

Modélisation avancée en UML et en relationnel

*stph.scenari-community.org/bdd
mod3.pdf*



Stéphane Crozat

Table des matières



Objectifs	4
I - Cours	5
1. Modélisation avancée des associations en UML	5
1.1. Exercice : Entreprise	5
1.2. Composition	5
1.3. Agrégation	7
1.4. Explicitation des associations (sens de lecture et rôle)	7
1.5. Associations réflexives	8
1.6. Notion de clé locale dans les compositions et les associations N:M	8
1.7. Classe d'association avec clé locale	9
1.8. Associations ternaires	10
2. Passage UML-Relationnel : Associations avancées	11
2.1. Trousseau de clés	11
2.2. Transformation des compositions	12
2.3. Transformation des agrégations	13
2.4. Transformation des classes d'association avec clé locale	14
2.5. Correspondance entre UML et relationnel	15
2.6. Exercice	16
2.7. Exercice	17
3. Modélisation avancée des associations 1:1 en relationnel	18
3.1. Transformation des associations 1:1 (approche générale)	18
3.2. Transformation des associations 1..1:1..1	19
3.3. Transformation des associations 0..1:1..1	20
3.4. Transformation des associations 0..1:0..1	20
3.5. Exemple de choix pour une relation 1:1	21
4. Autres éléments utiles en UML : packages et stéréotypes	21
4.1. Paquetages	21
4.2. Stéréotype	23
4.3. Énumération : stéréotype <<enumeration>>	23
4.4. Type utilisateurs : stéréotype <<dataType>>	24
5. Synthèse sur la modalisation UML et relationnelle	26
5.1. Quelques éléments de stylistique UML	26
5.2. Attention aux clés artificielles	26
5.3. Bibliographie commentée sur la modélisation UML	27
5.4. Synthèse : Les diagrammes de modélisation conceptuelle	29
II - Exercices	30
1. Exercice : Lab III	30
2. Exercice : Étudiants et UVs (introduction)	31
3. Exercice : Super-héros relationnels I	32

4. Exercice : Objets Numériques Libres	32
III - Devoirs	34
1. Exercice : Arbre de scène 3D	34
IV - Complément : Exercices de modélisation supplémentaire	35
1. Exercice : Appartements à louer	35
2. Exercice : Objectifs	35
Contenus annexes	38
Questions de synthèse	51
Solutions des exercices	53
Glossaire	63
Abréviations	64
Bibliographie	65
Webographie	66

Objectifs



Prérequis :

- Savoir faire un MCD UML avec des classes, des associations simples, de l'héritage.
- Savoir faire un MLD relationnel à parti d'un MCD UML avec des classes, des associations simples, de l'héritage.



Cours

I

1. Modélisation avancée des associations en UML

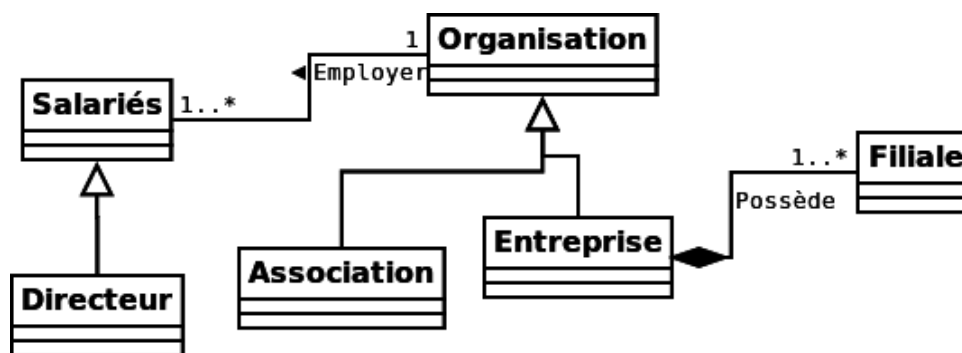
Objectifs

Maîtriser le diagramme de classe UML dans le cas de la conception de BD.

1.1. Exercice : Entreprise

[solution n°1 p.53]

En analysant le schéma UML ci-après, sélectionner toutes les assertions vraies.



MCD UML

- Une association peut employer un directeur.
- Une association peut employer plusieurs directeurs.
- Une association peut ne pas employer de directeur.
- Une filiale peut appartenir à plusieurs entreprises.
- Il existe des organisations qui ne sont ni des entreprises ni des associations.

1.2. Composition

 *Définition : Association de composition*

On appelle composition une association particulière qui possède les propriétés suivantes :

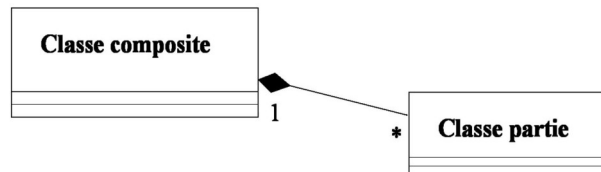
- La composition associe une classe composite et des classes parties, tel que tout objet partie appartient à un et un seul objet composite. C'est donc une association 1:N (voire 1:1).
- La composition n'est pas partageable, donc un objet partie ne peut appartenir qu'à un seul objet composite à la fois.

- Le cycle de vie des objets parties est lié à celui de l'objet composite, donc un objet partie disparaît quand l'objet composite auquel il est associé disparaît.

Remarque

- La composition est une association particulière (binaire de cardinalité contrainte).
- La composition n'est pas symétrique, une classe joue le rôle de conteneur pour les classes liées, elle prend donc un rôle particulier a priori.
- La composition est une agrégation avec des contraintes supplémentaires (non partageabilité et cycle de vie lié).

Syntaxe : Notation d'une composition en UML



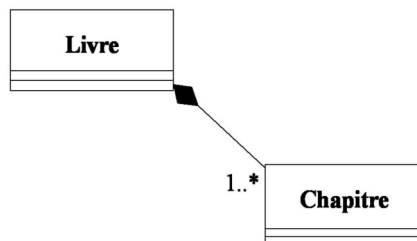
Notation de la composition en UML

Attention : Composition et cardinalité

La cardinalité côté composite est toujours de exactement 1.

Côté partie la cardinalité est libre, elle peut être 0..1, 1, * ou bien 1..*.

Exemple : Exemple de composition



Un livre

On voit bien ici qu'un chapitre n'a de sens que faisant partie d'un livre, qu'il ne peut exister dans deux livres différents et que si le livre n'existe plus, les chapitres le composant non plus.

Remarque : Composition et entités faibles

La composition permet d'exprimer une association analogue à celle qui relie une entité faible à une entité identifiante en modélisation E-A*. L'entité de type faible correspond à un objet partie et l'entité identifiante à un objet composite.

Conseil : Composition et attribut multivalué

- Une composition avec une classe partie dotée d'un seul attribut peut s'écrire avec un attribut multivalué.
- Un attribut composé et multivalué peut s'écrire avec une composition.

◆ Rappel : Voir aussi

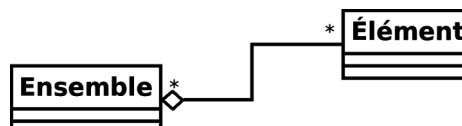
- *Attributs* (cf. p.38)
- *Agrégation* (cf. p.7)

1.3. Agrégation

🔑 Définition : Association d'agrégation

L'agrégation est une association particulière utilisée pour préciser une relation tout/partie (ou ensemble /élément), on parle d'association *méréologique*.

📦 Syntaxe



Notation de l'agrégation en UML

La cardinalité, peut être exprimée librement, en particulier les instances de la classe *Élément* peuvent être associées à plusieurs instances de la classe *Ensemble*, et même de plusieurs classes.

⚠ Attention

L'agrégation garde toutes les propriétés d'une association classique (cardinalité, cycle de vie, etc.), elle ajoute simplement une terminologie un plus précise via la notion de tout/partie.

1.4. Explicitation des associations (sens de lecture et rôle)

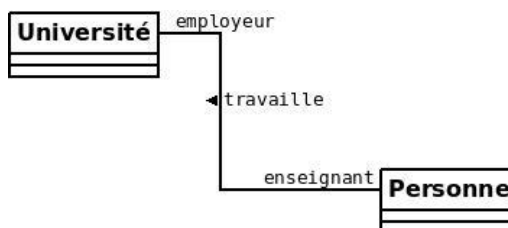
📦 Syntaxe : Sens de lecture

Il est possible d'ajouter le sens de lecture du verbe caractérisant l'association sur un diagramme de classe UML, afin d'en faciliter la lecture. On ajoute pour cela un signe < ou > (ou un triangle noir) à côté du nom de l'association

📦 Syntaxe : Rôle

Il est possible de préciser le rôle joué par une ou plusieurs des classes composant une association afin d'en faciliter la compréhension. On ajoute pour cela ce rôle à côté de la classe concernée (parfois dans un petit encadré collé au trait de l'association).

👉 Exemple



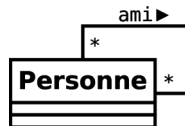
Rôle et sens de lecture sur une association

1.5. Associations réflexives

Définition : Association réflexive

Une association réflexive est une association qui associe une classe avec elle-même.

Exemple



Association réflexive « ami »

Méthode

L'explicitation des associations est souvent utile dans le cas des associations réflexives non symétrique (ou chaque objet ne joue pas le même rôle).

Attention : Auto-association dans les associations réflexives

Une instance peut être associée avec elle-même dans le cas de d'une association réflexive.

Si l'on souhaite exprimer le contraire (une instance peut être associée avec d'autres instances de la même classe, mais pas avec elle-même) :

- on ajoute une contrainte en UML (par exemple {les personnes ne se marient pas avec elles-mêmes}) ;
- que l'on traduira en relationnel par une contrainte du type AVEC $pk \neq fk$;
- que l'on traduira en SQL par une clause du type CHECK $pk \neq fk$.

1.6. Notion de clé locale dans les compositions et les associations N:M

Le concept de clé locale appartient au niveau conceptuel, il est hérité de l'entité faible du modèle conceptuel Entité-Association (équivalent de la composition en UML). Dans une entité faible ou une composition, une clé de la classe composant est dite locale, car elle ne permet d'identifier l'objet que si l'on connaît la classe composite.

Définition

Dans certaines constructions en UML (association N:M et composition) la clé peut être locale, c'est à dire qu'au lieu d'identifier pleinement un objet (comme une clé classique), elle identifie un objet étant donné un contexte (les membres de l'association N:M ou l'objet composite).

Attention

Une clé locale n'est donc pas une clé au sens relationnel, elle ne permet pas d'identifier un enregistrement, mais elle deviendra une partie d'une clé lors du passage au relationnel.

⚠ Attention

Dans une associations N:M on peut avoir des (vraies) clés ou des clés locales. En revanche dans une composition on n'a que des clés locales : en effet si le composant est identifiable indépendamment de son composite, c'est en général qu'il a une vie propre et donc que l'on est pas en présence d'une composition.

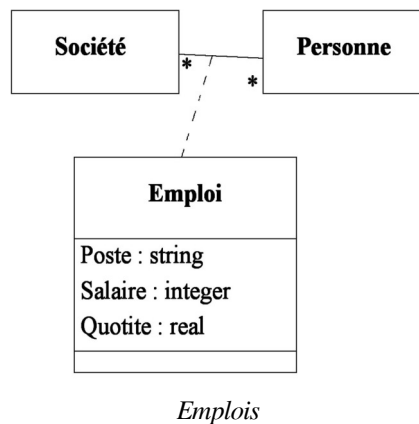
1.7. Classe d'association avec clé locale

📖 Rappel

Classe d'association (cf. p.40)

Contrainte inhérente à la relation N:M

Dans l'exemple suivant, chaque personne peut avoir un emploi dans plusieurs sociétés, mais elle ne peut pas avoir plusieurs emplois dans une même société.



La transformation en relationnelle est cohérente avec cette contrainte.

