Stéphane Pelle

poste 3159

ECOLE NATIONALE DES SCIENCES GEOGRAPHIQUES

14 mai 2001

Fiche de lecture du site "UML en français"

UML.free.fr

de Laurent PIECHOCKI consultant VALtech

Questions fréquemment posées à l'auteur du site http://uml.free.fr :

> Je suis enseignant et j'aimerais récupérer une partie / la totalité de ce site, pour mes cours. Cela est-il OK ?

Réponse :

> Je suis flatté de l'intérêt que vous portez à mon site... Récupérez tout ce que vous jugez intéressant !

Question :

> Je suis un professionnel et j'aimerais aspirer votre site en totalité / partie, pour le mettre à disposition sur notre intranet.

Réponse :

> Aucune objection. Merci cependant d'indiquer clairement que vous avez aspiré les pages concernées depuis http://uml.free.fr et que je ne répondrai à aucun mail à caractère professionnel.

Table des matières

1 Droits d'auteur ?

Le site "UML en français" dont les informations suivantes sont extraites indique ceci :

Auteur : Laurent Piechocki, consultant VALtech (mailto::laurent.piechocki@valtech.fr)

mini-FAQ : (Foire aux questions – "Frequently Asked Questions" --)

Question :

Je suis enseignant et j'aimerais récupérer une partie/la totalité de ce site, pour mes cours. Cela est-il OK ?

Réponse :

Je suis flatté de l'intérêt que vous portez à mon site... Récupérez tout ce que vous jugez intéressant !

Question :

Je suis un professionnel et j'aimerais aspirer votre site en totalité/partie, pour le mettre à disposition sur notre intranet.

Réponse :

Aucune objection. Merci cependant d'indiquer clairement que vous avez aspiré les pages concernées depuis http://uml.free.fr et que je ne répondrai à aucun mail à caractère professionnel.

Question :

Existe-t-il un fichier PDF, Word ou Powerpoint qui contient les pages du cours UML de ce site ?

Réponse :

Non, désolé...

2 Extraits des pages "UML en français"

UML : fusion de OMT, BOOCH et OOSE (années 90). Fin 1997, UML est devenu une norme OMG (Object Management Group).

Types Abstraits => Classes : Simula 1967
Approche objet : Smalltalk 1976 (encapsulation, agrégation, héritage), C++ années 80, Eiffel, Objective C et Loops…

"**Premier hic** : l'approche objet est moins intuitive que l'approche fonctionnelle. Malgré les apparences, il est plus naturel pour l'esprit humain de décomposer un problème informatique sous forme d'une hiérarchie de fonctions atomiques et de données, qu'en terme d'objets et d'interaction entre ces objets."

"Or les langages orientés objet ne sont que des outils qui proposent une manière particulière d'implémenter certains concepts objet. Ils ne valident en rien l'utilisation de ces moyens techniques pour concevoir un système conforme à la philosophie objet."

"Pour remédier à ces inconvénients majeurs de l'approche objet, il nous faut donc :

1) un **langage** (pour s'exprimer clairement à l'aide des concepts objets), qui doit permettre de

- représenter des concepts abstraits (graphiquement par exemple),
- limiter les ambiguïtés (parler un langage commun, au vocabulaire précis, indépendant des langages orientés objet),
- faciliter l'analyse (simplifier la comparaison et l'évaluation de solutions).

2) une **démarche d'analyse et de conception objet**, pour

- ne pas effectuer une analyse fonctionnelle et se contenter d'une implémentation objet, mais penser objet dès le départ, définir les vues qui permettent de décrire tous les aspects d'un système avec des concepts objets.

En d'autres termes : il faut disposer d'un outil qui donne une dimension méthodologique à l'approche objet et qui permette de mieux maîtriser sa richesse."

"La prise de conscience de l'importance d'une méthode spécifiquement objet ("comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements, mais sur les deux"), ne date pas d'hier. Plus de 50 méthodes objet sont apparues durant le milieu des années 90 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE...). Aucune ne s'est réellement imposée."

"L'absence de consensus sur une méthode d'analyse objet a longtemps freiné l'essor des technologies objet. Ce n'est que récemment que les grands acteurs du monde informatique ont pris conscience de ce problème. L'unification et la normalisation des méthodes objet dominantes (OMT, Booch et OOSE) ne datent que de 1995. UML est le fruit de cette fusion. UML, ainsi que les méthodes dont il est issu, s'accordent sur un point : une analyse objet passe par une modélisation objet."

UML possède plusieurs facettes. C'est une norme, un langage de modélisation objet, un support de communication, un cadre méthodologique. UML est tout cela à la fois, ce qui semble d'ailleurs engendrer quelques confusions...
CORBA : Common Object Request Broker Architecture

IDL : Interface Definition Language)

Pour conduire une analyse objet cohérente, il ne faut pas directement penser en terme de pointeurs, d'attributs et de tableaux, mais en terme d'association, de propriétés et de cardinalités... Utiliser le langage de programmation comme support de conception ne revient bien souvent qu'à juxtaposer de manière fonctionnelle un ensemble de mécanismes d'implémentation, pour résoudre un problème qui nécessite en réalité une modélisation objet.

Toutes les dérives fonctionnelles de code objet ont pour origine le non respect des concepts de base de l'approche objet (encapsulation...) ou une utilisation détournée de ces concepts (héritage sans classification...). Ces dérives ne sont pas dues à de mauvaises techniques de programmation ; la racine du mal est bien plus profonde ! Bref, programmer en C++ ou en Java n'implique pas forcément concevoir objet...

UML comble une lacune importante des technologies objet. Il permet d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation. Il a été pensé pour servir de support à une analyse basée sur les concepts objet.

La notation graphique d'UML n'est que le support du langage. La véritable force d'UML, c'est qu'il repose sur un métamodèle. En d'autres termes : la puissance et l'intérêt d'UML, c'est qu'il normalise la sémantique des concepts qu'il véhicule !

UML est avant tout un support de communication performant, qui facilite la représentation et la compréhension de solutions objet :

- Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions.
- L'aspect formel de sa notation, limite les ambiguïtés et les incompréhensions.
- Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus, en font un langage universel.

Grâce au principe d'élaboration des modèles, UML permet de mieux maîtriser la part d'inconnu et d'incertitudes qui caractérisent les systèmes complexes. Mais cet aspect méthodologique d'UML ne doit pas vous induire en erreur. UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles !

Qualifier UML de "méthode objet" n'est donc pas tout à fait approprié. Une méthode propose aussi un processus, qui régit notamment l'enchaînement des activités de production d'une entreprise. Or UML n'a pas été pensé pour régir les activités de l'entreprise ; ce n'est pas DOD-STD-2167A ou CMM.

Les auteurs d'UML sont tout à fait conscients de l'importance du processus, mais ce sujet a été intentionnellement exclu des travaux de l'OMG. Comment prendre en compte toutes les organisations et cultures d'entreprises ? Un processus est adapté (donc très lié) au domaine d'activité de l'entreprise ; même s'il constitue un cadre général, il faut l'adapter au contexte de l'entreprise. Bref, améliorer un processus est une discipline à part entière, c'est un objectif qui dépasse très largement le cadre de l'OMA.

Approche fonctionnelle versus approche objet

Approche fonctionnelle : hiérarchie de fonctions pour factoriser certains comportements communs. Mais la maintenance est difficile et l'évolution rédhibitoire.

Séparation des données : une structure de données manipulées par des fonctions. Les pièges rencontrés conduisent à :

- centraliser dans les structures de données, les valeurs qui leurs sont propres ;
- centraliser les traitements associés à un type, auprès du type.

Les concepts fondateurs de l'approche objet :

- la notion d'objet et de classe (d'objets) ;
- l'encapsulation (les interfaces des objets) ;
- l'héritage (les hiérarchies d'objets) ;
- l'agrégation (la construction d'objets à l'aide d'objets).

Remarque : les langages orientés objet fournissent de nombreux autres mécanismes qui affinent ces concepts de base, favorisent la généricité du code ou améliorent sa robustesse.

Objet : entité autonome qui regroupe un ensemble de propriétés cohérentes et de traitements associés (et constitue le concept fondateur de l'approche du même nom)…

- un objet est une entité aux frontières précises qui possède une identité (un nom) ;
- un ensemble d'attributs caractérise l'état de l'objet ;
- un ensemble d'opérations (méthodes) en définissent le comportement ;
- un objet est une instance de classe (une occurrence d'un type abstrait).

Classe : type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés.

Encapsulation : concept de l'orienté objet qui…

- consiste à masquer les détails d'implémentation d'un objet, en définissant une interface. L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.
- facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface.
- garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'accesseurs).

