heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Ecole d'Ingénieurs de l'état de Vaud
ERIC LEFRANÇOIS
25 Avril 2018

Génie Logiciel

Chap 6 – Modélisation de domaine

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Sommaire

4 Modélisation de domaine

La modélisation du domaine est une activité essentielle de l'analyse des besoins. Du point de vue de la méthode UP, cette activité fait partie de la discipline «Modélisation métier».

Si les cas d'utilisation constituent un artefact très important de l'analyse des besoins, ces derniers ne sont pas orientés «objet». Ils sont plutôt orientés «processus» puisqu'ils décrivent essentiellement les fonctionnalités du système.

Le **modèle du domaine**, - qui représente les **classes conceptuelles** les plus importantes du système à développer -, constitue au contraire un aterfact **orienté objet** et - à ce titre - sera directement utilisé lors de la conception des classes logicielles.

La modélisation du monde réel

Définition 1: Modèle du Domaine [Martin Fowler - 1996]

Le modèle du domaine est une représentation des classes conceptuelles ou des objets du *monde réel* dans un *domaine* donné.

Le **monde réel** désigne le système existant pour lequel il a été décidé de concevoir un logiciel dans le but d'une automatisation de processus, d'une amélioration de la qualité, etc.

Par exemple, pour réaliser un logiciel de gestion des horaires dans une école comme la notre (HEIG-VD), on définira un modèle de domaine s'appliquant :

- **à un monde réel**: l'école et son fonctionnement avec ses étudiants, son corps enseignant et son personnel administratif;
- **à un domaine**: celui de la gestion des horaires.

Qu'en est-il des mondes irréels ?

Les développeurs sont souvent amenés à modéliser des systèmes informatiques dans des domaines éloignés de mondes réels *concrets* et familiers comme peuvent l'être l'HEIG-VD, une banque, une administration, etc.

C'est par exemple le cas des logiciels de télécommunications qui manipulent des concepts purement logiques ou abstraits tels que «Connexion», «Port», «Dialogue de routage», «Protocole», etc. La définition d'un modèle de domaine est bien entendue tout à fait indiquée, la seule différence étant que les concepts mis en jeu seront plus abstraits et moins familiers, on ne pourra pas toujours les «toucher du doigt».

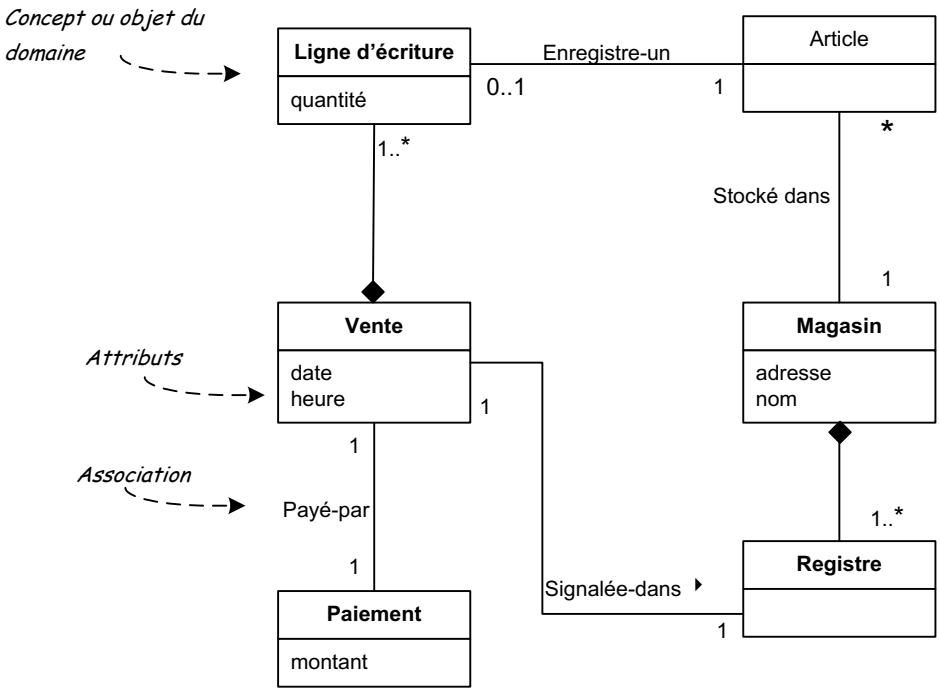
Le Modèle du domaine et la méthode UP

En général, la définition du modèle de domaine fait partie intégrante de la *phase d'élaboration*. Il sera à la fois commencé et terminé pendant cette phase.

Le Modèle du Domaine et UML

La représentation visuelle d'un modèle de domaine s'appuie sur la syntaxe UML du *diagramme de classes*, comme dans l'exemple qui suit, qui représente la modélisation de la gestion des ventes dans un magasin du monde réel.

Figure 1. Représentation d'un modèle de domaine en UML



☺ ***Note*** *: On supposera dans ce document que le lecteur connaît «tout» de la syntaxe des diagrammes de classe UML. Sinon, se reporter à l'annexe correspondante.*

4.1 CARACTERISTIQUES ESSENTIELLES DE LA MODELISATION DE DOMAINE

Pour celles et ceux qui seraient habitués à utiliser la syntaxe du diagramme de classes dans le cadre d'une conception ou d'une implémentation de logiciel, ATTENTION!

La modélisation de domaine s'inscrit dans le cadre de l'analyse d'un problème et de sa compréhension. Les entités que nous retrouverons dans ces diagrammes sont donc un **reflet du problème** et non pas un reflet du logiciel qui sera mis en œuvre pour informatiser le problème en question.

☺ *Rassurons-nous toutefois...*

Les diagrammes de classes qui seront élaborés en phase de conception seront issus et inspirés directement des diagrammes de modélisation de domaine. Ces derniers sont donc d'une importance capitale, tant pour la compréhension du problème que pour la conception du futur logiciel. Ils constituent en quelque sorte un pont entre l'analyse et la conception.

Voici donc deux particularités, - que nous considérons comme essentielles -, à associer aux diagrammes de modélisation de domaine.

Le modèle du domaine est une abstraction

Du point de vue des attributs, on ignore les détails sans intérêts (la vue est partielle). Dans l'optique de la méthode UP, les détails seront souvent repris et analysés dans le cadre des itérations de la phase de construction.

Le modèle du domaine est une représentation visuelle du monde réel

Les classes représentées modélisent les **concepts du monde réel** : il s'agit de classes conceptuelles et non pas des classes logicielles que l'on rencontrera dans la réalisation du programme en Java ou en C#.

Définition 2: Notion de classe conceptuelle

Une *classe conceptuelle* appartient au monde réel du domaine qui nous intéresse. Elle en illustre le vocabulaire. Elle peut représenter :

- une chose ou un objet tangible (un document, un appareillage, etc.),
- une personne pouvant être impliquée dans le fonctionnement du système (un acteur principal, auxiliaire, etc.),
- une idée,
- etc.