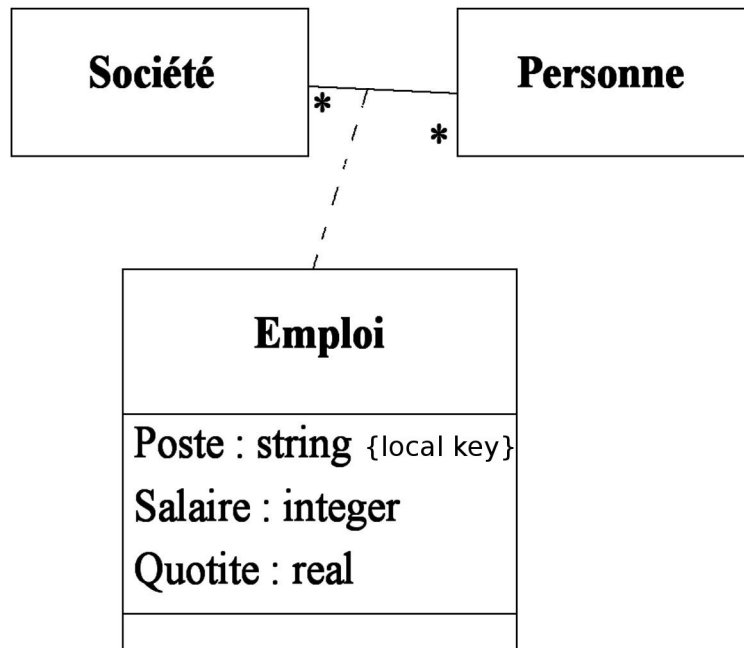
```

1 Société(...)
2 Personne(...)
3 Emploi(#personne=>Personne, #societe=>Societe, poste:string, salaire:integer,
  quotite:numeric(1,2))
  
```

#personne	#societe	poste	salaire	quotite
AI	Canonical	Directeur	100000	0,5
AI	Canonical	Développeur	50000	0,5

✂ Méthode : Intérêt de la clé locale

La spécification d'une clé locale dans emploi, par exemple ici le poste, permet de lever cette contrainte, lorsqu'on le souhaite, en permettant d'identifier chaque instance de l'association, ici l'emploi d'une personne par sa société.



La transformation en relationnelle permettra de maintenir la mod lisation, en ajoutant la cl  locale   la cl  initiale

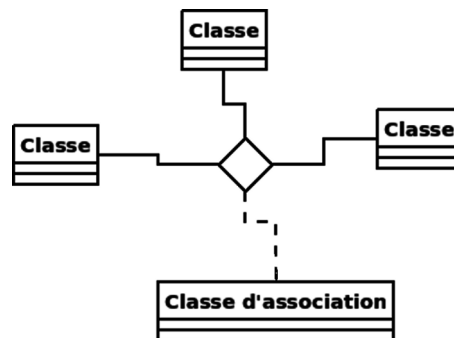
```

1 Soci t (...)
2 Personne(...)
3 Employe(#personne=>Personne, #societe=>Societe, #poste:string, salaire:integer,
quotite:numeric(7,2))
  
```

#personne	#societe	#poste	salaire	quotite
Al	Canonical	Directeur	100000	0,5
Al	Canonical	D�veloppeur	50000	0,5

1.8. Associations ternaires

Syntaxe



Notation d'une association ternaire

Conseil : Ne pas abuser des associations ternaires

Il est toujours possible de r crire une association ternaire avec trois associations binaires, en transformant l'association en classe.

Conseil : Pas de degré supérieur à 3

En pratique on n'utilise jamais en UML d'association de degré supérieur à 3.

2. Passage UML-Relationnel : Associations avancées

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel dans tous les cas.

Reconnaître les cas de transformation qui se traitent toujours de la même façon et ceux qui nécessitent une modélisation complémentaire.

2.1. Trousseau de clés

Attention

En UML et en relationnel, il existe plusieurs termes mobilisant le mot « clé », le seul concept qui est commun est le concept de *clé*. Tous les autres sont spécifiques au niveau conceptuel ou relationnel.

Rappel : Concept commun au niveau UML et relationnel

Clé (key)*

Rappel : Concept spécifique au niveau UML

Clé locale (local key)*

Rappel : Concepts spécifiques au niveau relationnel

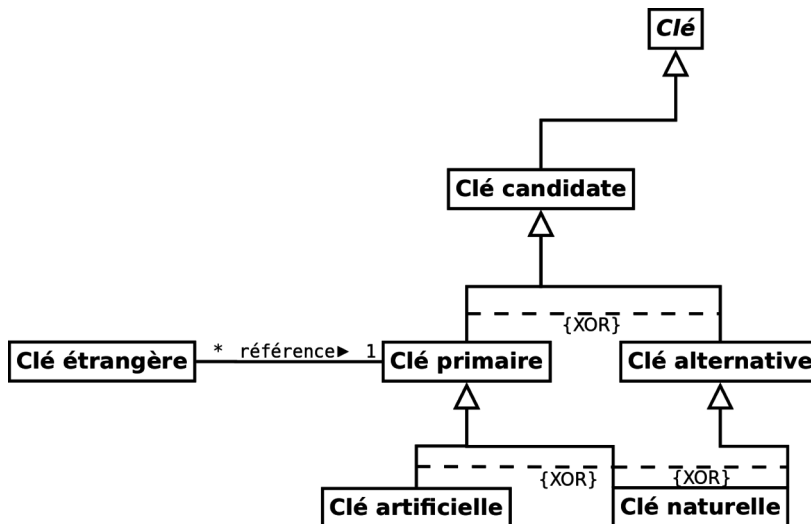
Clé (key)*

Clé candidate (candidate key)*

Clé primaire (primary key)* et Clé alternative (alternate key)*

Clé artificielle (surrogate key)* et Clé naturelle (natural key, business key)*

Clé étrangère (foreign key)*



Définition des clés en relationnel

2.2. Transformation des compositions

Méthode

Une composition

- est transformée comme une association 1:N,
- puis on ajoute à la clé de la classe partie (dite clé locale) la clé étrangère vers la classe composite pour construire une clé primaire composée.



Composition

Classe1 (#a , b)

Classe2 (#c , #a=>Classe1 , d)

Remarque : Clé locale

Pour identifier une classe partie dans une composition, on utilise une clé locale concaténée à la clé étrangère vers la classe composite, afin d'exprimer la dépendance entre les deux classes.

Si une clé naturelle globale permet d'identifier de façon unique une partie indépendamment du tout, on préférera la conserver comme clé candidate plutôt que de la prendre pour clé primaire.

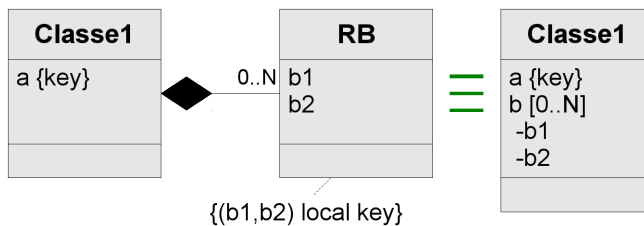
Si on la choisit comme clé primaire cela revient à avoir transformé la composition en agrégation, en redonnant une vie propre aux objets composants.

Complément : Composition et entités faibles en E-A

Une composition est transformée selon les mêmes principes qu'une entité faible en E-A.

Complément : Attributs multivalués et composés

La transformation d'un attribut composé multivalué donne un résultat équivalent à la transformation d'une composition.

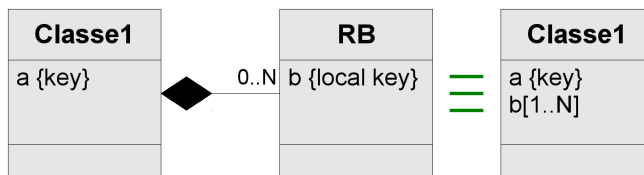


Composition et attribut composé multivalué

Classe1 (#a)

RB (#b_b1, #b_b2, #a=>Classe1)

La transformation d'une composition avec un seul attribut pour la classe composante donne un résultat équivalent à la transformation d'un attribut multivalué.



Composition et attribut multivalué

Classe1 (#a)

RB (#b, #a=>Classe1)

Rappel : Voir aussi

Transformation des attributs (cf. p.41)

2.3. Transformation des agrégations

Rappel : Agrégation

Les associations de type agrégation se traitent de la même façon que les associations classiques.



Agrégation 1:N

Classe1 (#a, b)

Classe2 (#c, d, a=>Classe1)



Agrégation N:M

Classe1 (#a , b)

Classe2 (#c , d)

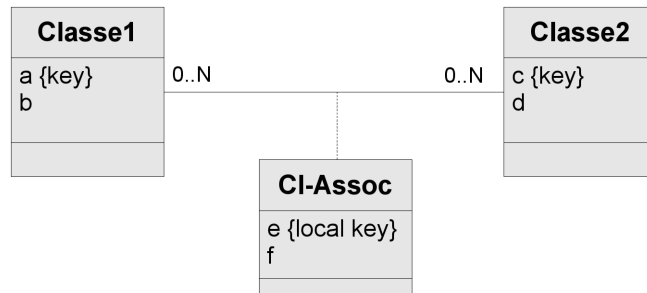
Assoc (#a=>Classe1 , #c=>Classe2)

2.4. Transformation des classes d'association avec clé locale

Méthode : Classe d'association N:M

Les attributs de la classe d'association,

- sont ajoutés à la relation issue de l'association N:M ;
- la clé locale de la classe d'association est concaténée aux clés étrangères composant déjà la clé primaire de la relation d'association.



Classe association (N:M)

Classe1 (#a , b)

Classe2 (#c , d)

Assoc (#a=>Classe1 , #c=>Classe2 , #e , f)

2.5. Correspondance entre UML et relationnel

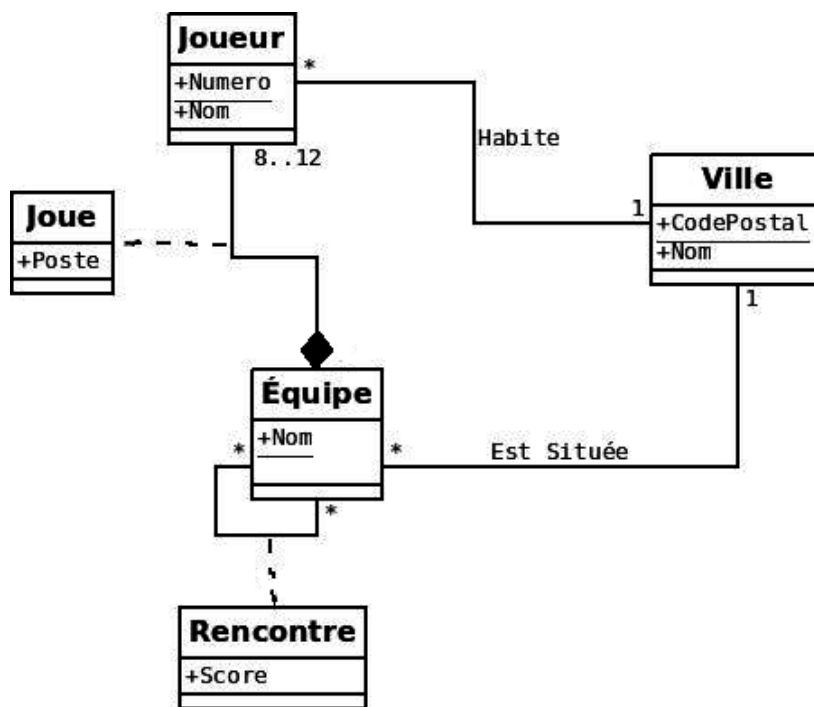
Modèle UML	Modèle relationnel
Classe instanciable	Relation
Classe abstraite	Rien
Objet	Nuplet
Attribut simple	Attribut atomique
Attribut composite	Ensemble d'attributs
Attribut multivalué	Relation et clé étrangère
Attribut dérivé	Procédure stockée ou contrainte dynamique
Méthode	Procédure stockée
Association 1:N	Clé étrangère
Association N:M	Relation et clé étrangère
Association de degré 3 ou supérieur	Relation et clé étrangère
Classe d'association	Attributs
Composition	Clé

Passage UML vers Relationnel

2.6. Exercice

[solution n°2 p.53]

Quel(s) schéma(s) relationnel(s) corresponde(nt) au modèle UML suivant ?