Héritage (et polymorphisme) : concept de l'orienté objet qui…

- est un mécanisme de transmission des propriétés d'une classe (ses attributs et méthodes) vers une sous-classe. Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines. Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes. La spécialisation et la généralisation permettent de construire des hiérarchies de classes ;
- peut être simple ou multiple.
- évite la duplication et encourage la réutilisation.
- (le polymorphisme) représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes. Le polymorphisme augmente la généricité du code.

Agrégation : concept de l'orienté objet qui…

- est une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe ;
- permet donc de définir des objets composés d'autres objets ;
- permet d'assembler des objets de base, afin de construire des objets plus complexes.

L'approche objet : une solution parfaite ?

En résumé, l'approche objet c'est :
- un ensemble de concepts stables, éprouvés et normalisés ;
- une solution destinée à faciliter l'évolution d'applications complexes ;
- une panoplie d'outils et de langages performants pour le développement.

Oui, MAIS...

- malgré les apparences, il est plus naturel pour nos esprits cartésiens, de décomposer un problème informatique sous forme d'une hiérarchie de fonctions atomiques et de données, qu'en terme d'objets et d'interaction entre ces objets. De plus, le vocabulaire précis est un facteur d'échec important dans la mise en œuvre d'une approche objet.
- 'approche objet est moins intuitive que l'approche fonctionnelle !
- quels moyens utiliser pour faciliter l'analyse objet ?
- quels critères identifient une conception objet pertinente ?
- comment comparer deux solutions de découpe objet d'un système ?
- l'application des concepts objets nécessite une grande rigueur !
- le vocabulaire est précis (risques d'ambiguïtés, d'incompréhensions).
- comment décrire la structure objet d'un système de manière pertinente ?

Quels sont les remèdes aux inconvénients de l'approche objet ?

Un langage pour exprimer les concepts objet qu'on utilise, afin de pouvoir :
- représenter des concepts abstraits (graphiquement par exemple).
- limiter les ambiguïtés (parler un langage commun).
- faciliter l'analyse (simplifier la comparaison et l'évaluation de solutions).

Une démarche d'analyse et de conception objet, pour :
- ne pas effectuer une analyse fonctionnelle et se contenter d'une implémentation objet, mais penser objet dès le départ.
- définir les vues qui permettent de couvrir tous les aspects d'un système, avec des concepts objets.

Bref : il nous faut un outil qui apporte une dimension méthodologique à l'approche objet, afin de mieux maîtriser sa richesse et sa complexité.

Les points faibles d'UML :

- la mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation. Même si l'Espéranto est une utopie, la nécessité de s'accorder sur des modes d'expression communs est vitale en informatique. UML n 'est pas à l'origine des concepts objets, mais en constitue une étape majeure, car il unifie les différentes approches et en donne une définition plus formelle.
- le processus (non couvert par UML) est une autre clé de la réussite d'un projet. Or, l'intégration d'UML dans un processus n'est pas triviale et améliorer un processus est un tâche complexe et longue.

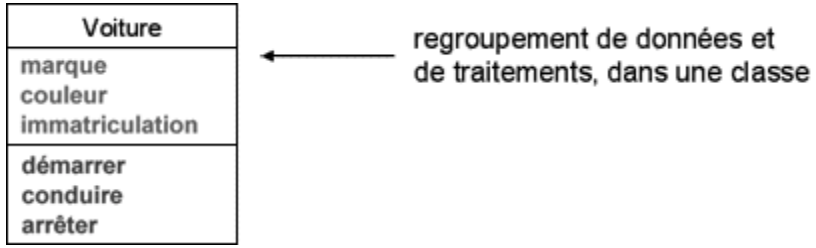
Les auteurs d'UML sont tout à fait conscients de l'importance du processus, mais l'acceptabilité industrielle de la modélisation objet passe d'abord par la disponibilité d'un langage d'analyse objet performant et standard.

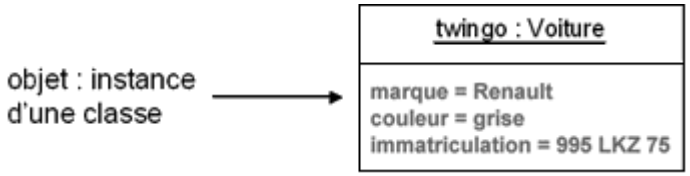
3 Comparaison des concepts UML et HBDS

Vues statiques d'UML : diagramme de classes

Classe UML

Une classe est un type abstrait caractérisé par des propriétés (attributs et méthodes) communes à un ensemble d'objets et permettant de créer des objets ayant ces propriétés. Classe = attributs + méthodes + instanciation

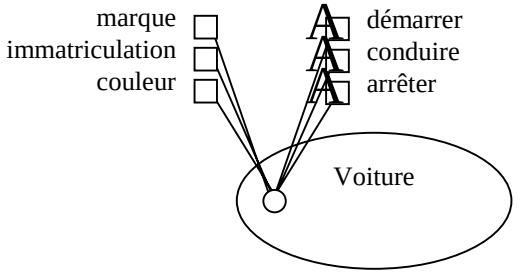


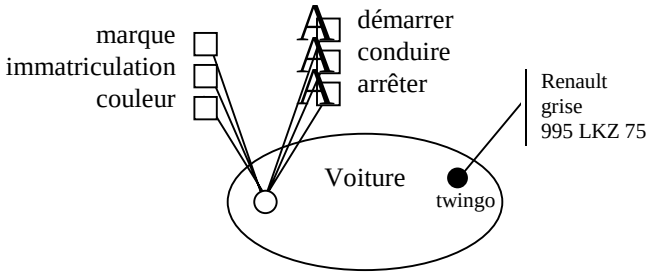


Hypergraph Based Data Structure : données

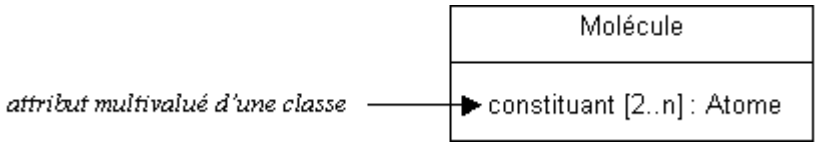
Classe HBDS

Une classe est un ensemble d'éléments appelés objets. Les objets de cette classe sont caractérisés par des attributs (qui prennent des valeurs ou sont des algorithmes signalés par un 'A').