✍ **Ce qui a pour conséquence...**

- Que seuls les attributs y sont représentés. La notion de méthode n'y a pas de sens.
- Que les associations sont par nature bidirectionnelles. La notion de «navigabilité» des associations n'y a pas de sens.

4.2 CONSEILS

Des particularités énoncées précédemment découlent un certain nombre de conseils. Les garder à l'esprit.

☺ Sur-spécifier au niveau des classes

Ne pas trop détailler du point de vue des attributs, mais ne pas hésiter - *au contraire* - à sur-spécifier avec de nombreuses classes. Chercher à réduire le nombre de classes conceptuelles n'a pas de sens.

☺ Ne pas se polariser sur la découverte des attributs

Comme l'accent est placé sur l'identification des concepts et sur l'identification des relations entre ces différents concepts, on se retrouvera ainsi avec de nombreuses classes conceptuelles sans attribut, ce qui n'aura rien de choquant en soi. L'identification complète des attributs se fera plus tard, de manière itérative. Au départ, considérer les attributs comme du détail...

Ainsi, on ne va pas éliminer une classe conceptuelle en s'appuyant sur le fait qu'il n'y aura aucune information à mémoriser la concernant. Un critère qui - par contre - est important en phase de conception ou d'implémentation.

☺ Ne rien rajouter à la réalité

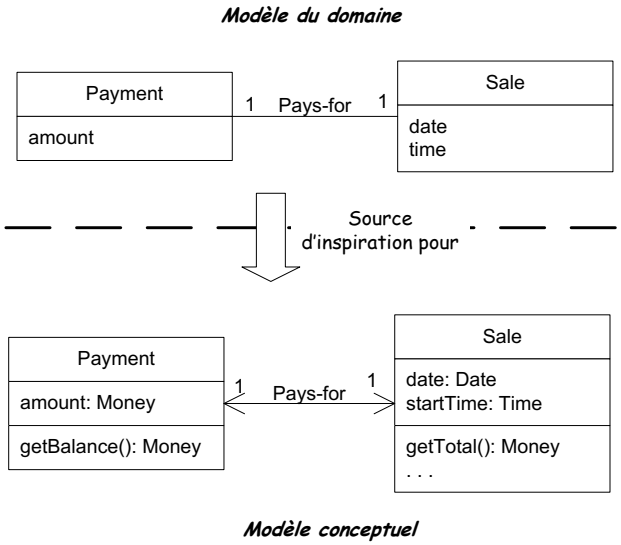
On doit se focaliser sur la modélisation du monde réel et se placer dans l'esprit de celui ou de ceux qui interviennent et agissent dans ce cadre. Rajouter des concepts de son propre cru serait très dangereux.

☺ Le choix des noms

Les noms doivent refléter le vocabulaire du domaine, au plus près. Cela facilite la compréhension du modèle.

Par la suite, comme le modèle de domaine inspirera directement le modèle conceptuel (classes logicielles), les noms qui seront alors donnés aux différentes classes, aux méthodes, etc. continueront logiquement de refléter le vocabulaire du domaine. C'est ainsi que l'on réduira le décalage entre les concepts manipulés dans le produit final et le monde réel. **Un très bon point pour faciliter la maintenance ultérieure du logiciel**.

Figure 2. Du modèle de domaine au modèle conceptuel



☺ Quelles associations retenir?

- Ne retenir que les associations pour lesquelles il est nécessaire de mémoriser la relation ; même si ces dernières n'ont lieu que de manière transitoire (comme par exemple qu'une vente «Est-Sollicitée-Par-Un-Client».
- Eviter les associations redondantes (notamment celles qui dérivent d'autres associations)

☺ Type de données : attribut ou classe conceptuelle?

Les attributs représentent des valeurs qui caractérisent une classe conceptuelle. A ce titre, ils correspondent le plus généralement à :

- des **types primitifs** au sens Java : nombres, booléens et caractères
- des **textes**,
- des **types énumérés,**
- des **types « record »**.

Types énumérés

Comme p.ex. le type «Semaine», ou le type «Couleur». Les valeurs de ces types énumérés seront spécifiées directement par l'analyste, ou plus tard par le concepteur.

Types «record»

Le type des attributs peut correspondre à des types structurés correspondant à la notion de «record» des langages de type Pascal ou Ada (p.e le type «`Adressse`», le type «`PointGraphique`»), à savoir un simple regroupement de valeurs.

En phase d'implémentation, les types « record » sont mis en œuvre la plupart du temps sous la forme d'une classe logicielle (cela dépend du langage choisi).

Dans le cadre de la modélisation de domaine, qu'en est-il ? Faut-il les faire apparaître sous la forme d'une *classe conceptuelle* ou se contenter d'indiquer leur nom (p.e. : `Adresse`) comme type d'attribut.

Si le type record fait l'objet d'associations spécifiques, d'une généralisation, d'une spécialisation, la réponse est claire: le faire apparaître sous la forme d'une classe conceptuelle.

Dans le cas contraire, cela dépend...

- de ce que l'on désire communiquer,
- «faire passer» au travers du modèle,
- du degré de détails, etc.

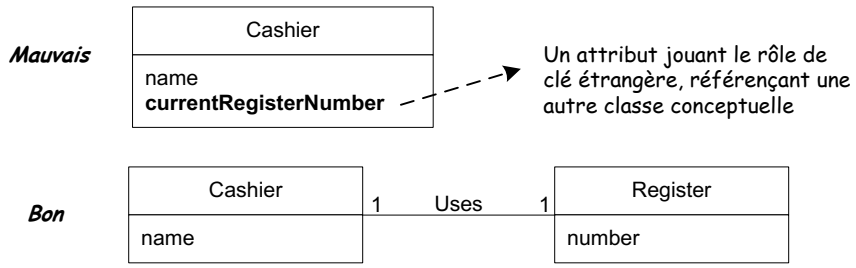
Il est donc impossible de répondre de manière catégorique à ce problème.

☺ Un attribut n'est pas une clé étrangère !

Une clé étrangère est une référence qui associerait une classe conceptuelle avec une autre.

Une clé étrangère est du domaine de la conception et de l'implémentation. Dans la modélisation de domaine, ces dernières sont simplement représentées par la ligne d'association correspondante. Plus tard, dans les classes logicielles, elles apparaîtront sous la forme d'attribut et seront visuellement mises en évidence par le symbole de navigabilité des associations.

Figure 3. Clés étrangères



4.3 HEURISTIQUE: COMMENT IDENTIFIER LES CLASSES CONCEPTUELLES

L'identification des classes conceptuelles requiert un peu de pratique - comme toujours ! -.

Analyse linguistique

Une première méthode consiste à analyser les descriptions textuelles du problème, à repérer les **noms** et à les considérer comme des **classes conceptuelles** ou des **attributs** potentiels.

On peut se baser par exemple sur les textes rédigés dans le cadre de la définition des *cas d'utilisation*.

Les **verbes** repérés dans ces descriptions textuelles correspondent généralement à des **associations** entre les différentes classes conceptuelles. Il se peut toutefois que l'action correspondante soit prise en compte sous la forme d'une classe conceptuelle.