Volley ball

Joueur(#Numero, Nom, Ville=>Ville)

Equipe(#Nom, Joueur=>Joueur, Poste, Rencontre=>Equipe, Score, Ville=>Ville) Ville(#CodePostal, Nom)

Joueur(#Numero, #Equipe=>Equipe, Nom, Poste, Equipe(#Nom, Ville=>Ville)

Ville(#CodePostal, Nom) Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Score)

Joueur(#Numero, Nom, Equipe(#Nom, Joueur=>Joueur, Poste, Ville=>Ville)

Ville(#CodePostal, Nom) Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Score)

Joueur(#Numero, Equipe(#Nom), Ville(#CodePostal, Nom)

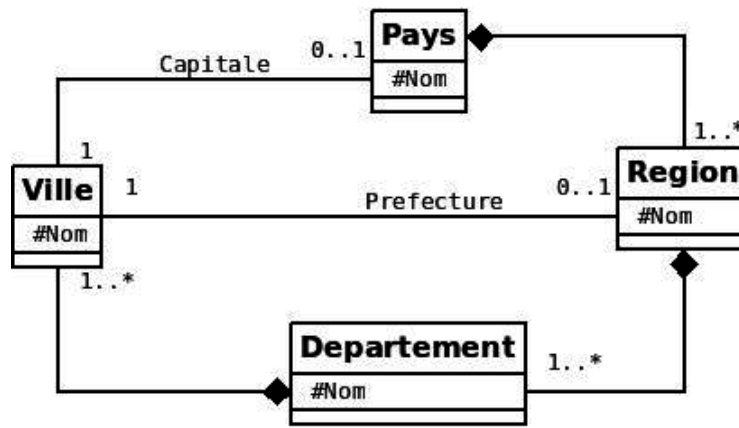
Habite(#Numero=>Joueur, #CodePostal=>Ville) Joueur(#Numero=>Joueur, #Nom=>Equipe, Poste)

EstSituée(#Nom=>Equipe, #CodePostal=>Ville) Rencontre(#NomE1=>Equipe, #NomE2=>Equipe, Score)

2.7. Exercice

[solution n°3 p.54]

Soit le schéma UML suivant :



Villes

Quelles sont les modèles relationnels qui correspondent à ce modèle conceptuel ?

Pays (#Nom, Region(#Nom, Prefecture=>Ville, Capitale=>Ville) Pays=>Pays)

Departement(#Nom, Region=>Region) Ville(#Nom, Departement=>Departement)

Pays(#Nom, Capitale=>Ville) Pays=>Pays) Region(#Nom, Prefecture=>Ville,

Departement(#Nom, Region=>Region, Pays=>Pays)

Ville(#Nom, Departement=>Departement, Region=>Region, Pays=>Pays)

Pays (#Nom) Region(#Nom, Departement(#Nom, Pays=>Pays) Region=>Region)

Ville(#Nom, Departement=>Departement, Capitale=>Pays, Prefecture=>Region)

Pays (#Nom, Region(#Nom, #Pays=>Pays, Capitale=>Ville) Prefecture=>Ville)

Departement(#Nom, #Region=>Region) Ville (#Nom, #Departement=>Departement)

Pays(#Nom, CapitaleVille=>Ville, CapitaleDepartement=>Ville, CapitaleRegion=>Ville, CapitalePays=>Ville)

Region(#Nom, #Pays=>Pays, PrefectureVille=>Ville, PrefectureDepartement=>Ville, PrefectureRegion=>Ville, PrefecturePays=>Ville)

Departement(#Nom, #Region=>Region, #Pays=>Region)

Ville(#Nom, #Departement=>Departement, #Region=>Departement, #Pays=>Departement)

3. Modélisation avancée des associations 1:1 en relationnel

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel.

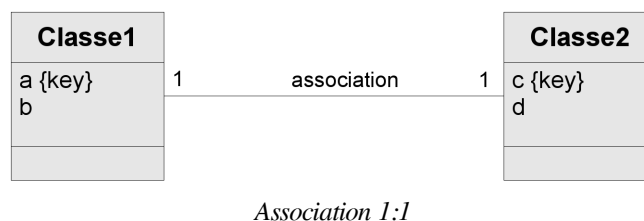
Connaître les choix possibles pour le cas de l'association 1:1 et savoir faire le meilleur choix.

Il existe des formulations conceptuelles en UML qui sont plus délicates à traduire au niveau logique en relationnel, comme l'association 1:1. Ces cas requièrent un choix éclairé de la part du concepteur.

3.1. Transformation des associations 1:1 (approche générale)

Il existe deux solutions pour transformer une association 1:1 :

- Avec deux relations : on traite l'association 1:1 comme une association 1:N, puis l'on ajoute une contrainte UNIQUE sur la clé étrangère pour limiter la cardinalité maximale à 1 ;
- Avec une seule relation : on fusionne les deux classes en une seule relation.



✂ Méthode : Avec deux relations (clé étrangère)

- Une des deux relations est choisie pour porter la clé étrangère ;
- on ajoute les contraintes : UNIQUE ou KEY (clé candidate) sur la clé étrangère ; et si nécessaire une contrainte imposant l'instanciation simultanée des deux relations.

Classe1(#a,b,c=>Classe2) avec c UNIQUE ou KEY

Classe2(#c,d)

ou

Classe1(#a,b)

Classe2(#c,d,a=>Classe1) avec a UNIQUE ou KEY

✂ Méthode : Avec une relation (fusion)

- On crée une seule relation contenant l'ensemble des attributs des deux classes ;

- on choisit une clé parmi les clés candidates.

Classe12(#a,b,c,d) avec c UNIQUE ou KEY

ou

Classe21(#c,d,a,b) avec a UNIQUE ou KEY

Remarque : Fusion des relations dans le cas de la traduction de l'association 1:1

Ce choix entre les deux méthodes sera conduit par une appréciation du rapport entre :

- La complexité introduite par le fait d'avoir deux relations là ou une suffit
- La pertinence de la séparation des deux relations d'un point de vue sémantique
- Les pertes de performance dues à l'éclatement des relations
- Les pertes de performance dues au fait d'avoir une grande relation
- Les questions de sécurité et de sûreté factorisées ou non au niveau des deux relations
- ...

Complément

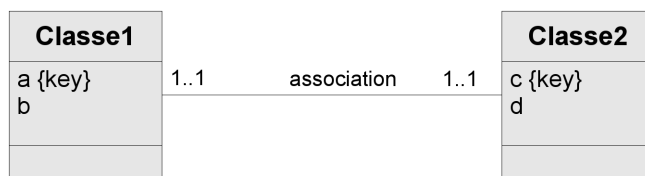
Dans le cas d'une association 1..1:1..1 il faudra ajouter une contrainte complémentaire.

Transformation des associations 1..1:1..1 (cf. p.19)

3.2. Transformation des associations 1..1:1..1

Méthode : Association 1..1:1..1

- Le plus souvent c'est méthode par *fusion* des relations qui est la plus adaptée à ce cas.
- Lorsqu'elle ne l'est pas, et que l'on choisit deux relations il faut ajouter une contrainte dynamique qui contrôlera que les deux relations sont bien toujours instanciées ensemble. Notons que la clé étrangère peut être choisie comme clé primaire.



Association 1:1

Classe12(#a,b,c,d) avec c KEY

ou

Classe21(#c,d,a,b) avec a KEY

ou

Classe1(#a,b,c=>Classe2) avec c KEY

Classe2(#c,d)

Contrainte : PROJECTION(Classe1,c) = PROJECTION(Classe2,c)

ou

Classe1(#a,b)

Classe2(#c,d,a=>Classe1) avec a KEY

Contrainte : PROJECTION(Classe1,a) = PROJECTION(Classe2,a)

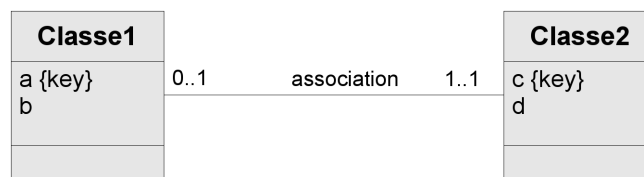
Complément : Contrainte dynamique

La contrainte dynamique exprimée ici suit le même principe que pour les association 1:N. *Contrainte de cardinalité minimale 1 dans les associations 1:N (cf. p.44)*

3.3. Transformation des associations 0..1:1..1

Méthode : Association 0..1:1..1

- Le plus souvent c'est la méthode par les deux relations qui est la plus adaptée, *on choisira toujours la relation côté 0..1 pour être référençante.*
- Il est possible d'utiliser la fusion, dans ce cas les clés côté 0..1 deviennent des attributs UNIQUE, il ne peuvent plus être clés, pouvant être NULL.



Association 0..1:1

Classe1(#a,b,c=>Classe2) avec c KEY

Classe2(#c,d)

ou

Classe12(#c,d,a,b) avec a UNIQUE

3.4. Transformation des associations 0..1:0..1

Méthode : Association 0..1:0..1

- On choisit la solution avec deux relations, d'un côté ou de l'autre ; la clé étrangère est associée à la contrainte UNIQUE, ce n'est pas une clé car elle peut être nulle.
- Il n'est pas souhaitable de choisir la fusion, l'une des deux relations pouvant être nulle, on ne pourrait plus trouver de clé naturelle.



Association 0..1:0..1

Classe1(#a,b,c=>Classe2) avec c UNIQUE

Classe2(#c,d)

ou

```
Classe1(#a,b)
```

```
Classe2(#c,d,a=>Classe1) avec a UNIQUE
```

3.5. Exemple de choix pour une relation 1:1

Exemple

- Soit deux entités, "homme" et "femme" et une association "mariage" de cardinalité 1..1:1..1 entre ces deux entités (hommes et femmes sont donc obligatoirement mariés).
- Les entités "homme" et "femme" sont identifiées par un attribut "nom" (dans ce modèle chaque personne a un nom unique, on pourrait remplacer le nom par un identifiant comme le numéro de sécurité social ou par une clé artificielle pour être plus proche de la réalité).

Bien que de type 1..1:1..1, le choix de la fusion n'est pas très opportun car il s'agit bien d'objets distincts que l'on veut modéliser.

- On choisira donc plutôt la représentation avec deux relations.
- La clé étrangère pourra être du côté "homme" ou "femme", même si la pratique dominante nous incite à la mettre du côté femme.
- On pourra également décider de prendre la clé étrangère comme clé primaire.

```
1 homme (#nom)
2 femme (#mariage=>homme, nom) avec nom KEY
3 Contrainte : PROJ(homme,nom) = PROJ(femme,mariage)
```

Exemple

Si l'association avait été de cardinalité 0..1:0..1 (certains hommes et femmes ne sont pas mariés), un choix similaire se serait imposé, avec l'impossibilité de choisir la clé étrangère comme clé primaire, celle-ci pouvant être nulle et n'étant donc plus candidate.

```
1 homme (#nom)
2 femme (#nom, mariage=>homme) avec mariage UNIQUE
```

4. Autres éléments utiles en UML : packages et stéréotypes

Objectifs

Maîtriser le diagramme de classe UML dans le cas de la conception de BD.

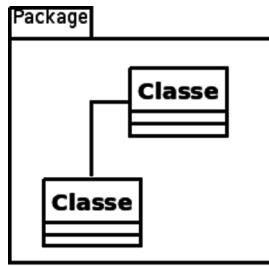
4.1. Paquetages

Définition : Package

Les paquetages (plus communément appelés *package*) sont des éléments servant à organiser un modèle.

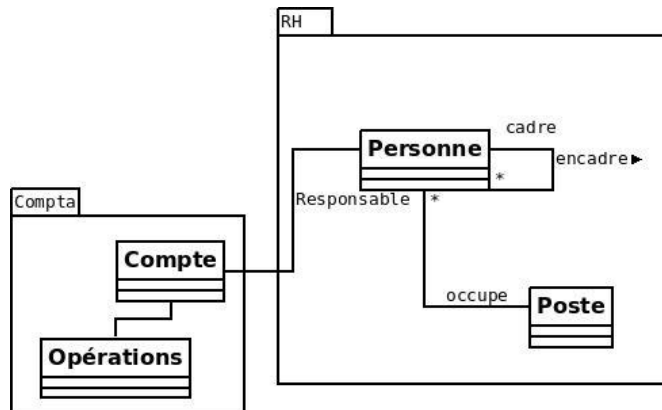
Ils sont particulièrement utiles dès que le modèle comporte de nombreuses classes et que celles-ci peuvent être triées selon plusieurs aspects structurants.

Syntaxe



Notation des paquetages en UML

Exemple

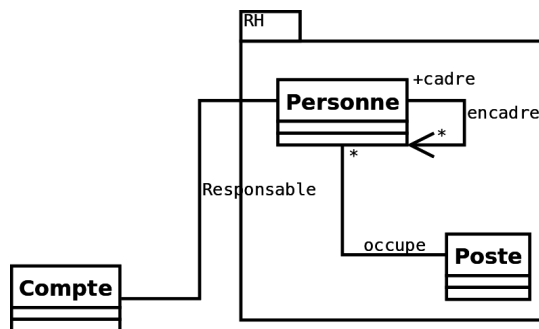


Exemple d'utilisation des paquetages

Méthode

On représente chaque classe au sein d'un *package*. Il est alors possible de faire une présentation globale du modèle (tous les *packages*), partielle (une partie des *packages*) ou centrée sur un seul *package*.