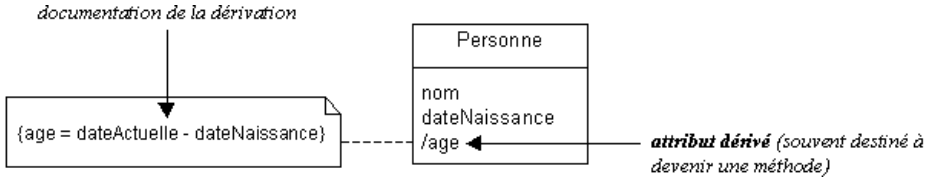




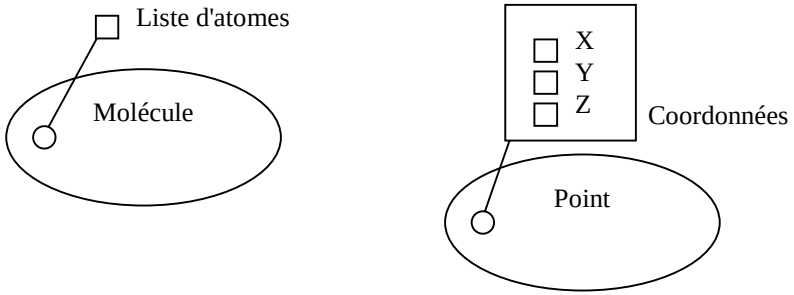
Attribut multivalué d'une classe UML



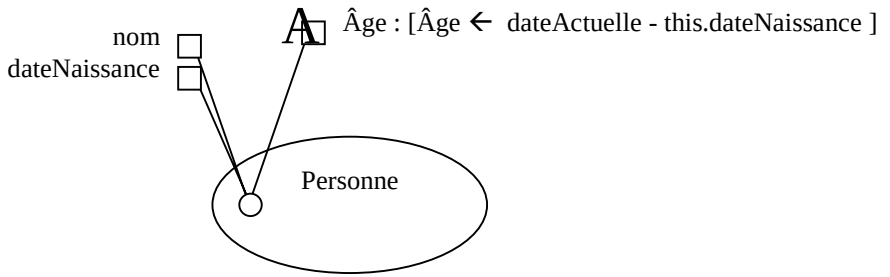
Attribut dérivé d'une classe UML



Attribut multivalué ou composé d'une classe HBDS

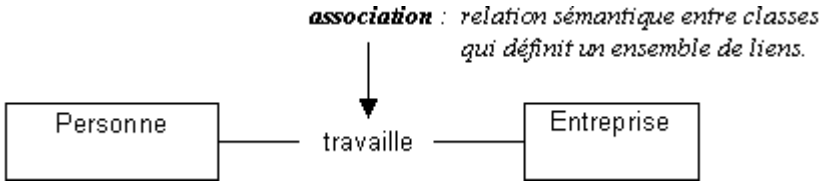


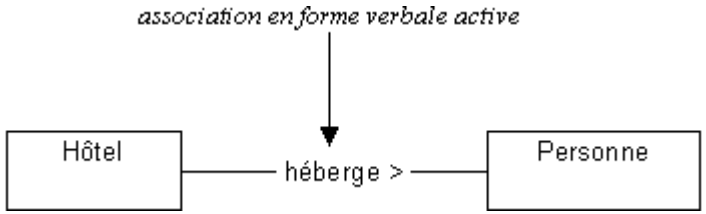
Attribut calculé d'une classe HBDS

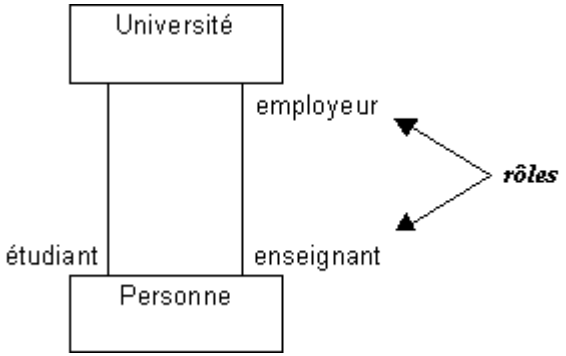


Associations entre classes UML

Une association exprime une connexion sémantique bidirectionnelle entre deux classes.



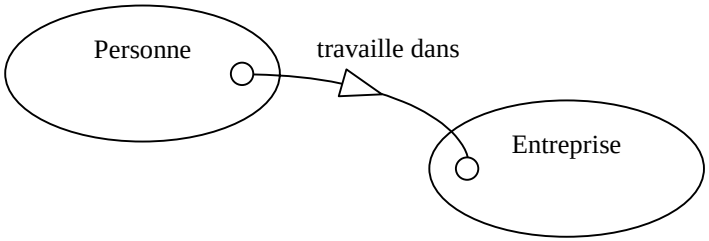


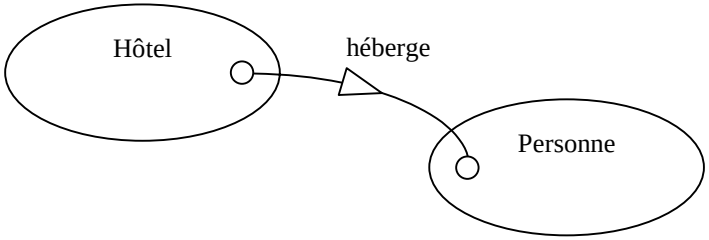


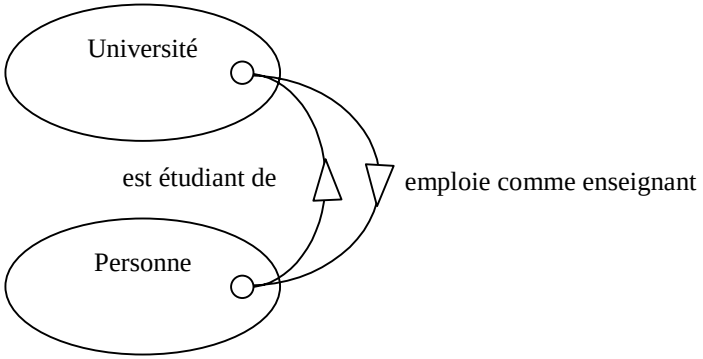
Rôles : spécifie la fonction d'une classe pour une association donnée (indispensable pour les associations réflexives).

Liens entre classes HBDS

Un lien entre classes exprime une relation potentielle entre un objet de la classe de départ et un objet de la classe. Le lien inverse est implicite.

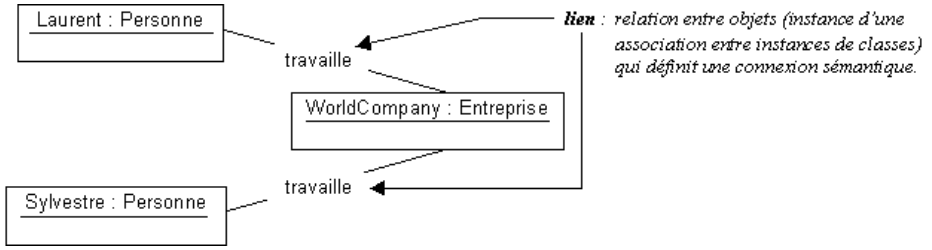






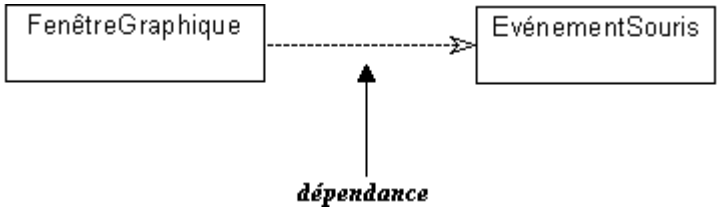
Instanciation d'une association UML

L'association est instanciable dans un diagramme d'objets ou de collaboration, sous forme de liens entre objets issus de classes associées.



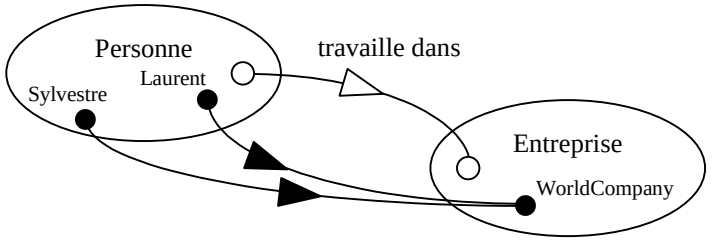
Relation de dépendance UML

Une relation de dépendance est une relation d'utilisation unidirectionnelle et d'obsolescence (une modification de l'élément dont on dépend, peut nécessiter une mise à jour de l'élément dépendant).

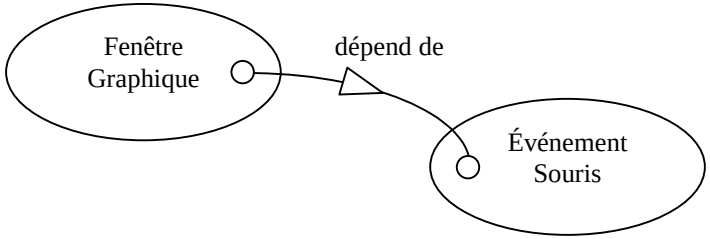


Lien entre objets HBDS

Les objets d'une classe peuvent être en relation avec des objets de la même classe ou d'autres classes grâce à la notion de lien "entre objets".

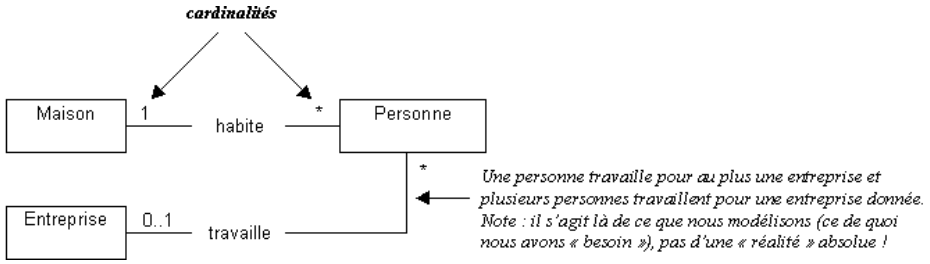


Lien HBDS pour traduire une "relation de dépendance"



Cardinalités UML

La cardinalité précise le nombre d'instances qui participent à une relation.