> ✍ *Cette méthode doit être appliquée avec prudence et beaucoup de recul car le langage naturel sur lequel elle s'appuie comporte beaucoup d'imprécision et d'ambiguïté. Elle a le mérite d'être extrêmement naturelle et constitue une approche idéale pour le débutant.*

☺ **Un conseil...**

Attribut ou classe conceptuelle?

Une difficulté que l'on rencontre fréquemment, lorsque l'on a repéré un *nom* (p.e. : `Vente`, `date`, ..), est d'hésiter entre la notion *d'attribut* et la notion *de concept* (classe conceptuelle).

Voici une petite règle..

- Un concept est une entité qui possède une certaine identité. Il existe et se justifie en lui-même: il pourra être «manipulé» pour lui-même indépendamment des autres entités.
- Un attribut ne se justifie pas à lui tout seul et n'a de sens qu'associé au concept qu'il caractérise.

Une «`Vente`» constitue par exemple un bon exemple de classe conceptuelle: elle se justifie en elle-même, on peut l'enregistrer, la supprimer du registre, modifier ces caractéristique, etc.

Une «`date`» constitue plutôt un bon exemple d'attribut. Une date n'a souvent pas de sens en elle-même sinon pour caractériser un concept. On parlera par exemple de la «`date` **`de la`** `vente xy`» mais pas de la «`date`» tout court.

Patterns d'analyse

Une deuxième approche (en dehors de l'analyse linguistique) consiste à mettre en œuvre des modèles d'analyse («patterns») que l'on trouvera décrits dans des ouvrages spécialisés comme par exemple «Analysis patterns» de Martin Fowler ou encore «Data model Patterns» de D. Hay.

L'expérience..

Enfin, une troisième approche consiste à se baser sur *sa propre expérience*...

Blague mise à part, voici un tableau qui contient un certain nombre de catégories courantes de classes conceptuelles que l'on retrouve de manière habituelle.

A titre d'illustration, le domaine des ventes et celui de la réservation aérienne.

Classes conceptuelles	**Par exemple**
Objets physiques	Avion, Place, Registre
Lieux	Aéroport, Enregistrement
Rôle de personne	Secrétaire, Hôtesse, Pilote
Conteneurs	Entrepôt, Armoire, Catalogues
Transaction	Réservation, Vente, Paiement
Evènement	Annulation, Vente, Retard
Organisation	Département, Impôts
Système informatique externe	Contrôleur de taxe, Contrôleur de trafic
Concept (abstrait)	PeurDuVide

Processus	Inventaire, Réservation
Document	Contrat, Billet, Devis, Autorisation
Outils et instruments	Calculatrice
Manuels, Descriptions, spécifications	SpécificationDunProduit, ManuelDeRéparation

4.4 HEURISTIQUE: LISTE DES ASSOCIATIONS COURANTES

Comme pour l'identification des classes conceptuelles, on peut se baser sur l'analyse linguistique (voir plus haut : les verbes repérés dans ces descriptions textuelles correspondent généralement à des associations entre les différentes classes conceptuelles).

Mais on peut se baser également sur «l'expérience» en donnant une liste des associations typiques que l'on rencontre en modélisation de domaine.

Types d'associations entre deux classes conceptuelles A & B	***Par exemple***
A est un élément physique de B	Livre>Rayon
A est un élément logique de B	Entête>Protocole
A est un membre de B	Pilote>Compagnie
A décrit B	DescriptionDeVol>Vol
A communique avec B	Caissier>Client
A utilise B	Caissier>Registre
A contrôle B	Pilote>Avion
A est voisin de B	VilleA>VilleB
A est un événement associé à B	Départ>Vol, Achat>Client

Nom à donner aux associations

Par convention, les associations seront désignées par un **verbe**, en commençant par une majuscule :

- `Appartient-A`
- *ou* `AppartientA`

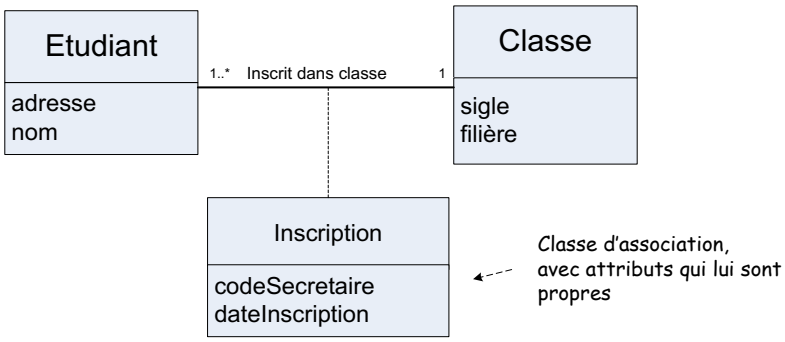
4.5 CLASSES D'ASSOCIATION

Il est possible de rajouter des attributs aux associations elles-mêmes.

☺ En règle générale, on ajoutera les attributs à l'association elle-même plutôt qu'aux classes associées lorsqu'on remarquera que la durée de vie de ces attributs dépend directement de l'association et non pas des objets associés.

En UML, cela se représente en traçant une ligne pointillée entre la ligne d'association et la classe d'association.

Figure 4. Classe d'association en UML



4.6 ARITE D'UNE ASSOCIATION

Définition 3: Arité

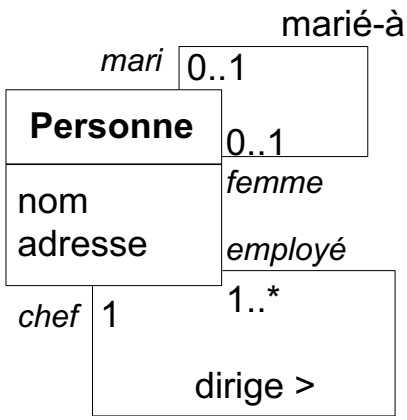
L'arité d'une association correspond au nombre d'entités qui participent à l'association.

Au niveau terminologique, une association d'arité n est dite association «**n-aire**».

Comme l'ont montré les différents exemples que nous avons rencontrés, la majorité des associations sont des **associations binaires**.

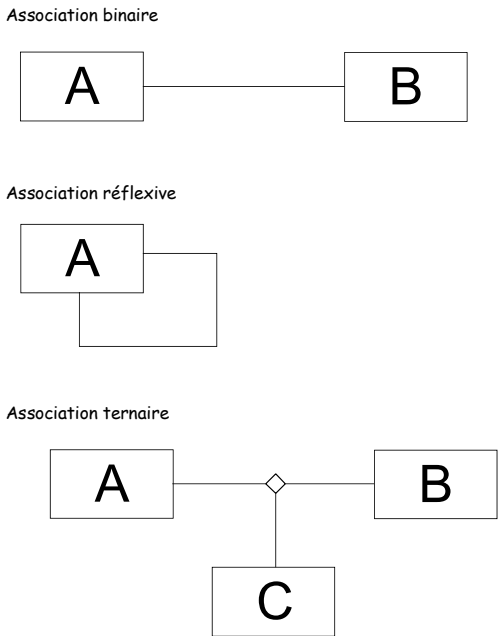
Un cas particulier d'association binaire est l'association dite **réflexive**, qui associe deux entités du même type. L'association «`marié à`» ou encore «`dirige`» reliant deux personnes en est une illustration.