Pour une représentation partielle ou centrée sur un *package*, on représente les *packages* concernés avec leurs classes propres, ainsi que toutes les classes liées des autres *packages* (et seulement celles-ci).



Présentation partielle du modèle centrée sur un package

4.2. Stéréotype

Définition : Stéréotype UML

Un stéréotype UML est une syntaxe permettant d'ajouter de la sémantique à la modélisation des classes. Il permet de définir des *types de classe*, afin de regrouper conceptuellement un ensemble de classes (à l'instar d'une classe qui permet de regrouper conceptuellement un ensemble d'objets).

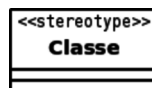
C'est une mécanique de méta-modélisation : elle permet d'étendre le méta-modèle UML, c'est à dire le modèle conceptuel du modèle conceptuel.

Définition : Méta-modèle

Un méta-modèle est le modèle d'un modèle. Par exemple le méta-modèle UML comprend les concepts de classe, attribut, association, cardinalité, composition, agrégation, contraintes, annotations, ... On mobilise ces concepts (on les instancie) pour exprimer un modèle particulier suivant le formalisme UML.

Les stéréotypes permettent donc d'ajouter au méta-modèle UML standard, celui que tout le monde utilise, des concepts locaux pour enrichir le langage de modélisation que l'on utilise pour réaliser des modèles.

Syntaxe



Notation d'un stéréotype en UML

Conseil : Stéréotypes spécifiques et stéréotypes standard

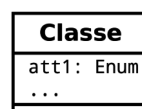
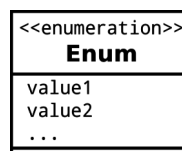
Un stéréotype spécifique enrichit le méta-modèle UML, mais selon une sémantique qui est propre à celui qui l'a posé, non standard donc. La conséquence est que pour un tiers, l'interprétation du stéréotype n'est plus normalisée, et sera potentiellement plus facilement erronée. Il convient donc de ne pas abuser de cette mécanique.

Deux ou trois stéréotypes spécifiques, correctement définis, sont faciles à transmettre, plusieurs dizaines représenteraient un nouveau langage complet à apprendre pour le lecteur du modèle.

Il existe des stéréotypes fournis en standard par UML, ou communément utilisés par les modélisateurs. L'avantage est qu'il seront compris plus largement, au même titre que le reste du méta-modèle (ils ont une valeur de standard).

4.3. Énumération : stéréotype <<enumeration>>

Syntaxe



Stéréotype UML permettant d'exprimer une énumération

Exemple

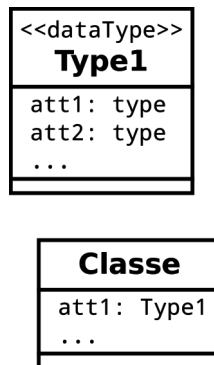


Exemple de modélisation UML d'énumération

4.4. Type utilisateurs : stéréotype <<dataType>>

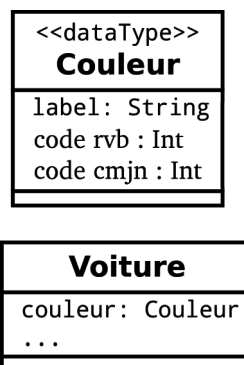
Les types utilisateurs permettent de définir des types complexes propres en extension des types primaires (entier, chaîne, date...).

Syntaxe



Stéréotype dataType

Exemple



Stéréotype dataType (exemple)

Méthode : Attributs composés

Cette modélisation est équivalente à la modélisation des attributs composés directement dans la classe principale. C'est une représentation plus standard en UML.



5. Synthèse sur la modalisation UML et relationnelle

5.1. Quelques éléments de stylistique UML

Attention

- Toutes les associations doivent être nommées (sauf composition, héritage, agrégation).
- Ne pas utiliser de nom générique pour les Classes comme "Entité", "Classe", "Objet", "Truc"...
- Éviter les noms génériques pour les associations (comme "est associé à")
- Attention au sens des compositions et agrégation, le losange est côté ensemble, et n'oubliez pas les cardinalités, notamment côté parties.

Conseil

- N'utilisez pas le souligné ni les # en UML pour identifier les clés, préférez la contrainte {key}
- Préférez l'héritage aux booléens de typage en UML
- Les attributs dérivés sont réservés aux dérivations simples (des attributs de la même classe), si c'est plus complexe, préférez des méthodes (et dans le doute, précisez les modes de calcul sur le schéma ou dans une note à part)
- Donnez des exemples de contenu lorsque ce n'est pas évident (lorsque le couple nom d'attribut et type ne permet pas de façon évidente de comprendre de quoi l'on parle)
- Inutile de déclarer le type booléen en UML, utilisez-le directement comme un type de données connu

Complément

- Si tous vos héritages sont exclusifs, notez-le à part pour alléger votre schéma (et éviter l'abondance de XOR)

5.2. Attention aux clés artificielles

Fondamental

- en UML : on ne pose jamais de clés artificielles
- en relationnel : on pose rarement des clé artificielles sauf dans le cas de clés étrangères vraiment trop compliquées
- en SQL : on peut poser des clés artificielles si on a une bonne raison (ce n'est donc pas systématique, et c'est à justifier)

Clés artificielles et niveau conceptuel

On n'ajoutera pas de clé artificielle en UML au moment de la modélisation conceptuelle des données.

1. *Formellement* en UML la notion de clé n'existe pas (contrairement à l'E-A*), elle est ajoutée par les pratiquants des BD*.
2. *Logiquement* on a pas besoin de cette notion en UML, les clés artificielles servent en relationnel et dans certains cas uniquement.
3. *Pratiquement* cela peut conduire à des situations absurdes (comme enlever au niveau logique des clés artificielles ajoutées au niveau conceptuel) :

- si on fait du non-relationnel on ne doit pas ajouter de clés artificielles dans certains cas (l'imbrication typiquement) ;
 - cela arrive même en relationnel, avec la transformation de l'héritage (dans certains cas toujours).
4. *Méthodologiquement* il faut se concentrer à chaque phase sur ce qui est important, donc au moment du MCD on traduit les besoins, on repère les contraintes explicites (clé, unicité, non nullité...) sans se préoccuper de ce qui sera rendu nécessaire par la suite par la modélisation relationnelle (les clés étrangères par exemple) ou l'implémentation (l'optimisation par exemple). À chaque jour suffit sa peine !
5. *Pédagogiquement*, enfin, les "débutants" ont tendance (à cause des environnements de conception graphique, comme phpMyAdmin notamment) à systématiser les clés artificielles en SQL (on pourrait en discuter), mais également à ne pas faire le travail de recherche des clés naturelles (au niveau relationnel notamment, ce qui est une faute de modélisation). Donc au plus tard on fait intervenir les clés artificielles, au plus on a une chance de penser aux clés naturelles.

Clés artificielles et optimisation : est-il toujours plus performant d'utiliser des clés artificielles ?

Soit le modèle relationnel suivant :

```
1 Etu (#id, numEtu...) avec numEtu clé
2 UV (#id, codeUv...) avec codeUv clé
3 Inscriptions (#id, uv=>UV, etu=>Etu) avec (uv,etu) clé
```

La question qui permet d'afficher la liste des étudiants (numEtu) avec leurs UVs (codeUv) nécessite une jointure à cause des clés artificielles. Un modèle sans ces clés artificielles aurait été plus performant pour répondre à cette question, puisque toutes les informations se trouvent dans la relation Inscriptions.

```
1 Etu (#numEtu...)
2 UV (#codeUv...)
3 Inscriptions (#uv=>UV, #etu=>Etu)
```

Conseil

- Ne soyez pas systématique.
- Il est en effet fréquent dans un projet réel d'adopter des clés artificielles presque systématiquement, vous pourrez le faire en connaissance de cause quand vous aurez bien compris pourquoi c'est intéressant et quand ça ne l'est pas.
- Les clés artificielles sont intéressantes pour autre chose que les performances (l'évolutivité par exemple, choisissez-les quand vous savez pourquoi).
- Les clés artificielles ne dispensent pas de rechercher les clés naturelles !
- Les clés artificielles ne sont pas la seule façon d'optimiser une base de données (indexation, dénormalisation...)

5.3. Bibliographie commentée sur la modélisation UML

Complément : Outils de modélisation UML

Il existe de nombreux outils de modélisation UML. On pourra citer :

- Dia* : logiciel Open Source et multi-plateformes facile d'usage (qui marche néanmoins mieux sur Linux que sur Windows).
- Objectteering* (version gratuite).

À voir également en Open Source : *ArgoUML* ou *EclipseUML*. (non testé par l'auteur).

Complément : Modélisation UML

UML2 en
action *

Pour un aperçu plus détaillé des possibilités d'expression du diagramme de classe UML, lire le chapitre 7 : Développement du modèle statique (pages 133 à 163).

On pourra notamment y trouver :

- L'association d'agrégation
- Les propriétés d'association
- L'expression de rôles dans les associations
- Les attributs de classe
- Les qualificatifs
- Les opérations (ou méthodes)

Le chapitre donne de plus des conseils méthodologiques pour la conception (voir en particulier la synthèse page 163).

On pourra également y trouver :

- Des principes de choix de modélisation entre attributs et classes et sur la segmentation des classes
- Des principes de sélection des attributs (redondance avec les associations, avec les classes, etc.)
- Des principes de sélection des associations
- Des principes de choix de cardinalité (notamment pour la gestion d'historisation)
- Des principes de sélection des relations de généralisation (héritage)
- Des principes d'introduction de métaclasses (types)

Complément : Référence UML en ligne

UML en
Français *

Une très bonne référence en ligne sur la modélisation UML, avec des cours, des liens vers la norme, etc.

Le contenu dépasse très largement l'usage d'UML pour la modélisation de BD (et ne fait d'ailleurs pas de référence précise à ce sous-ensemble particulier).

On pourra consulter en particulier le chapitre sur les diagrammes de classe : <http://uml.free.fr/cours/i-p14.html>


Complément : Tutoriel sur la modélisation UML.

UML en 5 étapes*

On consultera en particulier le tutoriel sur les diagrammes de classe : http://developpeur.journaldunet.com/tutoriel/cpt/010607cpt_umlintro.shtml

Complément : Conseils

Cinq petits conseils pour un schéma UML
efficace *

 **Complément : Pratique**

UML2 par la pratique* (chapitre 3)

Des explications, exemples et études de cas.

5.4. Synthèse : Les diagrammes de modélisation conceptuelle

Un modèle conceptuel peut être représenté sous forme de diagramme E-A ou sous forme de diagramme de classe UML.

- Classe ou Entité
 - Attribut ou Propriété
 - Typé
 - Multi-valué
 - Composé
 - Dérivé
 - Méthode
 - Paramètres
 - Valeur de retour
- Association
 - Association
 - Verbe
 - Cardinalité
 - Héritage
 - Héritage d'attributs
 - Héritage de méthodes
 - Composition (ou entité faible)
 - Cardinalité

Exercices


 II

1. Exercice : Lab III

[20 min]

Description du problème

Un laboratoire souhaite gérer les médicaments qu'il conçoit.

- Un médicament est décrit par un nom, qui permet de l'identifier. En effet il n'existe pas deux médicaments avec le même nom. Un médicament comporte une description courte en français, ainsi qu'une description longue en latin. On gère aussi le conditionnement du médicament, c'est à dire le nombre de pilules par boîte (qui est un nombre entier).
- À chaque médicament on associe une liste dédiée de contre-indications, généralement plusieurs, parfois aucune. Les contre-indications sont triées par ordre d'importance. L'ordre est total et strict pour un médicament, donc chaque contre-indication possède une importance et il n'existe pas deux contre-indications associées au même médicament avec la même importance.
- Tout médicament possède au moins un composant, souvent plusieurs. Un composant est identifié par un code unique et possède un intitulé. Tout composant peut intervenir dans la fabrication de plusieurs médicaments. Il existe des composants qui ne sont pas utilisés pour fabriquer des médicaments et que l'on veut quand même gérer.

Données de test

Afin de matérialiser notre base de données, nous obtenons les descriptions suivantes :

- Le *Chourix* a pour description courte « *Médicament contre la chute des choux* » et pour description longue « *Vivamus fermentum semper porta. Nunc diam velit, adipiscing ut tristique vitae, sagittis vel odio. Maecenas convallis ullamcorper ultricies. Curabitur ornare.* ». Il est conditionné en boîte de 13.