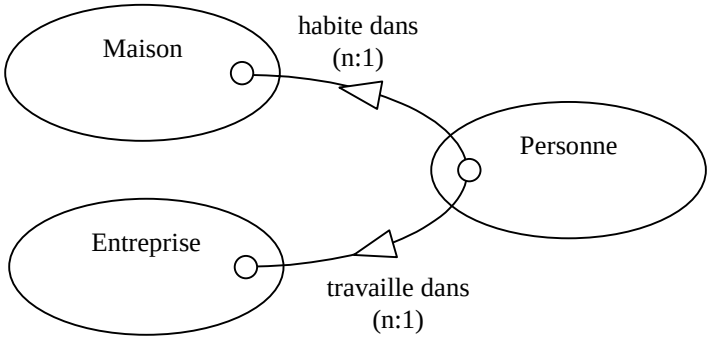


Expression des cardinalités d'une relation UML

- n : exactement "n" (n, entier naturel > 0)
  exemples : "1", "7"
- n..m : de "n" à "m" (entiers naturels ou variables, m > n)
  exemples : "0..1", "3..n", "1..31"
- * : plusieurs (équivalent à "0..n" et "0..*")
- n..* : "n" ou plus (n, entier naturel ou variable)
  exemples : "0..*", "5..*"

Cardinalités HBDS

La cardinalité indique une restriction sous la forme d'une borne supérieure pour le demi-degré positif ou négatif de chaque objet qui est à l'extrémité d'un lien. Elle s'exprime sous la forme : (borne_supérieure_du_demi-degré-négatif : borne_supérieure_du_demi-degré-positif)



Pour rendre plus lisibles les cardinalités, il est préférable de dessiner les liens de gauche à droite et de bas en haut…

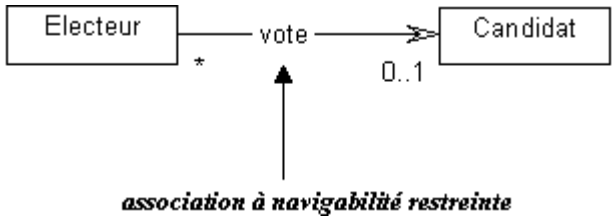
Expression des cardinalités d'un lien HBDS

- (n:m) : aucune restriction sur le nombre de successeurs et de prédécesseurs.
- (1:n) : un objet peut avoir plusieurs successeurs mais un seul prédécesseur.
- (n:1) : un objet peut avoir plusieurs prédécesseurs mais un seul successeur.
- (1:1) : un objet ne peut avoir qu'un seul prédécesseur ou qu'un seul successeur.

On peut éventuellement remplacer n (ou m) signifiant "plusieurs" par un entier indiquant précisément le nombre maximum…

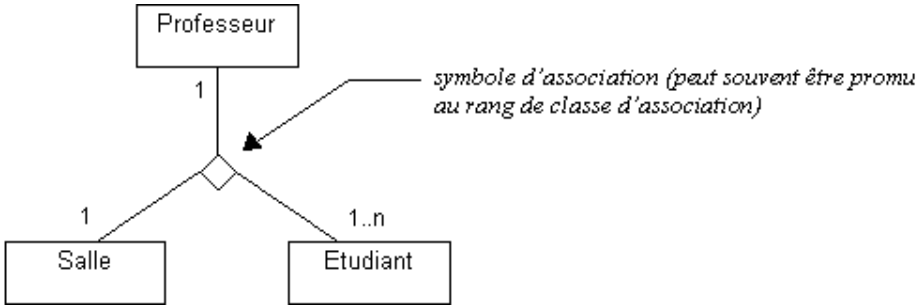
Association UML à navigabilité restreinte

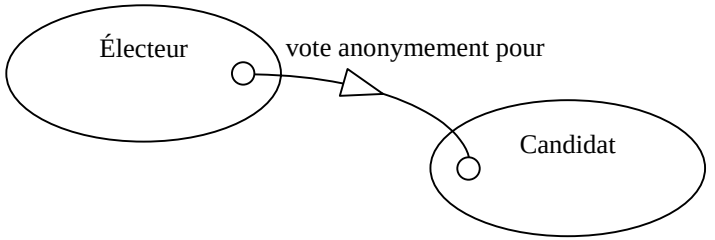
Par défaut, une association est navigable dans les deux sens. La réduction de la portée de l'association est souvent réalisée en phase d'implémentation, mais peut aussi être exprimée dans un modèle pour indiquer que les instances d'une classe ne "connaissent" pas les instances d'une autre.



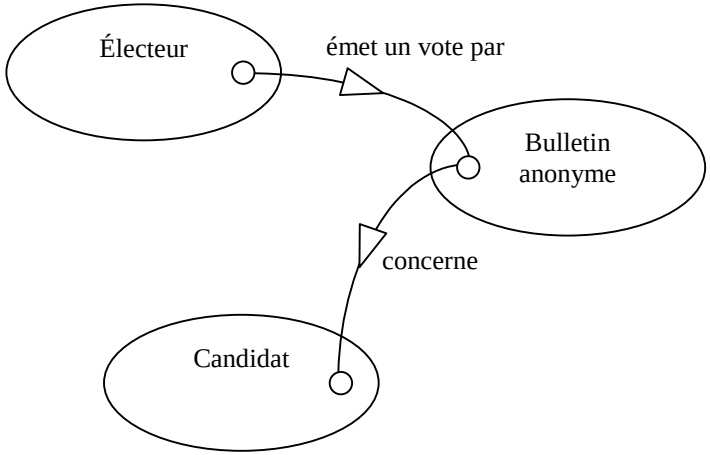
Association UML n-aire

Une association n-aire est une association qui relie plus de deux classes... De telles associations sont difficiles à déchiffrer et peuvent induire en erreur. Il vaut mieux limiter leur utilisation, en définissant de nouvelles catégories d'associations.

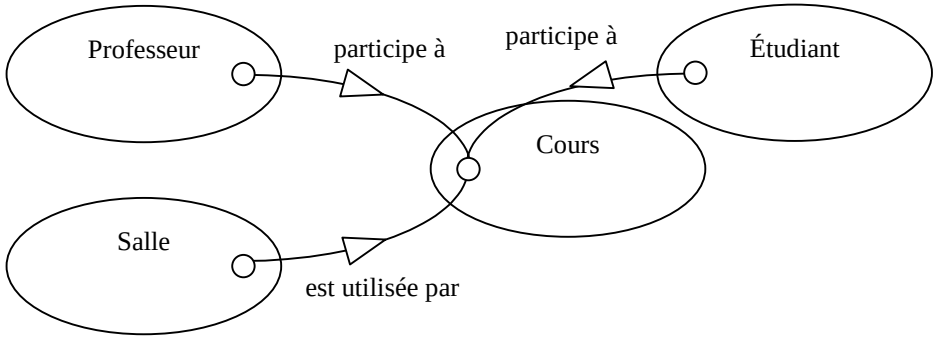




ou

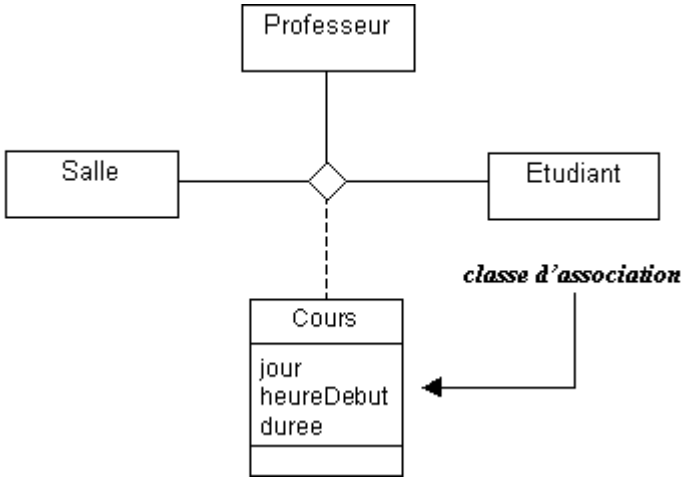


Classe HBDS pour traduire une "association n-aire"

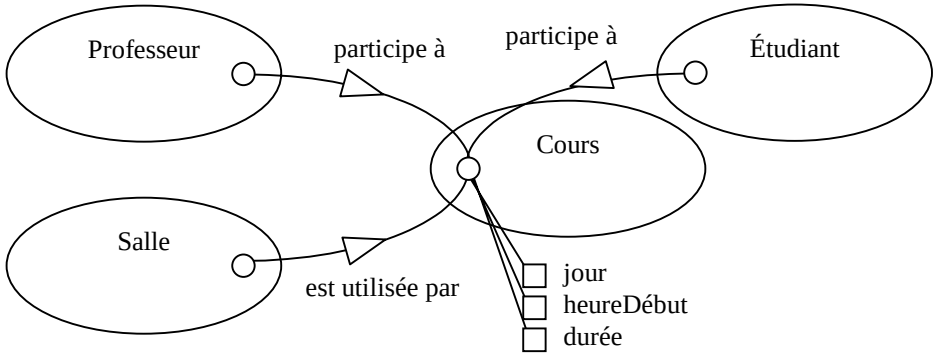


Classe d'association UML

Une classe d'association est une classe qui réalise la navigation entre les instances d'autres classes.