Figure 5. Association réflexive



On rencontre encore fréquemment des associations ternaires, comme illustré dans le paragraphe suivant.

Figure 6. Arité d'une association

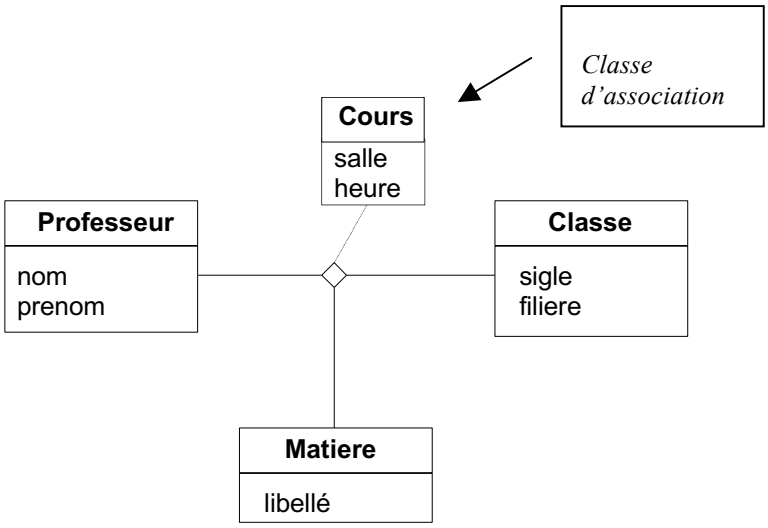


4.7 ASSOCIATIONS TERNAIRES

Se dit d'une association mettant en correspondance 3 types objets.

La figure ci-dessous nous présente l'association «Cours», qui relie les 3 types d'objets `Professeur`, `Classe` et `Matiere`.

Figure 7. Un «cours» comme association ternaire



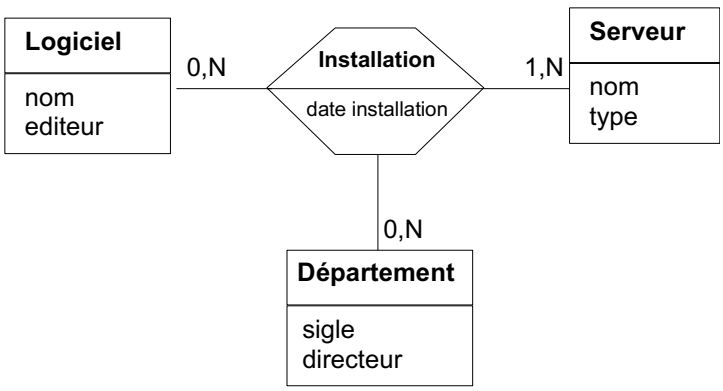
On notera que la notation UML utilise un **losange** sur lequel se trouvent connectés les 3 types d'objets. Le même losange peut être connecté à une *classe d'association* dans laquelle on retrouve le ou les attributs de l'association.

Cardinalité des associations ternaires au sens Merise et UML

Nous allons considérer un deuxième exemple, celui d'une entreprise qui désire conserver une trace de l'installation des logiciels dans chacun de ses départements. A cet effet, une association ternaire nous permettra d'enregistrer la date à laquelle tel ou tel logiciel a été installé pour un département donné sur un serveur spécifique.

La notation *Merise* est utilisée dans le cadre de la modélisation des bases de données.

Figure 8. Association ternaire et notation Merise



On notera que l'association ternaire possède un attribut : la date d'installation du logiciel.

Pour mieux comprendre le sens donné aux cardinalités indiquées sur ce schéma, donnons à titre d'illustration l'allure que pourrait prendre la table relationnelle que l'on utiliserait pour mettre en œuvre cette association ternaire.

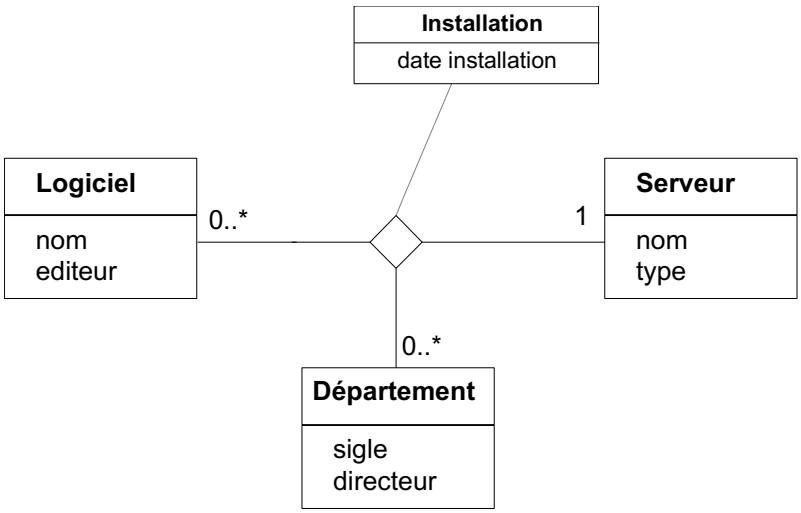
Logiciel	***Département***	***Serveur***	***Date d'installation***
Word	Info	ServeurINFO_1	1er janvier 01
Word	Tcom	ServeurTCOM	12 mars 01
Rational Rose	Info	ServeurINFO_1	15 juillet 02
PowerAMC	Tcom	ServeurINFO_2	1er sept 02
PowerAMC	Info	ServeurINFO_2	1er sept 02

Au sens de Merise, les cardinalités indiquées sur le modèle doivent être interprétées de la manière suivante :

- **Logiciel - [0, N]**
  Le même logiciel peut apparaître plusieurs fois dans la table d'association, ou ne pas apparaître du tout. Autrement dit, le même logiciel peut être associé à 0 ou à plusieurs couples `<Département, Serveur>`.
- **Département - [0, N]**
  Le même département peut apparaître plusieurs fois dans la table d'association, ou ne pas apparaître du tout. Autrement dit, le même département peut être associé à 0 ou à plusieurs couples `<Logiciel, Serveur>`.
- **Serveur - [1, N]**
  Le même Serveur peut apparaître plusieurs fois dans la table d'association, il apparaîtra au moins une fois étant donné qu'un serveur donné doit héberger au moins un logiciel. Autrement dit, le même serveur peut être associé à un ou à plusieurs couples `<Département, Logiciel>`.

Voici enfin toujours le même schéma conceptuel, mais répondant cette fois-ci au **modèle UML**.