Ses contre-indications sont :

1. Le *Chourix* ne doit jamais être pris après minuit.
2. Le *Chourix* ne doit jamais être mis au contact avec de l'eau.

Ses composants sont le *HG79* et le *SN50*.

- Le *Tropas* a pour description courte « *Médicament contre les dysfonctionnements intellectuels* » et pour description longue « *Suspendisse lectus leo, consecetur in tempor sit amet, placerat quis neque. Etiam luctus porttitor lorem, sed suscipit est rutrum non.* ». Il est conditionné en boîte de 42.

Ses contre-indications sont :

1. Le *Tropas* doit être gardé à l'abri de la lumière du soleil

Son unique composant est le *HG79*.

- Les composants existants sont :
 - *HG79* : "Vif-argent allégé"

- *HG81* : "Vif-argent alourdi"
- *SN50* : "Pur étain"

Question 1

[solution n°4 p.56]

Effectuez le modèle conceptuel en UML de ce problème.

Indices :

On note dans l'énoncé que les contre-indications sont *dédiée* aux médicaments, et par ailleurs on note dans les données exemples que les contre-indications sont énoncées spécifiquement par rapport aux médicaments.

Ce n'est pas parce que les composants s'appellent ainsi dans l'énoncé que l'on est en présence d'une composition.

Question 2

[solution n°5 p.57]

En mobilisant les règles adéquates, proposer un modèle logique de données correspondant en relationnel. Le repérage des domaines et des clés est obligatoire.

Question 3

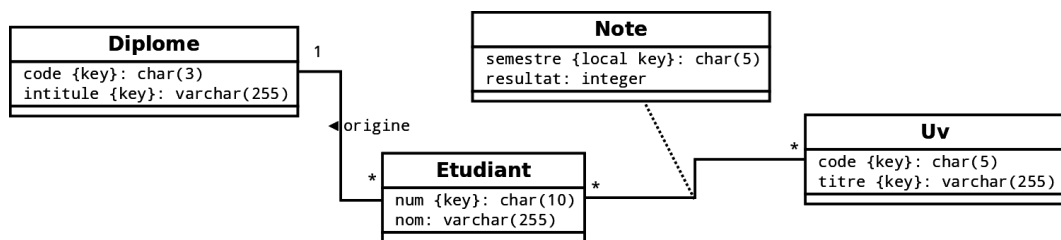
[solution n°6 p.57]

Dessiner des tableaux remplis avec les données fournies en exemple, afin de montrer que le modèle fonctionne selon le besoin exprimé initialement. On pourra mettre le premier mot seulement des descriptions pour gagner du temps.

2. Exercice : Étudiants et UVs (introduction)

[20 min]

On dispose du schéma UML ci-après qui décrit des étudiants, des UV, les notes obtenues par les étudiants à ces UV, et les diplômes d'origine de ces étudiants.



Étudiants et UVs

- {key} désigne des clés candidates, ici toutes les clés ne sont composées que d'un seul attribut
- {local key} désigne une clé locale
- un semestre est de la forme 'PYYYY' ou 'AYYYY' (où YYYY désigné une année sur 4 chiffre) ; exemple : 'A2013', 'P2014...

Rappel : *Notion de clé locale dans classes d'association (cf. p.49)*

Question

[solution n°7 p.57]

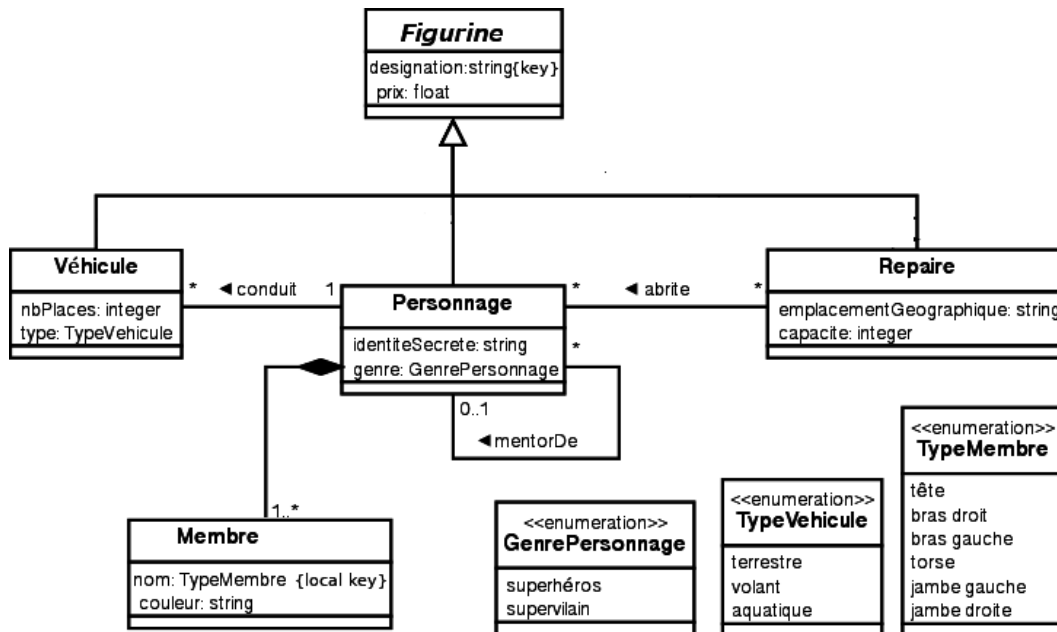
Traduire le schéma en modèle logique relationnel. (MLD1).

On choisira obligatoirement les clés primaires parmi celles nécessitant le plus petit nombre de bits possible pour leur codage.

3. Exercice : Super-héros relationnels I

[20 min]

La gamme de super-héros GARVEL veut réaliser la base de données de leurs figurines articulées. La société a fait réaliser un modèle UML qui doit servir de point de départ à la mise en œuvre.



Modèle UML Figurines GARVEL

Question

[solution n°8 p.58]

Transformer le modèle UML en modèle relationnel (justifier les passages non triviaux, en particulier la relation d'héritage).

4. Exercice : Objets Numériques Libres

[30 min]

L'association ONL (Objets Numériques Libres) est une association de promotion des logiciels libres. Elle souhaite exposer sur un site Internet une liste de logiciels libres. Ce site sera adossé à une base de données relationnelle ou relationnel-objet. La première étape de sa démarche est de réaliser un modèle conceptuel représentant ce qu'elle souhaite faire.

- La base de données permet de gérer des applications.
Les applications sont identifiées par leur nom (LibreOffice, Gimp...) et leur version (1.0, 2.1), et comportent une description courte et une URL. Tous les attributs sont obligatoires. Chaque application a une URL unique.
- La base de données permet de gérer des librairies.

Les bibliothèques sont des logiciels mais pas des applications. Elles ont les mêmes attributs que les applications (nom, version, description courte, URL), mais les URL ne sont pas nécessairement uniques. Les applications peuvent dépendre de bibliothèques ou d'autres applications, et les bibliothèques peuvent dépendre d'autres bibliothèques (mais pas d'une application).

- La base de données permet de gérer des composants.

Les composants sont intégrés à une application ou bibliothèque. Les composants ont un code interne à l'application ou la bibliothèque qu'il servent, une version et une description courte, et une URL. Le code et le numéro de version permettent d'identifier localement le composant au sein de la bibliothèque ou de l'application, la description courte et l'URL sont optionnelles.

- La base de données permet de gérer des licences.

Les applications, bibliothèques et composants sont attachés à une ou plusieurs licences identifiées par leur nom (GPL, MPL...), leur version et leur langue, et comportant le texte intégral de la licence. Les versions des logiciels et licences sont de type "numéro de licence majeur point numéro de licence mineur", comme "1.0" ou "2.2".

- La base de données permet de gérer des catégories.

Chaque logiciel est rangé dans une catégorie principale, et plusieurs catégories secondaires. Exemple de catégories : bureautique, dessin, multimédia, physique...

Exemple (factice) de données

- Applications :

- Scenari 4.1, une chaîne éditoriale XML, <http://scenari.org>, dépend de Libreoffice 4.3 et de ImageMagick 6.8
- Libreoffice 4.3, une suite bureautique WYSIWYG, <http://libreoffice.org>

- Bibliothèque :

- ImageMagick 6.8, permet de créer, de convertir, de modifier et d'afficher des images, <http://imagemagick.org>

- Composant :

- `impng 0.2` est un composant de ImageMagick 6.8, permet de compresser une image au format PNG.

- Toutes ces applications, bibliothèques et composants sont disponibles sous une licence LGPL 3.0 et GPL 3.0 françaises.

- Toutes ces applications et bibliothèques sont rangées dans la catégorie principale "document". Scenari est rangé dans la catégorie secondaire "Édition WYSIWYM", Libreoffice dans la catégorie secondaire "Bureautique", ImageMagick dans la catégorie secondaire "Multimédia".

Question

[solution n°9 p.58]

Réaliser un MCD en UML.

Devoirs

III

1. Exercice : Arbre de scène 3D

[30 minutes]

On souhaite pouvoir gérer les différents éléments composant des scènes 3D dans une base de données. Une scène contient des objets qui eux-mêmes peuvent appartenir à plusieurs scènes (au moins une), mais il ne peut y avoir plusieurs fois le même objet dans une même scène. Dans chaque scène, les objets peuvent être visibles ou invisibles. Les scènes et les objets sont identifiés de manière unique par un nom. Une scène peut être active ou inactive. Un objet a comme propriété une position dans l'espace, représentée par un vecteur de réels à trois composantes (x, y, z). Les objets sont organisés de manière hiérarchique : un objet peut être parent de plusieurs objets et chaque objet peut avoir au plus un parent. Des scripts peuvent être associés à un objet ou à une scène (à l'un ou à l'autre mais pas aux deux). Un script est identifié de manière unique par un nom et possède également un attribut permettant de connaître son état (actif ou inactif). Un personnage est un objet particulier qui possède des animations. Une animation est relative à un personnage et est identifiée de manière locale par un nom. À un objet est associé un maillage et celui-ci peut être utilisé par plusieurs objets. Un maillage est identifié de manière unique par un nom et est composé de plusieurs éléments. Chaque élément est relatif à un maillage et est identifié de manière locale par un numéro. Il existe exactement trois types d'élément : des nœuds, des arrêtes et des faces. Une face est délimitée par trois arrêtes et chaque arrête est délimitée par deux nœuds. Plusieurs arrêtes peuvent partager un même nœud et plusieurs faces peuvent partager une même arrête. Afin d'évaluer la complexité d'une scène, on souhaite pouvoir calculer le nombre de faces affichées pour une scène donnée (c'est-à-dire la somme du nombre de faces du maillage associé aux objets visibles de la scène). Un maillage possède plusieurs matériaux identifiés de manière unique par un nom. Un matériau peut être associé à plusieurs maillages. Un matériau est caractérisé par une couleur définie par un vecteur d'entiers à quatre composantes : rouge, vert, bleu, alpha. Un matériau peut posséder une texture et celle-ci peut être utilisée par plusieurs matériaux. Une texture est identifiée de manière unique par un nom et possède comme attribut une image.

Question 1

Proposez une clarification de ce problème. On pourra reclasser les informations par grande catégorie : scène, objets, scripts...

Question 2

Établissez un modèle conceptuel en UML de ce problème.

Complément : Exercices de modélisation supplémentaire

IV

1. Exercice : Appartements à louer

[30 min]

Une agence d'immobilier (ventes et locations) veut gérer son parc d'appartements dans une base de données. Chaque appartement possède plusieurs pièces, qui peuvent être des pièces d'habitation, des pièces de rangement, ou des commodités.

Chaque pièce a une superficie, un nombre de prises électriques et des meubles. Pour ces derniers, on veut enregistrer le nom du modèle et le type (chaise, lit, four, etc.). Rien n'empêche qu'une pièce comporte plusieurs meubles identiques (par exemple plusieurs chaises 'PIN IQUEA'). Les pièces d'habitation ont une ou plusieurs fenêtres, les pièces de rangement ont une ou plusieurs étagères et une surface de rangement utile totale. On veut aussi savoir dans quelles pièces de rangement se situent les tableaux électriques de l'appartement. Finalement, les pièces commodités peuvent être de trois types : cuisine, toilette, ou salle de bain.

L'agence veut aussi enregistrer les portes entre les différentes pièces. Une porte peut être à simple battant, double battant, coulissante, ou une simple ouverture.