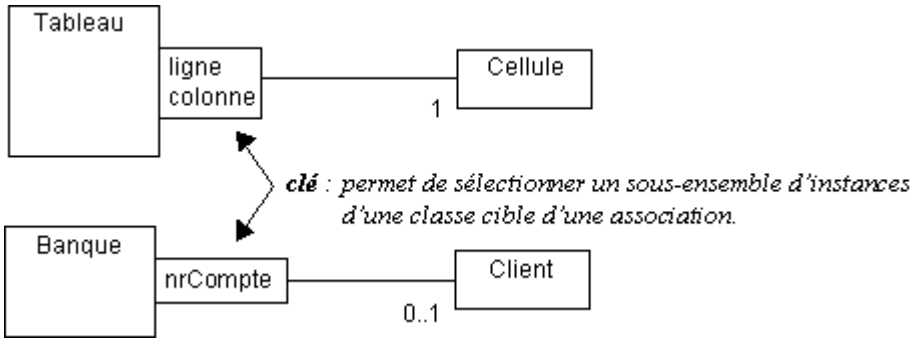


Classe HBDS pour traduire une "classe d'association"

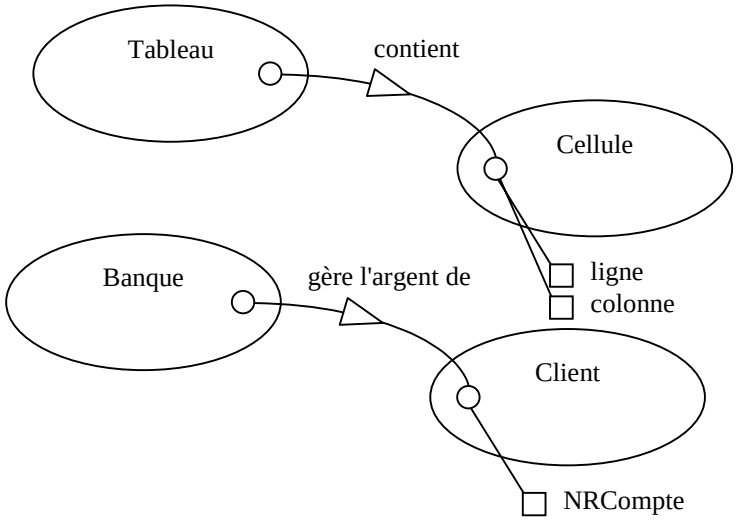


Qualification UML

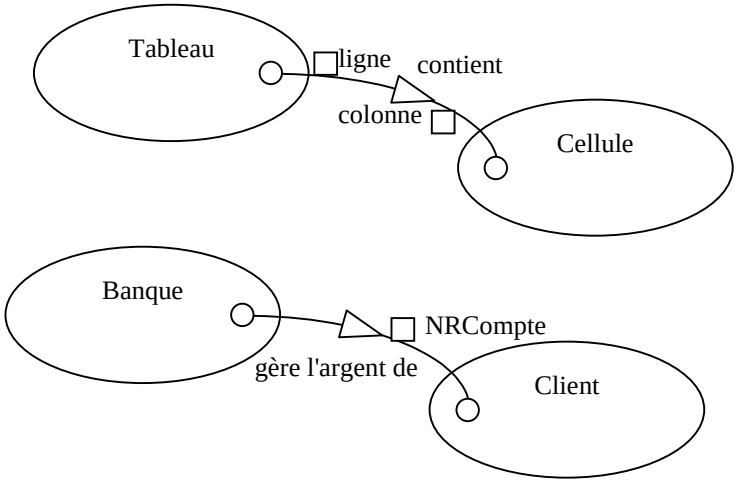
Une qualification permet de sélectionner un sous-ensemble d'objets, parmi l'ensemble des objets qui participent à une association. La restriction de l'association est définie par une clé, qui permet de sélectionner les objets ciblés.



Classes reliées pour traduire une "qualification UML"



Liens multivalués



Héritage UML

Les hiérarchies de classes permettent de gérer la complexité, en ordonnant les objets au sein d'arborescences de classes, d'abstraction croissante.

Spécialisation

Démarche descendante, qui consiste à capturer les particularités d'un ensemble d'objets, non discriminés par les classes déjà identifiées. Consiste à étendre les propriétés d'une classe, sous forme de sous-classes, plus spécifiques (permet l'extension du modèle par réutilisation).

Généralisation

Démarche ascendante, qui consiste à capturer les particularités communes d'un ensemble d'objets, issus de classes différentes. Consiste à factoriser les propriétés d'un ensemble de classes, sous forme d'une super-classe, plus abstraite (permet de gagner en généricité).

Classification

L'héritage (spécialisation et généralisation) permet la classification des objets. Une bonne classification est stable et extensible : ne classifiez pas les objets selon des critères instables (selon ce qui caractérise leur état) ou trop vagues (car cela génère trop de sous-classes).

Les critères de classification sont subjectifs. Le principe de substitution (Liksow, 1987) permet de déterminer si une relation d'héritage est bien employée pour la classification : "Il doit être possible de substituer n'importe quel instance d'une super-classe, par n'importe quel instance d'une de ses sous-classes, sans que la sémantique d'un programme écrit dans les termes de la super-classe n'en soit affectée." Si Y hérite de X, cela signifie que "Y est une sorte de X" (analogies entre classification et théorie des ensembles).

Extension des Types Abstraits de Données HBDS

François Bouillé est parti des concepts mathématiques d'Ensemble, d'Éléments, de Propriété et de Relation, pour arriver à 6 types abstraits de données (TAD) de base : Classe, Attribut de classe, Lien entre classes, Objet, Attribut d'Objet, Lien entre objets. Ces concepts ont été introduits en 1975 par Liksow et Zilles au MIT, mais les travaux de Dahl en 1961 utilisaient déjà les concepts d'objets.

Extension des TAD

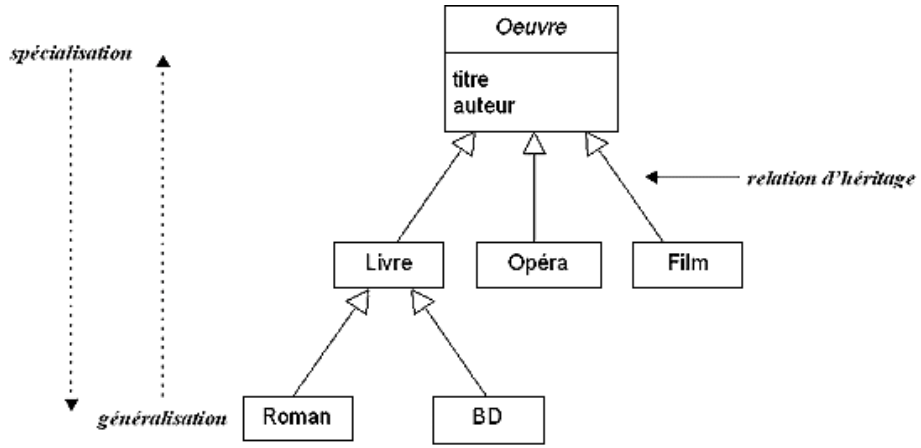
- Hyperclasse : regroupement de classes (il peut y avoir une intersection entre des hyperclasses mais pas entre classes).
- Hyperattribut : attribut d'une hyperclasse
- Hyperlien : lien entre hyperclasse (remarque toutes les classes d'une hyperclasse sont liées par les liens de l'hyperclasse).