Figure 9. Associations ternaires et UML



Au sens UML, et en adoptant le formalisme utilisé par WinDesign (un outil de modélisation), les cardinalités indiquées sur le modèle seront interprétées de la manière suivante :

- **Logiciel - [0..*]**
  Considérant un couple `<Département, Serveur>`, plusieurs logiciels pourront y être installés. Par contre il est possible qu'aucun logiciel n'y soit installé. D'où la cardinalité `[0..*]`.
- **Département - [0..*]**
  Considérant un couple `<Logiciel, Serveur>`, ce même couple pourra être installé pour plusieurs départements. En revanche, il est possible qu'aucun département ne soit concerné par cette installation.
  D'où la cardinalité `[0..*]`.
- **Serveur - [1]**
  Considérant un couple `<Département, Logiciel>`, l'installation sera opérée impérativement sur un serveur et un seul.

✍ On peut noter *dans ce cas particulier* que l'interprétation des cardinalités est plus riche que celle introduite dans la notation Merise. En effet, elle permet de prendre en compte les contraintes d'unicité du type «qu'un logiciel d'un département ne doit être installé que sur un seul serveur».

Les relations ternaires sont rarement utilisées

La mise en évidence d'associations ternaires intéresse avant tout la **phase d'analyse**.

✍ Pendant les **phases de conception et d'implémentation**, quand il s'agira de mettre en œuvre des classes dites «logicielles», on se contera d'associations binaires. Les associations ternaires seraient trop lourdes à manipuler.

> ☹ *Notons par ailleurs que la représentation des associations ternaires n'est pas proposée à l'heure actuelle par les outils Rational Rose et Together. Outils dédiés principalement à la conception et à l'implémentation.*

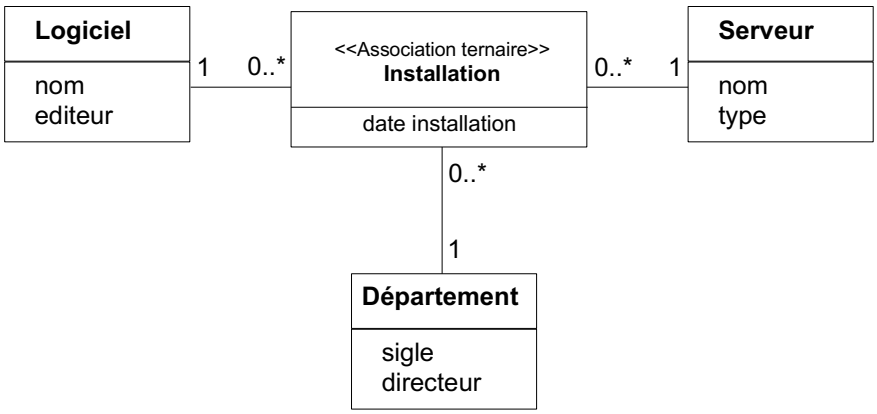
Alors.. comment représenter les associations ternaires ??

1ère approche : avec un stéréotype...

Dans une première approche, il est possible de représenter une association ternaire au moyen d'une *classe d'associations* portant le stéréotype **<<**`Association ternaire`**>>**.

Ce qui nous donne le schéma suivant...

Figure 10. Un stéréotype <<association ternaire>>



Dans cette représentation, les cardinalités indiquées sur le modèle sont interprétées selon le sens UML habituel :

- **Logiciel - Installation**
  Une installation concerne un et un seul logiciel. Un logiciel peut être installé plusieurs fois.
- **Département**
  Une installation concerne un et un seul département. Plusieurs installations ont pu être opérées pour le même département.

- **Serveur**
  Une installation est opérée sur un seul serveur. Le même serveur peut abriter plusieurs installations.

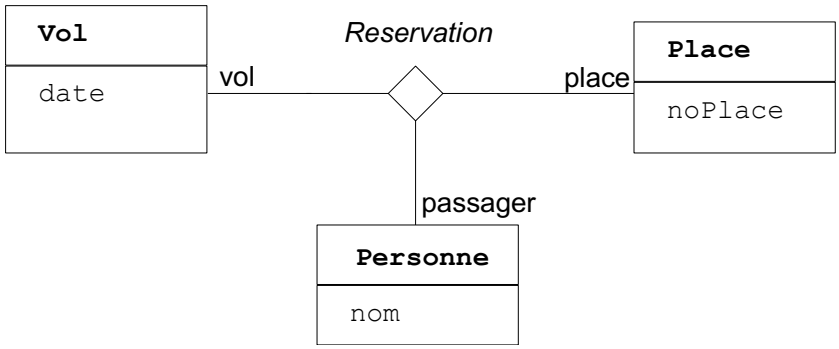
ATTENTION !!
Si on omet le stéréotype, on a l'impression que nos 4 classes sont reliées par 3 associations **indépendantes** l'une de l'autre. Ce qui n'est pas le cas : toute nouvelle installation agira sur les 3 associations simultanément !

✍ Notons que cette représentation ne permet pas, - dans notre exemple -, de prendre en compte la contrainte d'unicité «un logiciel d'un département ne doit être installé que sur un seul serveur». Par exemple, rien n'interdit à ce que le logiciel «Office» soit installé deux fois pour le même département, sur deux serveurs différents.

2ème approche : Binéarisation de l'association ternaire

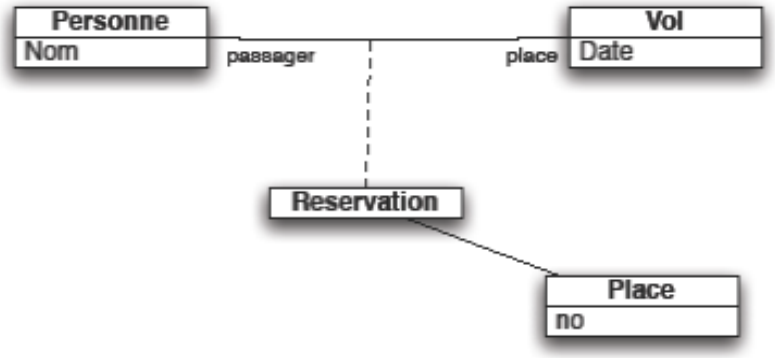
Dans cette deuxième approche, les associations ternaires sont remplacées par une association binaire à laquelle on rattache une classe d'associations (cette dernière contient alors les éléments intéressants du 3ème type d'objets, celui qui participait à l'association et que l'on a supprimé).

Considérons à titre d'illustration un système de réservation de place d'avion.

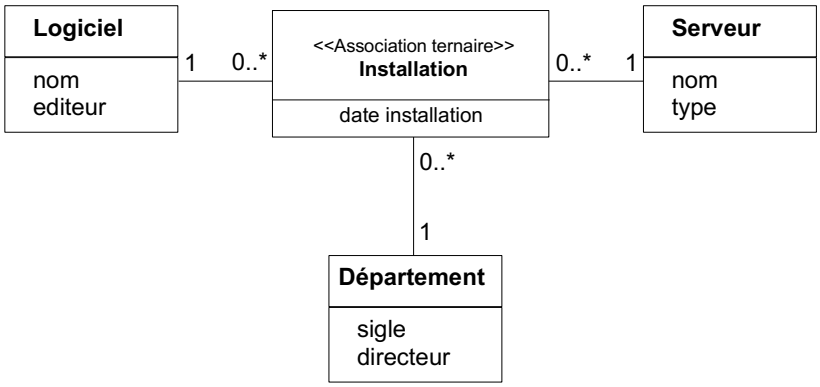


La transformation de cette association ternaire consiste à supprimer une des 3 classes, - en l'occurrence la classe Place -, en la remplaçant par une association binaire à laquelle on rattache tous les attributs de la classe supprimée.