Chaque appartement a une adresse, une superficie totale (qui est la somme des superficies des pièces) et un type, qui est la lettre 'T' suivie du nombre de pièces d'habitation de l'appartement. Pour un appartement donné, chaque pièce est identifiée par un numéro (commençant au numéro 1).

L'agence veut également enregistrer le quartier où se trouve l'appartement, auquel est lié un prix par mètre carré. Finalement, elle veut garder un historique de l'histoire de l'appartement : les périodes pour lesquelles il est libre, occupé, en travaux, ou vendu.

Question

[solution n°10 p.59]

Réalisez le diagramme UML répondant aux besoins de cette agence.

2. Exercice : Objectifs

[40 min]

Vous prenez la présidence de l'association "Objectifs", dont l'objet est d'aider ses étudiants membres à mener des projets dans le cadre de leurs études.

Vous constatez en arrivant dans l'association que personne ne sait exactement quels sont les projets en cours ni qui fait quoi dans les projets. Vous ne parvenez pas non plus à obtenir une liste exacte des partenaires sur lesquels l'association peut compter. La seule chose sur laquelle vous parvenez à mettre la main est un fichier tableur vaguement à jour.

Vous décidez qu'il est plus que temps de mettre en place une base de données afin d'assurer la bonne gestion des informations les plus importantes pour l'association. Vous vous attachez pour cela à réaliser une analyse des besoins.

Analysez les documents ci-après.

Etat des projets

Numéro	Projet	Chef de projet	Date	Remarques
1	La nuit du Pico	Paul Eau	25/12/2003	Année dernière, bières offertes par 1664
2	L'escalade de l'Evrest	Pierre Elelou	en 2005	Financement à trouver. Voir si Décathlon pourrait sponsoriser.
3	Tournoi de Volley-Ball	Marc Noflair	Intersemestre	Prix offerts par Yves Rocher. Appui logistique BDS.
3.1	Recrutement des équipes	Barbara Stressée		
3.2	Arbitrage	Barbara (elle est sportive de haut niveau et connaît tout au sport)		

NB : Hélène Deuxtrois est d'accord pour aider sur tous les projets, pour la compta.

État des projets de l'association Objectifs

Recueil de besoins

À partir de l'étude du fonctionnement actuel de l'association, vous relevez les choses importantes à gérer suivantes :

- Les membres de l'association, généralement identifiés par leur prénom.
- Les projets gérés par l'association, avec des dates de début et de fin précises et un chef de projet.
- Le détail des tâches de chaque projet, avec leurs dates de début et de fin aussi, et la liste de tous les membres qui y participent.
- La liste des partenaires des projets, avec leur rôle précis pour chacun des projets auxquels ils participent

Afin de mieux gérer les attributions des projets, il serait également intéressant de créer une liste des spécialités correspondant à ce qui se fait dans l'association, et d'affecter une spécialité à chaque membre ainsi qu'une à chaque projet, pour voir si ce sont bien les membres les mieux adaptés qui travaillent sur les projets.

Recueil de données

Les discussions avec les anciens membres de l'association vous apprennent les choses suivantes :

- L'association a terminé de gérer les trois projets, "Comédie Musicale" gérée par Nathalie sur les trois premiers mois du semestre d'automne 1001, "Science en fête" gérée par Pierre sur tout le semestre de printemps 1003. et "Nuit du piccolo" gérée par Julien en novembre 1001.
- Les spécialités recensées pour le moment sont : Ville, Université, Sport, Entreprise, Culture, International
- Il faut au moins dix membres à l'association (sinon elle ferme), recruter ceux qui manquent.
- Aidez l'association à diviser ses projets en tâches.

- Les partenaires suivants aident ou ont aidé l'association : la mairie pour la comédie musicale et la science en fête, qui apporte un soutien financier ; le ministère de la culture qui apporte son soutien logistique à la science en fête ; l'association des commerçants de la ville qui apporte son soutien publicitaire à la comédie musicale ; 1664 qui offre ses bières à moitié prix pour la nuit du Picolo.

Question 1

[solution n°11 p.59]

Afin de préparer une analyse des besoins sommaire, listez les données que vous devrez gérer à l'aide des documents et recueils de discussion à votre disposition. Faites des hypothèses lorsque les données sont incomplètes ou incohérentes.

Question 2

[solution n°12 p.60]

Réaliser le MCD en UML et en E-A à partir des données préparées dans la question précédente.

Question 3

[solution n°13 p.61]

Réaliser le MLD en relationnel en traduisant votre MCD.

Contenus annexes

> Attributs

Définition : Attribut

Un attribut est une information élémentaire qui caractérise une classe et dont la valeur dépend de l'objet instancié.

Un attribut est typé : Le domaine des valeurs que peut prendre l'attribut est fixé a priori.

- *Un attribut peut être multivalué* : Il peut prendre plusieurs valeurs distinctes dans son domaine.
- *Un attribut peut être dérivé* : Sa valeur alors est une fonction sur d'autres attributs de la classe
- *Un attribut peut être composé* (ou composite) : Il joue alors le rôle d'un groupe d'attributs (par exemple une adresse peut être un attribut composé des attributs numéro, type de voie, nom de la voie). Cette notion renvoie à la notion de variable de type `Record` dans les langages de programmation classiques.

Attention : On utilise peu les attributs dérivés et composés en UML

- En UML on préfère l'usage de méthodes aux attributs dérivés. On utilisera toujours des méthodes dès que la valeur de l'attribut dérivé dépend d'autres attributs extérieurs à sa classe.
- En UML on préfère l'usage de compositions aux attributs composés. On utilisera toujours des compositions pour les attributs composés et multivalués.

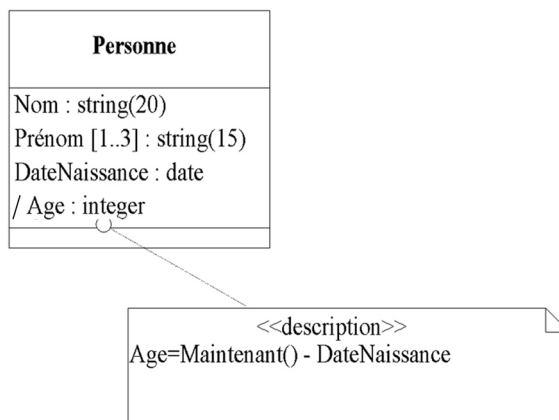
Syntaxe

```

1 attribut:type
2 attribut_multivalué[nbMinValeurs..nbMaxValeurs]:type
3 /attribut_dérivé:type
4 attribut_composé
5   - sous-attribut1:type
6   - sous-attribut2:type
7   - ...

```

Exemple : La classe Personne



Représentation d'attributs en UML

Dans cet exemple, les attributs Nom, Prénom sont de type *string*, l'un de 20 caractères et l'autre de 10, tandis que DateNaissance est de type *date* et Age de type *integer*. Prénom est un attribut multivalué, ici une personne peut avoir de 1 à 3 prénoms. Age est un attribut dérivé, il est calculé par une fonction sur DateNaissance.

Complément : Voir aussi

Méthodes (cf. p.39)

Composition (cf. p.5)

> Méthodes

🔑 Définition : Méthode

Une méthode (ou opération) est une fonction associée à une classe d'objet qui permet d'agir sur les objets de la classe ou qui permet à ces objets de renvoyer des valeurs (calculées en fonction de paramètres).

📦 Syntaxe

```
1 methode (paramètres) :type
```

🔍 Remarque : Méthodes et modélisation de BD

Pour la modélisation des bases de données, les méthodes sont surtout utilisées pour représenter des données calculées (à l'instar des attributs dérivées) ou pour mettre en exergue des fonctions importantes du système cible. Seules les méthodes les plus importantes sont représentées, l'approche est moins systématique qu'en modélisation objet par exemple.

🔍 Remarque : Méthodes, relationnel, relationnel-objet

Lors de la transformation du modèle conceptuel UML en modèle logique relationnel, les méthodes *ne seront généralement pas implémentées*. Leur repérage au niveau conceptuel sert donc surtout d'aide-mémoire pour l'implémentation au niveau applicatif.

Au contraire, un modèle logique relationnel-objet permettra l'implémentation de méthodes directement associées à des tables. Leur repérage au niveau conceptuel est donc encore plus important.

Complément

Transformation des méthodes par des vues (cf. p.40)

> Transformation des méthodes par des vues

Méthode

Lorsqu'une méthode est spécifiée sur un diagramme UML pour une classe C, si cette méthode est une fonction relationnelle (elle renvoie une unique valeur et elle peut être résolue par une requête SQL), alors on crée une vue qui reprend les attributs de la classe C et ajoute des colonnes calculées pour les méthodes.

Remarque : Attributs dérivés

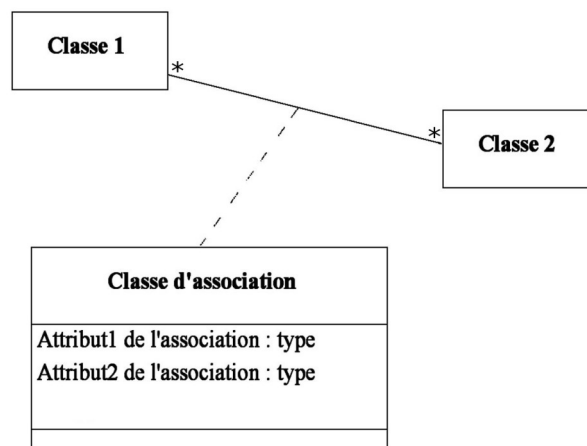
Les attributs dérivés étant apparentés à des méthodes, ils peuvent également être gérés par des vues.

> Classe d'association

Définition : Classe d'association

On utilise la notation des classes d'association lorsque l'on souhaite ajouter des propriétés à une association.

Syntaxe : Notation d'une classe d'association en UML



Notation d'une classe d'association en UML

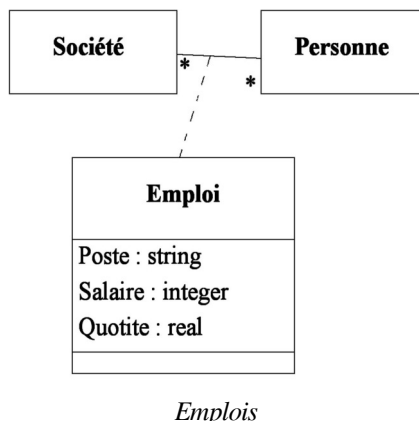
Méthode

On réserve en général les classes d'association aux associations N:M.

Il est toujours possible de réduire une classe d'association sur une association 1:N en migrant ses attributs sur la classe côté N, et c'est en général plus lisible ainsi.



Exemple : Exemple de classe d'association



✂ *Conseil*

Selon le standard UML une classe d'association est une classe et à ce titre elle peut être mobilisée dans d'autres associations ou dans des héritages. Nous déconseillons néanmoins ces notations qui ont tendance à complexifier la lecture et la transformation du diagramme.

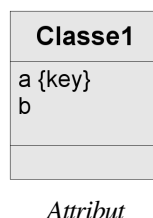
Nous conseillons donc de ne jamais associer une classe d'association.

> Transformation des attributs

✂ *Méthode : Attributs simples*

Pour chaque attribut élémentaire et monovalué d'une classe,

- on crée un attribut correspondant.

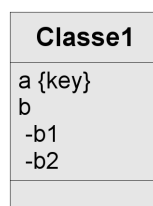


```
Classe1(#a,b)
```

✂ *Méthode : Attributs composites*

Pour chaque attribut composite comprenant N sous-attributs d'une classe,

- on crée N attributs correspondants,
- dont les noms sont la concaténation du nom de l'attribut composite avec celui du sous-attribut.



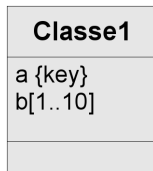
Attribut composé

Classe1 (#a, b_b1, b_b2)

 Méthode : Attributs multivalués

Pour chaque attribut multivalué b d'une classe C,


- on crée une nouvelle relation RB,
- qui comprend un attribut monovalué correspondant à b,
- plus la clé de la relation représentant C ;
- la clé de RB est la concaténation des deux attributs.



Attribut multivalué


Classe1 (#a)

RB (#b, #a=>Classe1)

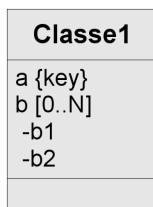
 Méthode : Attributs multivalués (méthode alternative)

Dans le cas où le nombre maximum de b est fini, et petit, on peut également adopter la transformation suivante : Classe1 (#a, b1, b2, b3, b4, b5, b6, b7, b8, b9, b10).