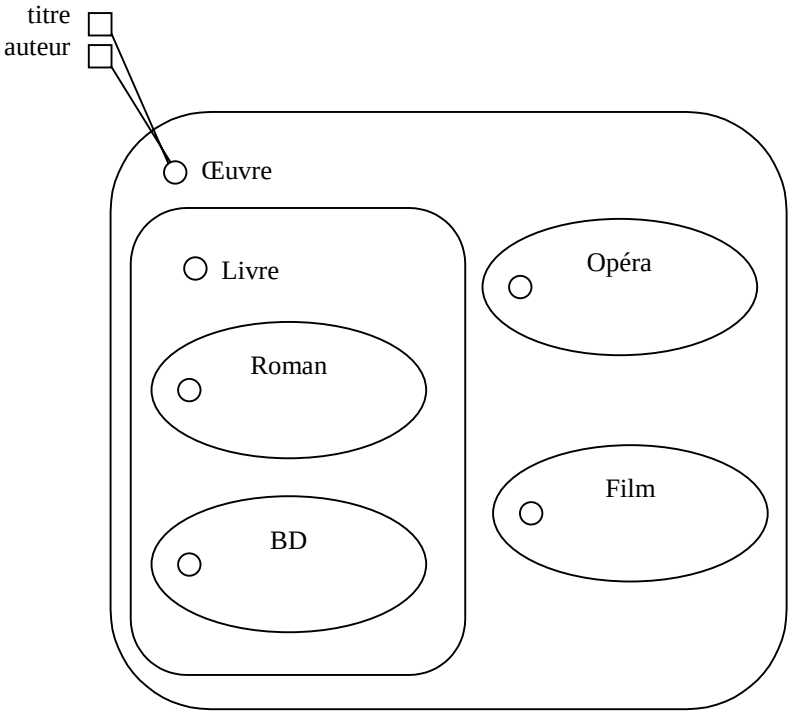
Héritage simple

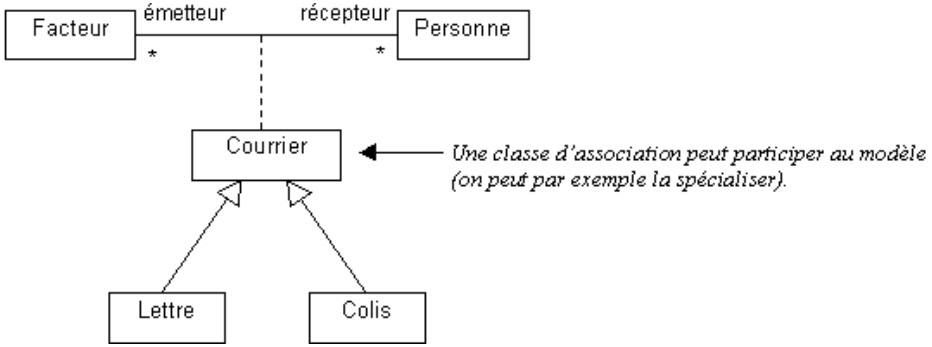
L'héritage simple consiste pour une classe à hériter tous ses attributs d'une hyperclasse.

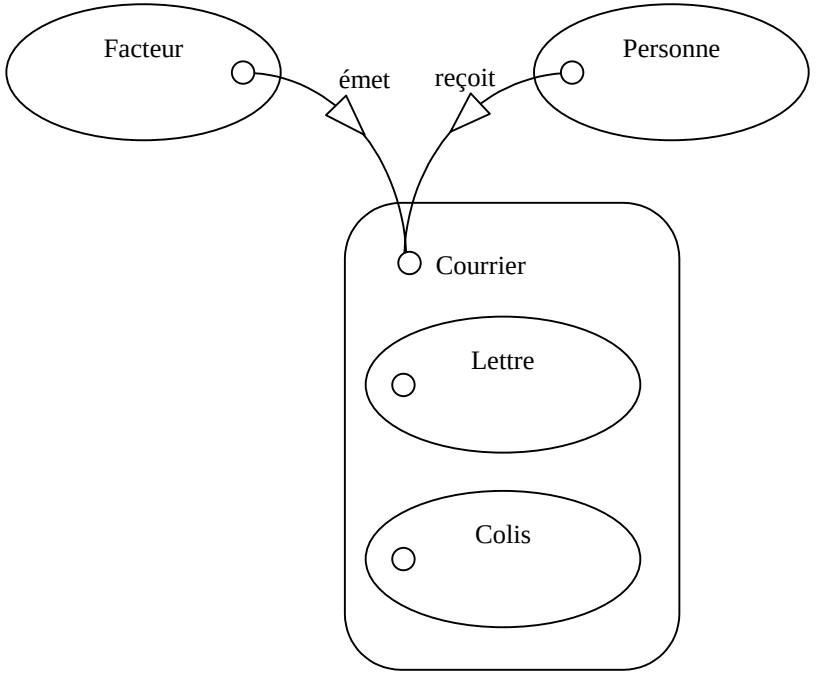
Héritage multiple

L'héritage multiple concerne les classes situées à l'intersection de deux hyperclasses.


spécialisation
généralisation
Oeuvre
titre
auteur
relation d'héritage
Livre
Opéra
Film
Roman
BD


titre
auteur
Œuvre
Livre
Roman
BD
Opéra
Film


Facteur
émetteur
récepteur
Personne
*
*
Courrier
Une classe d'association peut participer au modèle
(on peut par exemple la spécialiser).
Lettre
Colis


Facteur
émet
reçoit
Personne
Courrier
Lettre
Colis

Agrégation et composition UML

L'agrégation et la composition sont des vues subjectives. Lorsqu'on représente (avec UML) qu'une molécule est "composée" d'atomes, on sous-entend que la destruction d'une instance de la classe "Molécule", implique la destruction de ses composants, instances de la classe "Atome" (cf. propriétés de la composition). Bien qu'elle ne reflète pas la réalité ("rien ne se perd, rien ne se crée, tout se transforme"), cette abstraction de la réalité nous satisfait si l'objet principal de notre modélisation est la molécule...

Agrégation UML

L'agrégation est une association non symétrique, qui exprime un couplage fort et une relation de subordination. Elle représente une relation de type "ensemble / élément".

UML ne définit pas ce qu'est une relation de type "ensemble / élément", mais il permet cependant d'exprimer cette vue subjective de manière explicite.

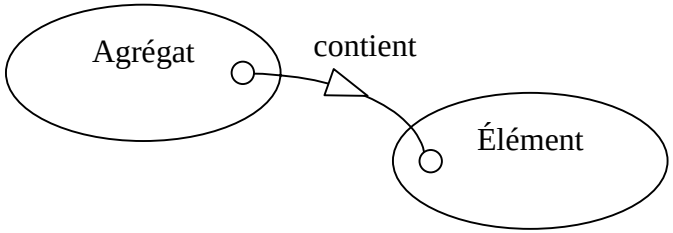
Une agrégation peut notamment (mais pas nécessairement) exprimer : qu'une classe (un "élément") fait partie d'une autre ("l'agrégat"), qu'un changement d'état d'une classe, entraîne un changement d'état d'une autre, qu'une action sur une classe, entraîne une action sur une autre.

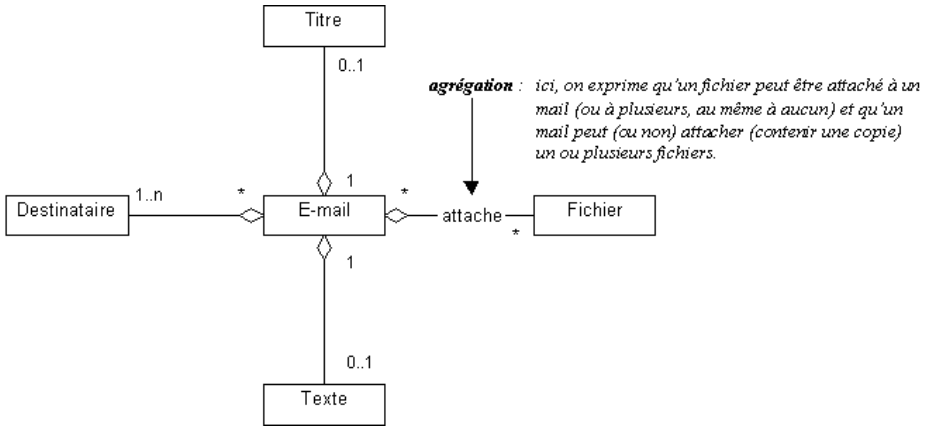
A un même moment, une instance d'élément agrégé peut être liée à plusieurs instances d'autres classes (l'élément agrégé peut être partagé). Une instance d'élément agrégé peut exister sans agrégat (et inversement) : les cycles de vies de l'agrégat et de ses éléments agrégés peuvent être indépendants.

Liens HBDS pour traduire l'agrégation et la composition d'UML

Lien HBDS "contient" pour traduire l'agrégation UML

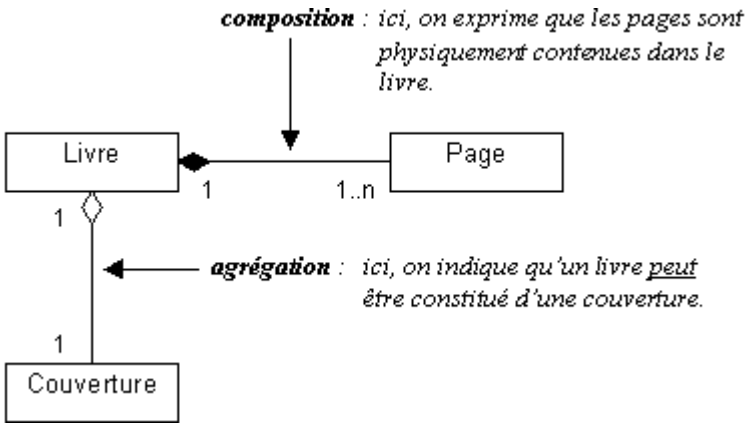
L'utilisation d'un lien "contient" suffit amplement à décrire le phénomène d'agrégation…