Transformation d'une association, de ternaire à binaire



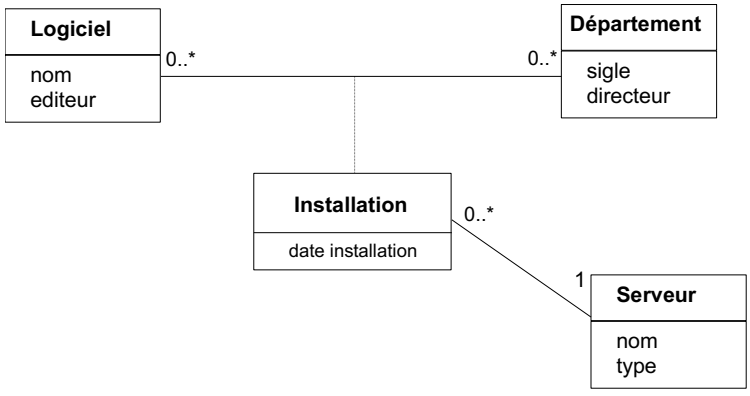
A titre d'illustration, reprenons maintenant l'exemple relatif à l'installation des logiciels et sa représentation UML avec l'outil Rational Rose. Rappelons que cette représentation ne permettait pas de prendre en compte la contrainte d'unicité «un logiciel d'un département ne doit être installé que sur un seul serveur».



Dans ce schéma, le concept «d'installation» représente une «une installation d'un *logiciel* pour un *département* sur un *serveur*».

Après transformation en association binaire, on retrouve la possibilité de représenter cette contrainte, comme l'illustre la figure suivante.

Figure 11. Transformation ternaire - binaire, deuxième exemple



Cette fois-ci, le concept «d'installation» représente non plus «une installation d'un *logiciel* pour un *département* sur un *serveur*» mais plutôt «une installation d'un logiciel pour un département». C'est peut-être un peu subtil, mais toute la différence est là.

4.8 AGREGATIONS ET COMPOSITIONS

UML opère un distinguo entre associations simples, associations de type agrégation et associations de type composition.

En préliminaire, il est bon de revenir sur certaines notions fondamentales.

Retour sur la notion d'attribut

- A la différence d'un objet, **un attribut n'a pas de sens à lui tout seul**. Un attribut ne sera jamais manipulé indépendamment de l'objet qu'il caractérise.
  Ainsi, on parlera de la hauteur d'un immeuble, de l'âge d'un individu, etc.

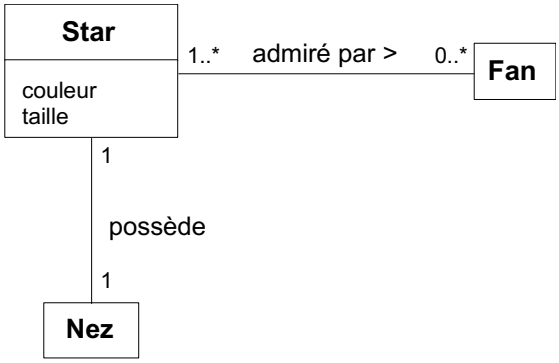
- **Supprimer un attribut n'a pas de sens** : l'objet perdrait son identité (son sens, sa raison d'exister).
  Par contre il est possible de modifier la valeur d'un individu.

☺ *Une personne ne peut pas perdre sa couleur ! Elle peut par contre changer de couleur.*

Retour sur la notion d'association

Au contraire des attributs, les associations peut être supprimées (la référence prend alors la valeur null), sans altérer l'identité de l'objet, même si ce dernier peut en devenir moins performant.

Figure 12. Le nez ne peut être qu'une association



Une personne ne peut pas perdre sa couleur ni sa taille (même si cette dernière prend la valeur 0, la personne aura toujours une taille). Par contre, une personne peut perdre son nez, une jambe, etc.

Cet exemple montre également que le nez d'une personne peut être manipulé en tant que tel, indépendamment de la personne à laquelle il appartient. Par contre, il serait difficile de prendre la couleur d'une personne, de la mettre dans un sac, et de lui changer sa valeur indépendamment de la personne.

Les associations simples

Les associations simples dénotent un lien faible entre les classes mises en relation : les cycles de vie des objets sont très indépendants les uns des autres. C'est le cas par exemple de la relation entre une star et un de ses fans (quoique...).

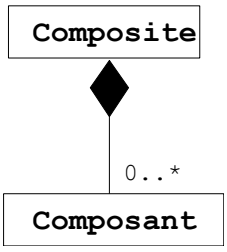
Les associations de type composition

Ces dernières, représentées par un losange rempli de noir du côté composite, dénotent au contraire un lien très fort entre le «**composite**» et le «**composant**», notamment du point de vue de leur cycle de vie.

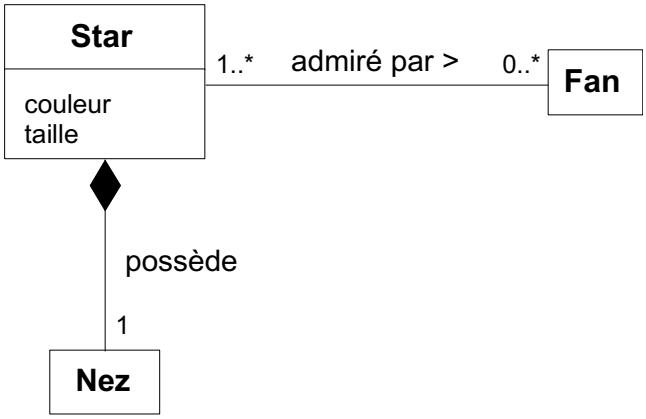
Définition 4: Composition

- Si le composite est détruit, ses composants le sont également !
- Le partage n'est pas possible.
  Un composant ne peut appartenir qu'à un seul composite.
  La cardinalité peut valoir 1 ou 0..1 du côté composite.