Si le nombre d'attributs est infini (b[1..*]) c'est impossible, s'il est trop grand ce n'est pas souhaitable.

 Méthode : Attributs composés multivalués


On combine les règles énoncées pour les attributs composés et pour les attributs multivalués.



Attribut composé multivalué

Classe1 (#a)

RB (#b_b1, #b_b2, #a=>Classe1)

 Rappel : Voir aussi

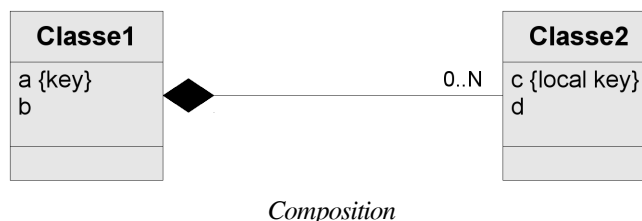
Transformation des compositions (cf. p.42)

> Transformation des compositions

Méthode

Une composition

- est transformée comme une association 1:N,
- puis on ajoute à la clé de la classe partie (dite clé locale) la clé étrangère vers la classe composite pour construire une clé primaire composée.



Classe1 (#a , b)

Classe2 (#c , #a=>Classe1 , d)

Remarque : Clé locale

Pour identifier une classe partie dans une composition, on utilise une clé locale concaténée à la clé étrangère vers la classe composite, afin d'exprimer la dépendance entre les deux classes.

Si une clé naturelle globale permet d'identifier de façon unique une partie indépendamment du tout, on préférera la conserver comme clé candidate plutôt que de la prendre pour clé primaire.

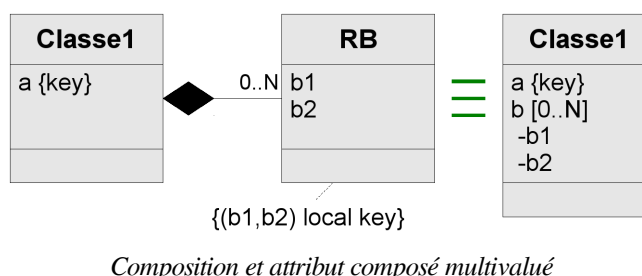
Si on la choisit comme clé primaire cela revient à avoir transformé la composition en agrégation, en redonnant une vie propre aux objets composants.

Complément : Composition et entités faibles en E-A

Une composition est transformée selon les mêmes principes qu'une entité faible en E-A.

Complément : Attributs multivalués et composés

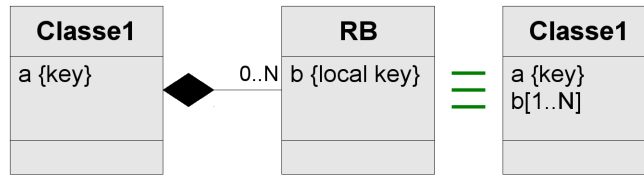
La transformation d'un attribut composé multivalué donne un résultat équivalent à la transformation d'une composition.



Classe1 (#a)

RB (#b_b1 , #b_b2 , #a=>Classe1)

La transformation d'une composition avec un seul attribut pour la classe composante donne un résultat équivalent à la transformation d'un attribut multivalué.



Composition et attribut multivalué

Classe1 (#a)

RB (#b, #a=>Classe1)

Rappel : Voir aussi

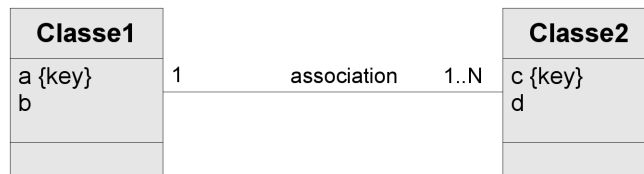
Transformation des attributs (cf. p.41)

> Contrainte de cardinalité minimale 1 dans les associations 1:N

Rappel

Transformation des associations 1:N (cf. p.45)

Méthode



Association 1:N

- Si la cardinalité est exactement 1 (1..1) côté 1, alors on ajoutera une contrainte de non nullité sur la clé étrangère,
- si la cardinalité est au moins 1 (1..N) côté N, on ajoutera une contrainte d'existence de tuples référant pour chaque tuple de la relation référencée.

R1 (#a, b)

R2 (#c, d, a=>R1)

Contraintes : a NOT NULL et PROJECTION(R1, a) = PROJECTION(R2, a)

On veut que tous les éléments de R1 soient référencés au moins une fois dans R2, donc il n'existe pas de tuple de R1 qui ne soit pas référencé par R2, a, donc PROJECTION(R1, a) = PROJECTION(R2, a).

#a	b
1	Lorem
2	Ipsum

R1

#c	b	a=>R1
a	Sed	1
b	Ut	2
c	Perspiciatis	2

R2

 Complément : Cas général : $PROJECTION(R1,a)$ $PROJECTION(R2,a)$


Si la cardinalité côté 1 est 0..1 alors il peut exister la valeur NULL dans R2.a et donc la contrainte devient : $PROJECTION(Classe1, a)$ $PROJECTION(Classe2, a)$.

#a	b
1	Lorem
2	Ipsum

R1

#c	b	a=>R1
a	Sed	1
b	Ut	2
c	Perspiciatis	2
d	Unde	NULL

R2

 Rappel : La projection élimine les doublons

Projection (cf. p.48)

> Transformation des associations 1:N

 Méthode

Pour chaque association binaire de type 1:N :

- on ajoute à la relation côté N une clé étrangère vers la relation côté 1.



Association 1:N

Classe1 (#a , b)

Classe2 (#c , d , a=>Classe1)

Complément

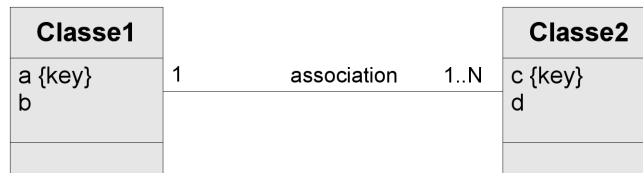
Contrainte de cardinalité minimale 1 dans les associations 1:N (cf. p.46)

> **Contrainte de cardinalité minimale 1 dans les associations 1:N**

Rappel

Transformation des associations 1:N (cf. p.45)

Méthode



Association 1:N

- Si la cardinalité est exactement 1 (1..1) côté 1, alors on ajoutera une contrainte de non nullité sur la clé étrangère,
- si la cardinalité est au moins 1 (1..N) côté N, on ajoutera une contrainte d'existence de tuples référencant pour chaque tuple de la relation référencée.

R1 (#a , b)

R2 (#c , d , a=>R1)

Contraintes : a NOT NULL et PROJECTION(R1 , a) = PROJECTION(R2 , a)

On veut que tous les éléments de R1 soient référencés au moins une fois dans R2, donc il n'existe pas de tuple de R1 qui ne soit pas référencé par R2.a, donc PROJECTION(R1 , a) = PROJECTION(R2 , a).

#a	b
1	Lorem
2	Ipsum

R1

#c	b	a=>R1
a	Sed	1
b	Ut	2
c	Perspiciatis	2

R2

 Complément : Cas général : $PROJECTION(R1,a) \text{ } PROJECTION(R2,a)$


Si la cardinalité côté 1 est 0..1 alors il peut exister la valeur NULL dans R2.a et donc la contrainte devient : $PROJECTION(Classe1,a) \text{ } PROJECTION(Classe2,a)$.

#a	b
1	Lorem
2	Ipsum

R1


#c	b	a=>R1
a	Sed	1
b	Ut	2
c	Perspiciatis	2
d	Unde	NULL

R2

 Rappel : La projection élimine les doublons

Projection (cf. p.47)

> Projection

 Définition : Projection

La projection est une opération unaire (c'est à dire portant sur une seule relation). La projection de R1 sur une partie de ses attributs {A1, A2, ...} produit une relation R2 dont le schéma est restreint aux attributs mentionnés en opérande, comportant les mêmes tuples que R1, et dont les doublons sont éliminés.

Syntaxe

$R = \text{Projection} (R1, a1, a2, \dots)$

Exemple

Soit la relation suivante : *Personne* (nom, prénom, age)

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Personne

Soit l'opération : $R = \text{Projection} (\text{Personne}, \text{nom}, \text{age})$

On obtient alors la relation R composée des tuples suivants :

nom	age
Dupont	20
Durand	30

R

Remarque : La projection élimine les doublons

Après suppression d'une partie des attributs du schéma, la relation peut comporter des doublons. Étant donné que l'on ne pourrait plus identifier ces doublons les uns par rapport aux autres, la seule solution sensée est donc de considérer que deux doublons sont équivalents, et donc de n'en garder qu'un seul dans la relation résultante.

Complément : Syntaxes alternatives

$R = (R1, a1, a2, \dots)$

$R = a1, a2, \dots (R1)$

> **Projection**

Définition : Projection

La projection est une opération unaire (c'est à dire portant sur une seule relation). La projection de R1 sur une partie de ses attributs {A1, A2, ...} produit une relation R2 dont le schéma est restreint aux attributs mentionnés en opérande, comportant les mêmes tuples que R1, et dont les doublons sont éliminés.

 **Syntaxe**

$R = \text{Projection} (R1, a1, a2, \dots)$

 **Exemple**

Soit la relation suivante : *Personne* (nom, prénom, age)

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Personne

Soit l'opération : $R = \text{Projection} (\textit{Personne}, \text{nom}, \text{age})$


On obtient alors la relation R composée des tuples suivants :

nom	age
Dupont	20
Durand	30

R

 **Remarque : La projection élimine les doublons**


Après suppression d'une partie des attributs du schéma, la relation peut comporter des doublons. Étant donné que l'on ne pourrait plus identifier ces doublons les uns par rapport aux autres, la seule solution sensée est donc de considérer que deux doublons sont équivalents, et donc de n'en garder qu'un seul dans la relation résultante.

 **Complément : Syntaxes alternatives**

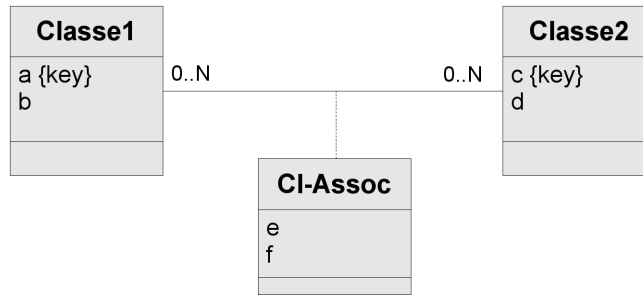
$R = (R1, a1, a2, \dots)$

$R = a1,a2,\dots (R1)$

> Transformation des classes d'association

 **Méthode : Classe d'association N:M**

Les attributs de la classe d'association sont ajoutés à la relation issue de l'association N:M.



Classe association (N:M)

```
Classe1(#a,b)
```

```
Classe2(#c,d)
```

```
Assoc(#a=>Classe1,#c=>Classe2,e,f)
```



Complément : Classe d'association 1:N

Les attributs de la classe d'association sont ajoutés à la relation issue de la classe côté N.



Complément : Classe d'association 1:1

Les attributs de la classe d'association sont ajoutés à la relation qui a été choisie pour recevoir la clé étrangère. Si les deux classes ont été fusionnées en une seule relation, les attributs sont ajoutés à celle-ci.

Questions de synthèse



Quand doit-on expliciter des contraintes sur les associations ?

.....
.....
.....
.....
.....
.....
.....

Le passage UML vers relationnel est-il systématique ou soumis à interprétation (pourrait-il être réalisé par un algorithme) ?

.....
.....
.....
.....
.....
.....
.....

Pourquoi dispose-t-on de plusieurs solutions pour traduire une association 1:1 ?

.....
.....
.....
.....
.....
.....
.....

Quand doit-on utiliser les paquetages ?

.....
.....
.....
.....
.....