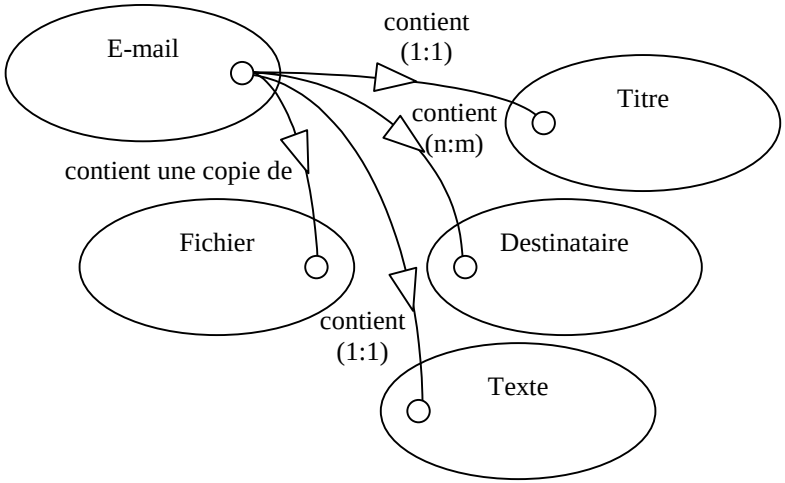




Composition UML

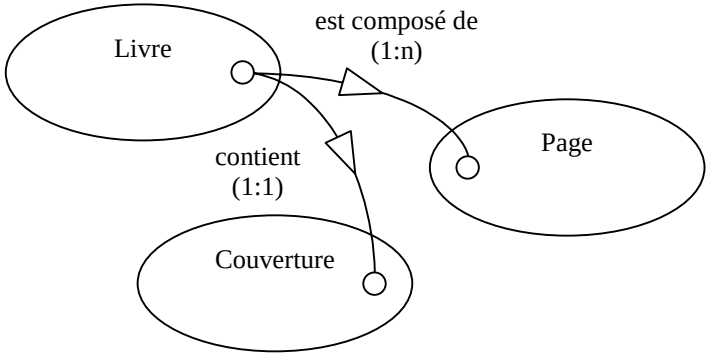
La composition est une agrégation forte (agrégation par valeur). Les cycles de vies des éléments (les "composants") et de l'agrégat sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi. A un même moment, une instance de composant ne peut être liée qu'à un seul agrégat. Les "objets composites" sont des instances de classes composées.





Lien "est composé de" pour traduire la composition UML

L'utilisation d'un lien "est composé de" suffit amplement à décrire le phénomène de composition…



Paquetages (packages) UML

Les paquetages sont des éléments d'organisation des modèles.

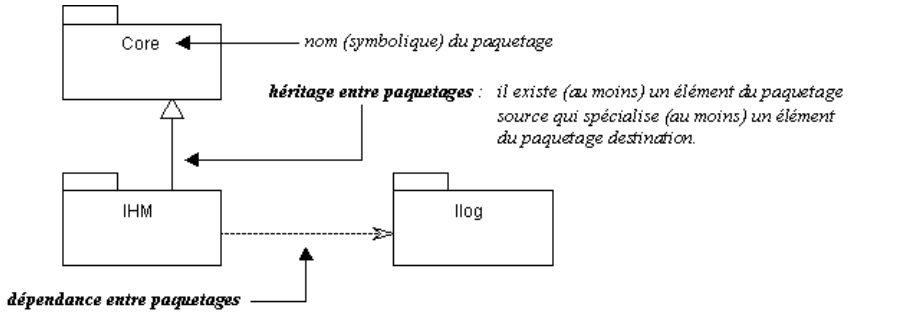
Ils regroupent des éléments de modélisation, selon des critères purement logiques.

Ils permettent d'encapsuler des éléments de modélisation (ils possèdent une interface).

Ils permettent de structurer un système en catégories (vue logique) et sous-systèmes (vue des composants).

Ils servent de "briques" de base dans la construction d'une architecture. Ils représentent le bon niveau de granularité pour la réutilisation. Les paquetages sont aussi des espaces de noms.

Paquetages : relations entre paquetages

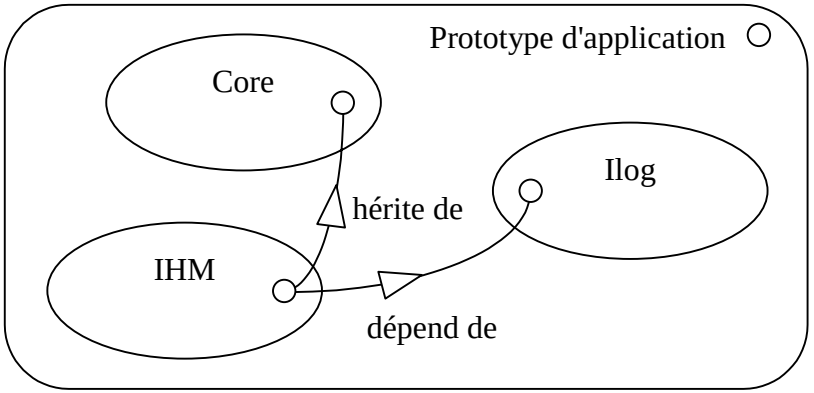


Extension des TAD et prototypes HBDS

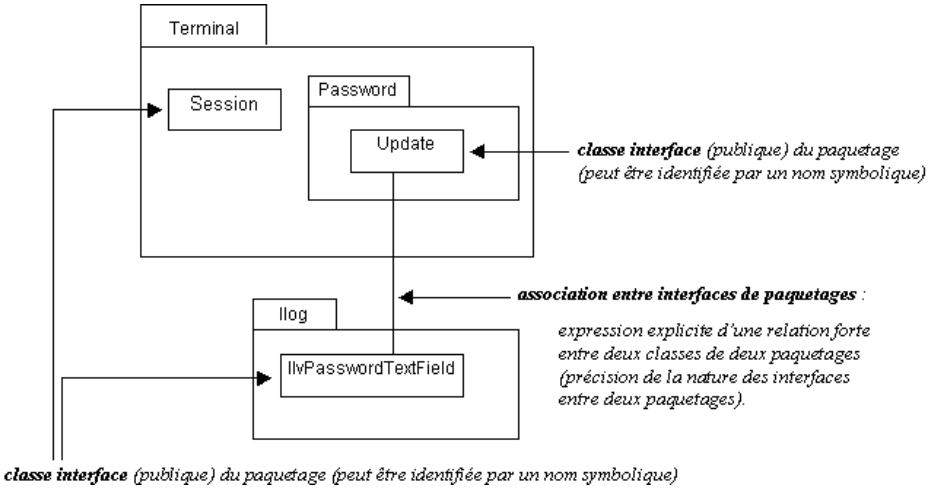
Un prototype HBDS est une structure HBDS représentant un modèle de données qui décrit les Types Abstraits de données qui peuvent être instanciés. Les TAD constituant un prototype sont appelés embryons.

Les règles d'instanciation sont les suivantes :

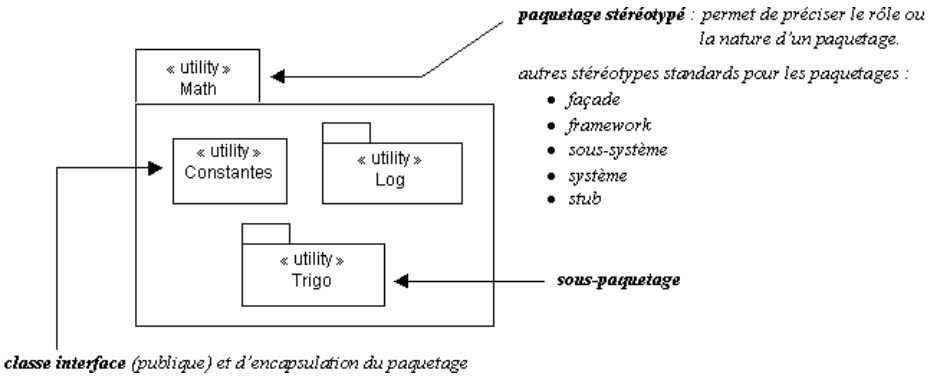
- toute classe du prototype génère une hyperclasse dans une structure ;
- tout attribut du prototype génère un hyperattribut dans une structure ;
- les liens du prototype restent des liens (pas de génération d'hyperlien).