Figure 13. La composition en UML



Ainsi, le diagramme de la «star» peut tenir compte de ce type d'association:



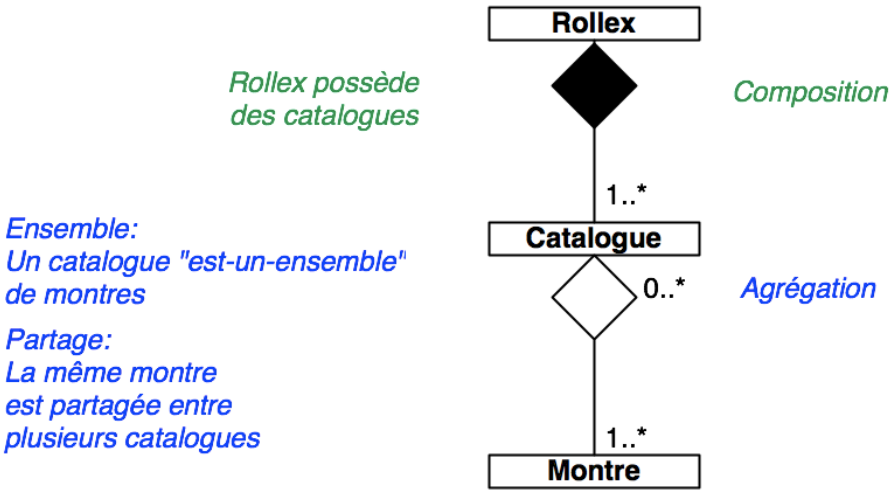
Les agrégations

Une agrégation dénote une association de type «**ensemble-élément**». Elles sont représentées au moyen d'un losange non rempli de noir, placé du côté de l'ensemble.

Définition 5: Agrégation

- les cycles de vie de l'ensemble et de ses éléments sont relativement indépendants
- le partage est possible: un «élément» peut faire partie le cas échéant de plusieurs «ensembles».

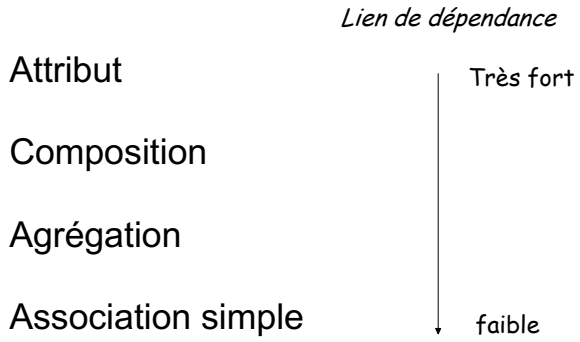
Figure 14. L'agrégation en UML



Hiérarchie Attribut > Composition > Agrégation > Association simple

Les lecteurs qui ont bien assimilé le discours qui précède auront remarqué qu'il existe une hiérarchie **du point de vue de la dépendance** entre les différentes notions d'attributs, de composition, d'agrégation et d'association simple.

Il existe un lien très fort, existentiel, entre un attribut et l'objet qu'il caractérise. Au contraire, une association simple dénote un lien de dépendance très faible entre les objets mis en relation.

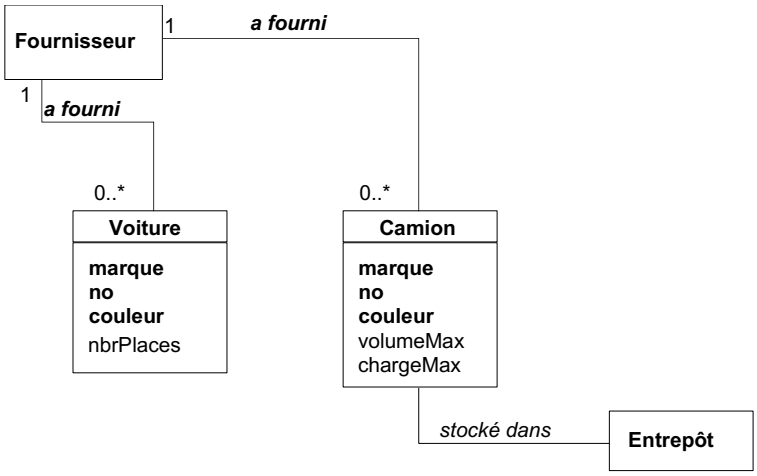


4.9 GENERALISATION - SPECIALISATION

> Ce concept dénote la possibilité de regrouper dans une seule classe toutes les propriétés (c.a.d les attributs et les associations) qui seraient communes à un ensemble de classes spécifiques.

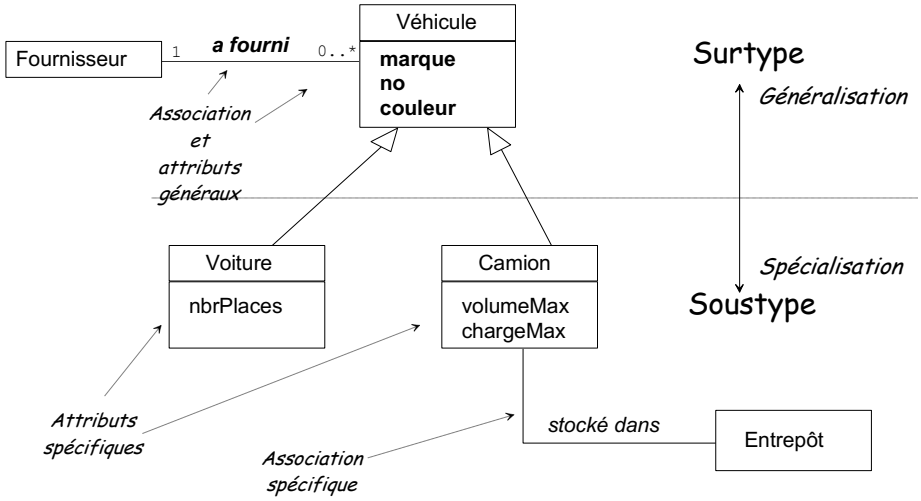
Prenons l'exemple d'une entreprise de location de véhicules.

Dans la figure suivante, nous remarquons que les deux classes `Voiture` et `Camion` sont toutes deux des véhicules et partagent à ce titre un certain nombre de caractéristiques. Le fait notamment d'avoir été obtenu auprès d'un fournisseur (association «`a fourni`»), mais aussi le fait d'être caractérisés par trois attributs identiques (une marque, un numéro d'identification et une couleur).



Le concept «Gen-Spec» (Généralisation - Spécialisation) appliqué à cet exemple nous permet de regrouper ces caractéristiques communes à l'intérieur d'une seule et même classe. Dans notre cas, il s'agira de la classe `Vehicule`, comme le montre la figure suivante.

Figure 15. Généralisation – Spécialisation



Pour quel bénéfice?

Le schéma conceptuel qui en résulte est beaucoup plus «lisible». D'autre part, toute modification ne concernant que les caractérisques générales (comme par exemple de changer le nom de l'attribut `noIdent` par `noIdentification`) ne sera opérée qu'à un seul endroit, tout simplement et sans erreur.

