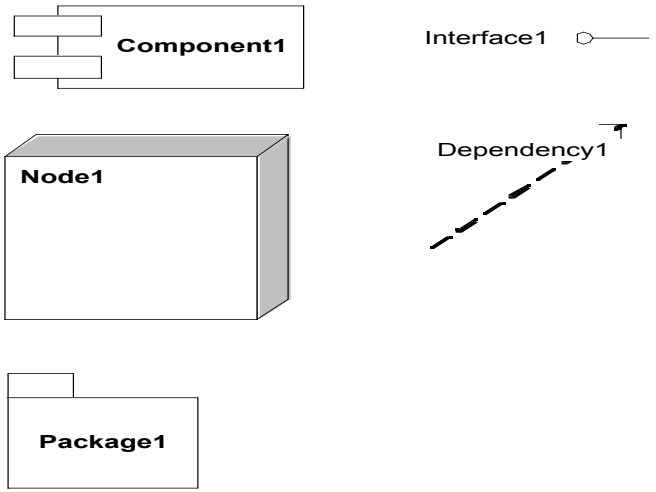
- Composant = unité de réalisation distribuable

Représentation de la structure statique :

- Des composants génériques reliés par des relations de dépendances
- Les composants ayant une identité à l'exécution se représentent dans des diagrammes de déploiement

Les diagrammes de composants : syntaxe

Syntaxe :

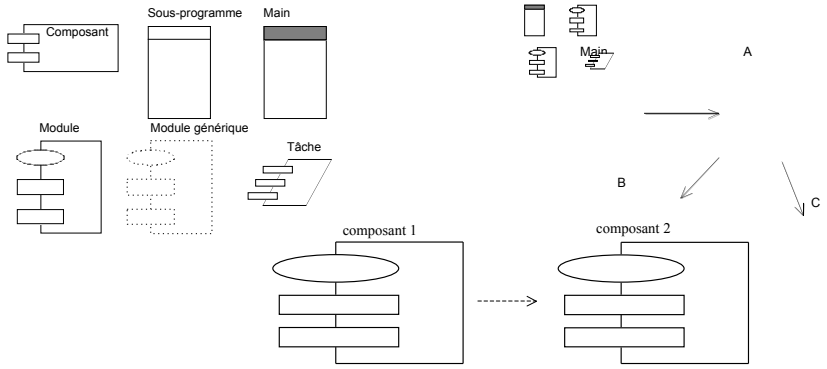


Les diagrammes de composants : relation de dépendance

Une relation d'obsolescence entre 2 composants :

- Flèche avec trait pointillé
- Concerne les éléments du modèles, pas forcément d'instance au runtime

4 stéréotypes prédéfinis : Trace, refinement, usage, binding



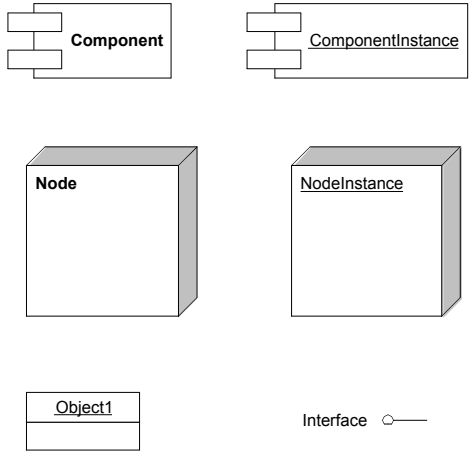
8. Diagrammes de déploiement

Les diagrammes de déploiement

- Ils représentent la **configuration des nœuds de processus en phase d'exécution ainsi que les composants qui y résident**
- Ils présentent la vue de déploiement statique d'une architecture
- Ils sont **liés aux diagrammes de composants**, dans le sens où un nœud renferme généralement un ou plusieurs composants.
- **Représentation de la structure d'un système lors de son exécution**
- **Relation entre composants logiciels et matériels**
- **Distributions des composants sur les processeurs**
- **Les composants n'ont pas d'existence propre à l'exécution**

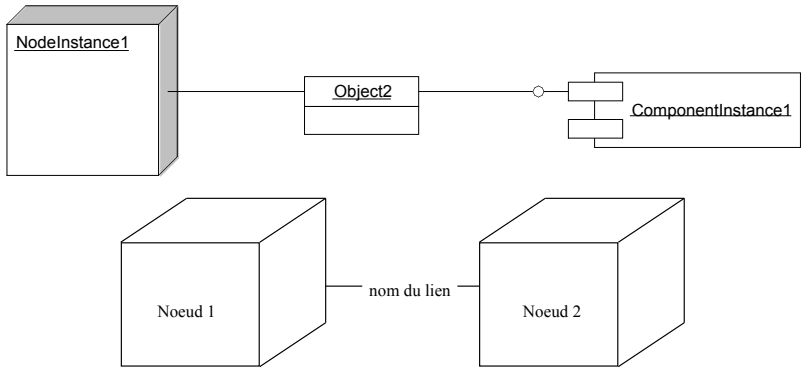
Les diagrammes de déploiement

Syntaxe :



Les diagrammes de déploiement

Syntaxe :



Les diagrammes de déploiement

Syntaxe :