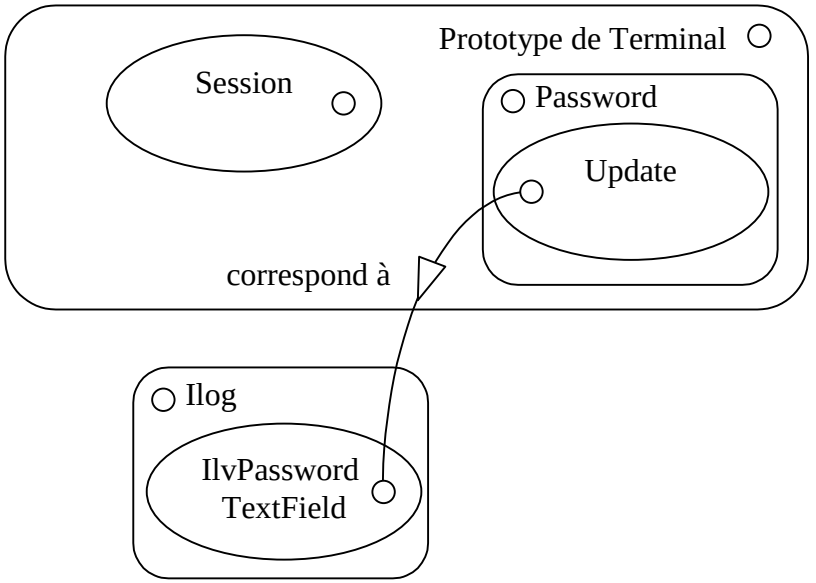


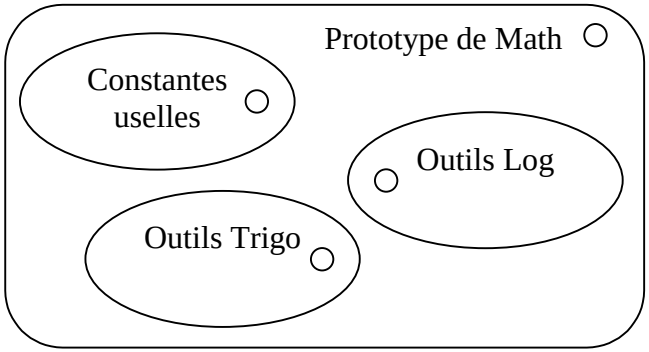
Paquetages : interfaces



Paquetages : stéréotypes





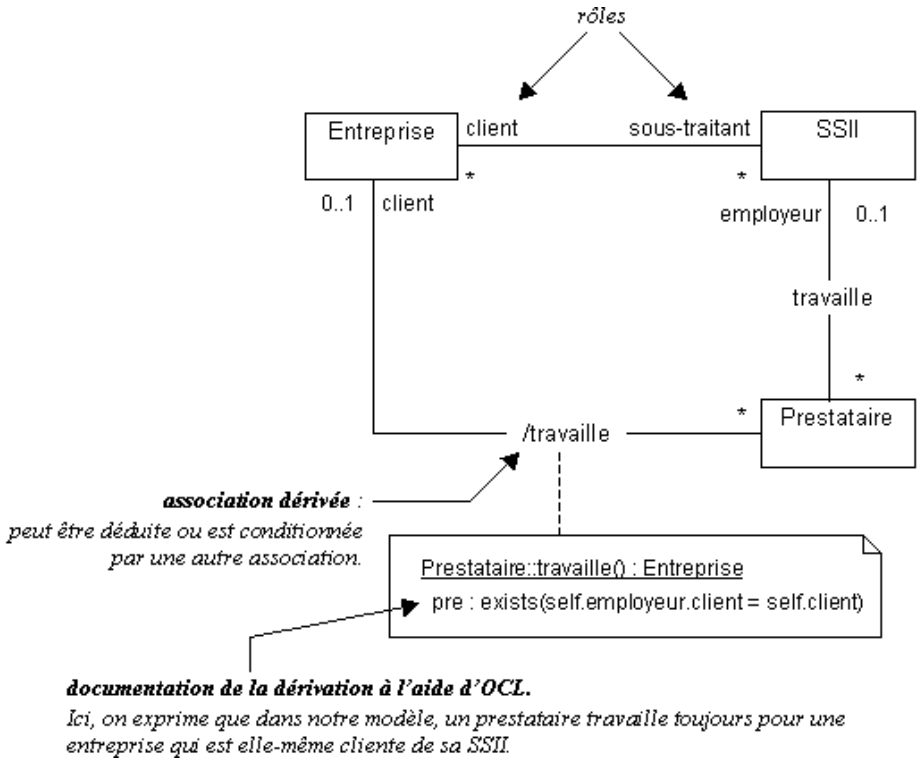


Association dérivée UML

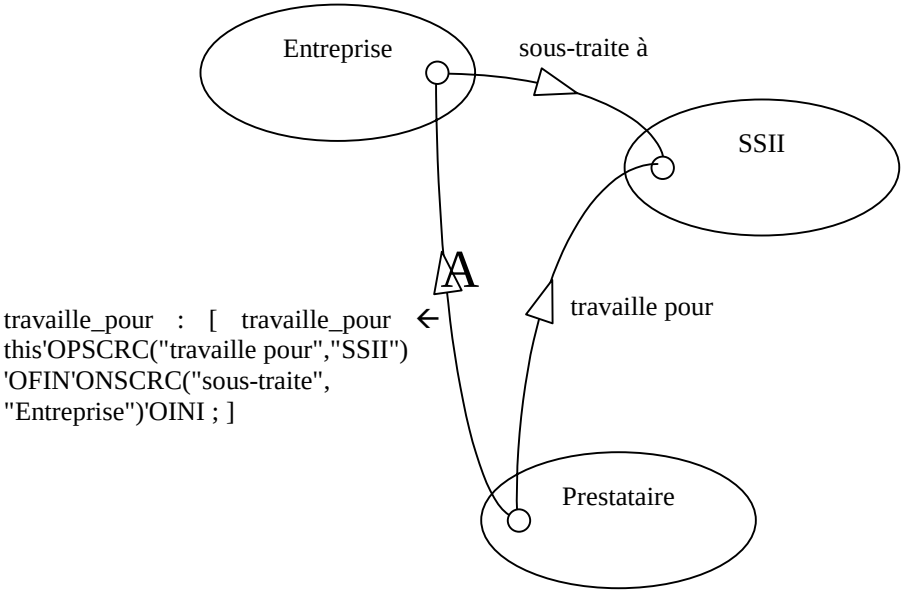
Les associations dérivées sont des associations redondantes, qu'on peut déduire d'une autre association ou d'un ensemble d'autres associations.

Elles permettent d'indiquer des chemins de navigation "calculés", sur un diagramme de classes.

Elles servent beaucoup à la compréhension de la navigation (comment joindre telles instances d'une classe à partir d'une autre).



Lien calculé et foncteur HBDS

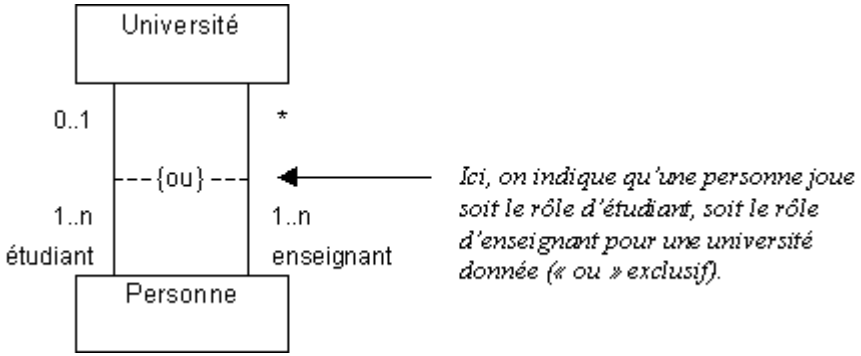


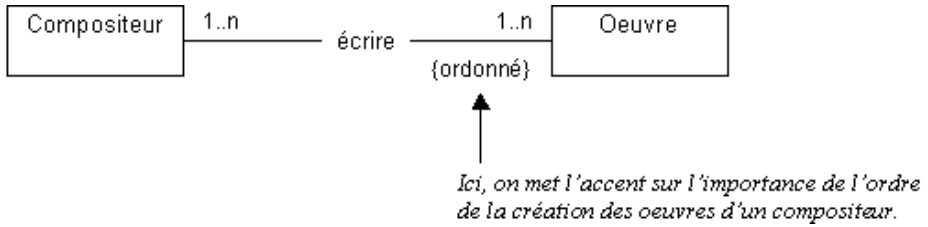
Contrainte sur une association UML

Les contraintes sont des expressions qui précisent le rôle ou la portée d'un élément de modélisation (elles permettent d'étendre ou préciser sa sémantique).

Sur une association, elles peuvent par exemple restreindre le nombre d'instances visées (ce sont alors des "expressions de navigation").

Les contraintes peuvent s'exprimer en langage naturel. Graphiquement, il s'agit d'un texte encadré d'accolades.





Traduction HBDS d'une contrainte sur une association UML