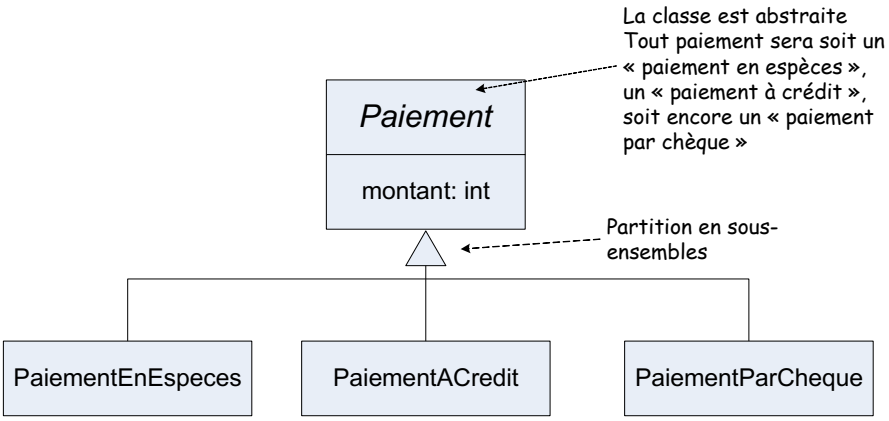
4.10 REPRESENTATION DES CLASSES ABSTRAITES

En UML, un nom de classe écrit en *italique* indique que cette dernière est une classe abstraite (une classe qui ne sera jamais instanciée, seules ses sous-classes le seront).

Définition 6: Classes abstraites et modélisation de domaine

Du point de vue de la modélisation de domaine, une classe conceptuelle sera spécifiée abstraite dans la mesure où l'ensemble de toutes ses sous-classes opère une **partition**. Autrement dit, tout membre de la classe abstraite doit être membre d'une sous-classe.

Figure 16. Représentation d'une classe abstraite



5 Exercices

5.1 Exo 1 - Courses de Canoë-Kayaks

Les courses de canoë-kayaks sont constituées de plusieurs manches.

Les compétiteurs doivent participer à toutes les manches et, pour chacune d'entre elles, doivent choisir une catégorie (à savoir le type de canoë).

Un « **parcours »** dénote le fait de participer à une manche.

Il s'agit d'une relation ternaire entre :

- 1 compétiteur
- 1 catégorie
- 1 manche

Chaque parcours sera caractérisé par le temps réalisé et le nombre de pénalités obtenues.

Modéliser la notion de parcours en utilisant les 3 formes possibles de représentation des relations ternaires en UML.

Note : Pour les cardinalités, on supposera qu'il y a en tout :

- 3 manches
- 100 compétiteurs
- 5 catégories de canoës

5.2 EXO 2 - LES PECHEURS

Gestion des droits de pêche sur les cours d'eau du canton

- Les pêcheurs, que l'on enregistrera avec leur nom et leur adresse, sont regroupés en associations. Un pêcheur peut faire ou non partie d'une association. Ils peuvent le cas échéant faire partie de plusieurs associations.

- Les cours d'eau, que l'on distingue par leur nom, sont abondamment peuplés de poisson (truite, brochet, ..). Les cours d'eau sont régulièrement contrôlés. On enregistre à cette occasion la population estimée de chaque type de poisson.

- Chaque type de poisson est caractérisé par une taille minimum autorisée pour la pêche. D'un cours d'eau à l'autre, pour tenir compte des derniers contrôles de population, cette taille minimum peut être ajustée.

- L'entretien de chaque cours d'eau est confié à une ou à plusieurs associations de pêcheurs. Chaque association bénéficie alors d'un rabais qui lui est spécifique (dont bénéficient les pêcheurs de l'association lors de l'achat d'un permis). Chacune des associations peut participer à l'entretien de plusieurs cours d'eau.

- Un pêcheur peut obtenir un permis de pêche. Un permis de pêche est spécifique à un cours d'eau particulier. Chaque permis est caractérisé par une date de début et une date de fin.

- Chaque cours d'eau met à disposition un certain nombre de place d'amarrage, caractérisées chacune par un no et une taille. Un pêcheur peut louer des places d'amarrage pour y laisser ses embarcations.

- Les lacs et les étangs – deux types de cours d'eau - seront caractérisés par une surface. Les rivières – un troisième type de cours d'eau - peuvent se rattacher à des lacs (elles se jettent dans les lacs ou les traversent).

Etape 1
Dresser un modèle de domaine avec Together. Se limiter à des simples associations binaires, en ignorant les notions de composition et d'agrégation.

Etape 2
Compléter/modifier le modèle de domaine en tenant compte des deux éléments suivants :

- La loi cantonale sur la pêche a été modifiée : les pêcheurs payeront ! (ce sont des passionnés).
  Dans cette nouvelle loi, un permis de pêche sera toujours spécifique à un cours d'eau particulier mais ne concernera qu'un seul type de poisson.

- Des repeuplements seront opérés régulièrement. On aimerait enregistrer le fait, par exemple, que les associations « Le doux pêcheur viril» et « Le poisson s'appâte» se sont regroupées 1er Mai 2011 pour effectuer le repeuplement de la truite dans la Venoge à raison de 5'000 unités.

Etape 3
Compléter le modèle de domaine en tenant compte des notions de composition et d'agrégation.

5.3 EXO 3 – STAGES DE CHANT

Modélisation de domaine - Inscrivez-vous au stage!

L'association régionale des théâtres organise des stages de chants et propose en outre des cours individuels :

1. Chaque stage est proposé par un théâtre. Il est défini par une date de début, une durée (en nombre de jours) et un prix.
2. Le cas échéant, chaque stage peut nécessiter le prérequis d'un autre stage *(pour pouvoir s'y inscrire, un étudiant doit avoir suivi le stage prérequis*).
3. Les professeurs, désignés par un nom, sont engagés par les théâtres. Ils peuvent être engagés par plusieurs théâtres.
4. La dispense d'un stage est opérée par un professeur (le professeur responsable) et un second professeur jouant le rôle d'assistant. Premier et second professeurs doivent être engagés par le théâtre qui propose le stage.
5. Les étudiants, désignés par un nom et une adresse, s'inscrivent auprès d'un ou de plusieurs théâtres.
6. Pour chaque inscription de stage, on enregistre la date de l'inscription ainsi que le montant des arrhes versés par l'étudiant. Un étudiant peut-être inscrit à plusieurs stages.
7. Sont également proposés par les théâtres des cours individuels auxquels les étudiants peuvent être inscrits en indiquant le professeur souhaité. Pour un cours donné, un étudiant ne peut être inscrit qu'une seule fois.
8. Les théâtres mettent à disposition deux types de salles : Salles d'échauffement et salles de chant (technique vocale)
9. Chacune des salles est désignée par un numéro.
10. Chacun des stages utilise une salle d'échauffement et deux salles de chant, ces 3 salles devant appartenir au théâtre qui propose le stage.

Remarques générales concernant les exercices de modélisation

1. Nommer les associations ;
2. Indiquer précisément les cardinalités ;
3. Ne pas rajouter d'informations qui ne se trouvent pas exprimées dans l'énoncé.
4. Toutefois, si l'énoncé vous paraît ambigu (cela peut arriver...), vous pouvez le compléter. Indiquez-le par une remarque.
5. Utiliser **l'agrégation** ou la **composition** partout où cela s'y prête. Accompagner votre schéma d'une justification (pourquoi agrégation plutôt que composition).
6. Utiliser la « **Généralisation-Spécialisation** » partout où cela s'y prête.
7. Signalez les contraintes que votre modèle ne montre pas