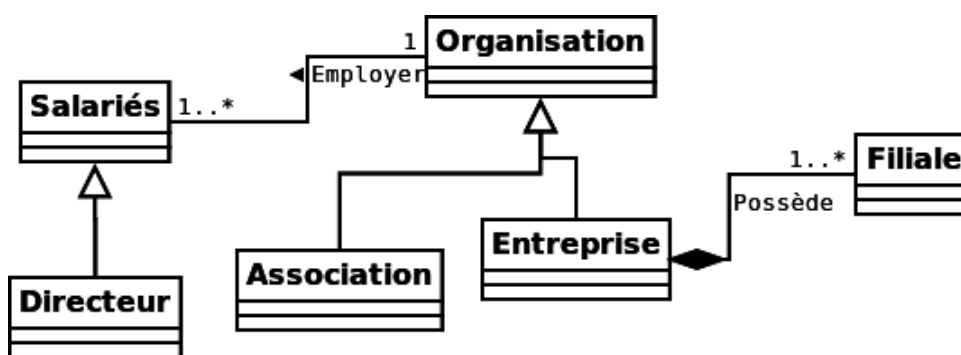
.....
.....

Solutions des exercices

> Solution n°1

Exercice p. 5

En analysant le schéma UML ci-après, sélectionner toutes les assertions vraies.



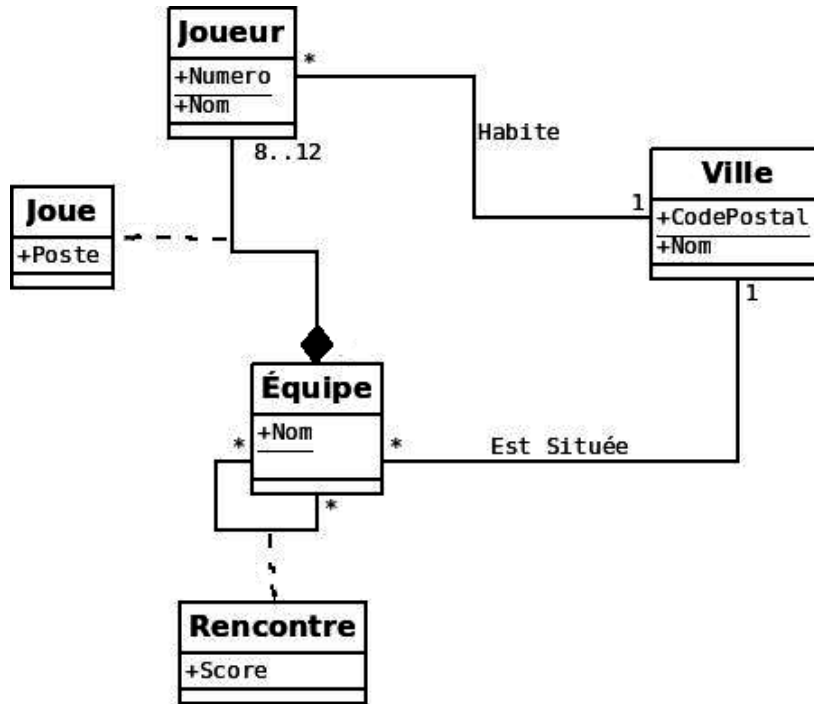
MCD UML

- Une association peut employer un directeur.
- Une association peut employer plusieurs directeurs.
- Une association peut ne pas employer de directeur.
- Une filiale peut appartenir à plusieurs entreprises.
- Il existe des organisations qui ne sont ni des entreprises ni des associations.

> Solution n°2

Exercice p. 16

Quel(s) schéma(s) relationnel(s) corresponde(nt) au modèle UML suivant ?



Volley ball

Joueur(#Numero, Nom, Ville=>Ville)

Equipe(#Nom, Joueur=>Joueur, Poste, Rencontre=>Equipe, Score, Ville=>Ville) Ville(#CodePostal, Nom)

Joueur(#Numero, #Equipe=>Equipe, Nom, Poste, Equipe(#Nom, Ville=>Ville) Ville=>Ville)

Ville(#CodePostal, Nom) Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Score)

Joueur(#Numero, Nom, Equipe(#Nom, Joueur=>Joueur, Poste, Ville=>Ville) Ville=>Ville)

Ville(#CodePostal, Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Nom) Score)

Joueur(#Numero, Equipe Ville(#CodePostal, Nom) (#Nom) Nom)

Habite(#Numero=>Joueur, #CodePostal=>Ville)

Joue(#Numero=>Joueur, #Nom=>Equipe, Poste)

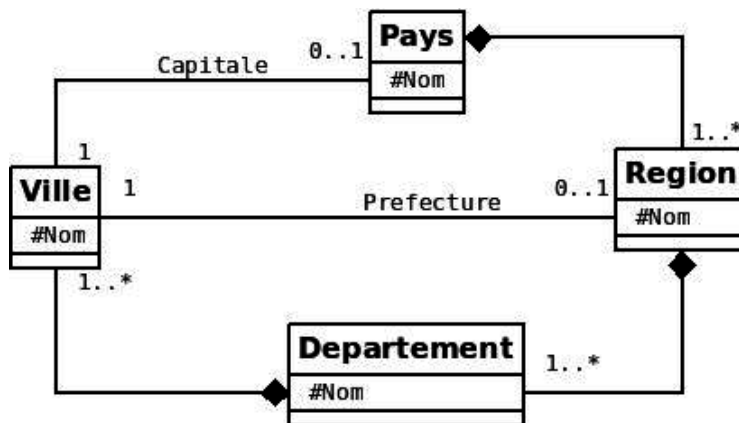
EstSituée(#Nom=>Equipe, #CodePostal=>Ville)

Rencontre(#NomE1=>Equipe, #NomE2=>Equipe, Score)

> **Solution n°3**

Exercice p. 17

Soit le schéma UML suivant :



Villes

Quelles sont les modèles relationnels qui correspondent à ce modèle conceptuel ?

Pays(#Nom, Capitale=>Ville) Region(#Nom, Prefecture=>Ville, Pays=>Pays)

Departement(#Nom, Region=>Region) Ville(#Nom, Departement=>Departement)

Pays(#Nom, Capitale=>Ville) Pays=>Pays Region(#Nom, Prefecture=>Ville,

Departement(#Nom, Region=>Region, Pays=>Pays)

Ville(#Nom, Departement=>Departement, Region=>Region, Pays=>Pays)

Pays(#Nom) Pays=>Pays Region(#Nom, Departement(#Nom, Region=>Region)

Ville(#Nom, Departement=>Departement, Capitale=>Pays, Prefecture=>Region)

Pays(#Nom, Capitale=>Ville) Region(#Nom, #Pays=>Pays, Prefecture=>Ville)

Departement(#Nom, #Region=>Region) Ville(#Nom, #Departement=>Departement)

Pays(#Nom, Capitale=>Ville, Capitale=>Ville, Capitale=>Ville, Capitale=>Ville, Capitale=>Ville)

Region(#Nom, #Pays=>Pays, Prefecture=>Ville, Prefecture=>Ville, Prefecture=>Ville, Prefecture=>Ville, Prefecture=>Ville)

Departement(#Nom, #Region=>Region, #Pays=>Region)

Ville(#Nom, #Departement=>Departement, #Region=>Region, #Pays=>Region)

Seule la dernière proposition est correcte, car toutes les entités (sauf pays) sont faibles et héritent donc de la clé primaire de leur entité identifiante (car les associations sont ici des associations de composition).

1. Clé de Pays = Nom
2. Clé de Région = NomRégion + clé de pays (entité faible) = NomR+NomP
3. Clé de Département = NomDep + clé de région (entité faible) = NomD+NomR+NomP
4. Clé de Ville = NomVille + clé de département (entité faible) = NomV+NomD+NomR+NomP

> Solution n°4

Exercice p. 31

Remarque

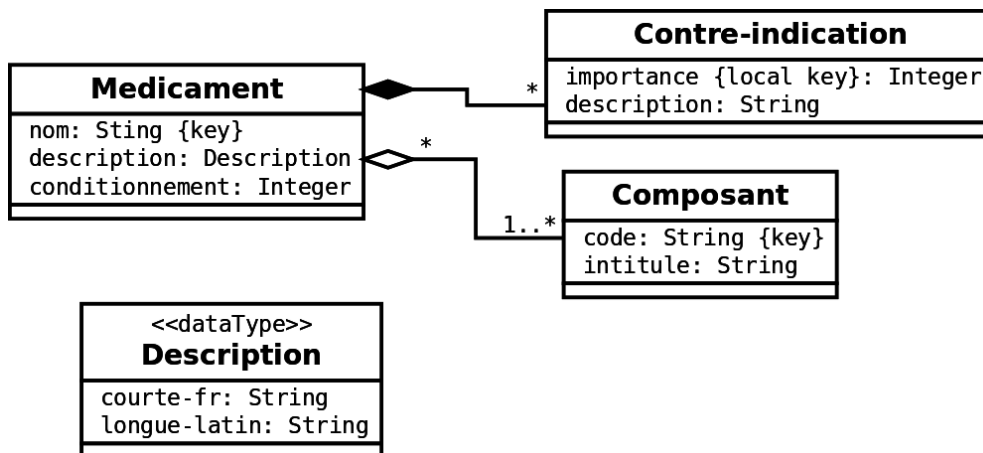
On note dans l'énoncé que les contre-indications sont *dédiée* aux médicaments, et par ailleurs on note dans les données exemples que les contre-indications sont énoncées spécifiquement par rapport aux médicaments.

On a donc affaire à une composition.

Remarque

Ce n'est pas parce que les composants s'appellent ainsi dans l'énoncé que l'on est en présence d'une composition. Ici les composants peuvent exister sans médicament et être partagés entre plusieurs médicaments, il s'agit donc d'une agrégation.

Modélisation de la base en UML



- On rappelle que les associations doivent être obligatoirement nommées, sauf dans le cas d'agrégations et de compositions, qui sont des associations déjà nommées.
- Les contre-indications peuvent également être modélisées comme un attribut multivalué.

Rappel

Associations (cf. p.)

Composition (cf. p.5)

Complément

On a choisi d'utiliser ici le stéréotype `dataType`, on aurait pu choisir deux attributs dans *Medicament* ou un attribut multivalué.

Type utilisateurs : stéréotype `<<dataType>>` (cf. p.24)

> Solution n°5

Exercice p. 31

```

1 Medicament (#nom:vvarchar, description:vvarchar, description_longue:vvarchar,
  conditionnement:number)
2 Contre_indication (#medicament=>Medicament, #importance:vvarchar, description:
  vvarchar)
3 Composant (#code:vvarchar, intitule:vvarchar)
4 Composition (#medicament=>Medicament, #composant=>Composant)
5

```

Remarque

En l'absence de clé locale naturelle pour *Contre_indication*, on ajoute une clé locale artificielle `pknum` (il y aura plusieurs valeurs identiques de `pknum`).

> Solution n°6

Exercice p. 31

Medicament

#nom	description	desc_longue	cond.
Chourix	Médicament contre la chute des choux	Vivamus...	11
Tropas	Médicament contre les dys...	Suspendisse...	42

Contre_indication

#importance	#medicament=>Medicament	description
1	Chourix	Ne jamais prendre après minuit
2	Chourix	Ne jamais mettre en contact avec l'eau.
1	Tropas	Garder à l'abri de la lumière du soleil

Composant

#code	intitule
HG79	Vif-argent allégé
HG81	Vif-argent alourdi
SN50	Pur étain

Composition

#medicament=>Medicament	#composant=>Composan
Chourix	HG79
Chourix	SN50
Tropas	HG79

> Solution n°7

Exercice p. 32

```

1 diplome (#code:char(3), intitule:vvarchar) avec intitule KEY

```

```

2 etudiant (#num:char(10), nom:varchar, origine=>diplome) avec origine NOT NULL
3 uv (#code:char(5), titre:varchar) avec titre KEY
4 note (#etu=>etudiant, #uv=>uv, #semestre:char(5), resultat:integer)

```

> Solution n°8

Exercice p. 32

```

1 Personnage (#designation:string, prix:float, identiteSecrete:string, genre:
  {superhéro, supervilain}, mentor=>Personnage(designation))
2
3 Vehicule (#designation:string, prix:float, nbPlaces:integer, type:{terrestre,
  volant, aquatique}, conducteur=>Personnage(designation)) WITH conducteur NOT NULL
4
5 Repaire (#designation:string, prix:float, emplacementGeographique:string,
  capacite:integer)
6
7 Membre (#proprietaire=>Personnage(designation), #nom:{tête, bras gauche, bras
  droit, torse, jambe gauche, jambe droite}, couleur:string)
8
9 Abrite (#repaire=>Repaire(designation), #personnage=>Personnage(designation))
10
11 Contrainte : PROJ(Personnage.designation) IN PROJ(Membre.proprietaire)

```

Remarque : Héritage

L'héritage est exclusif, un personnage ne peut pas être un véhicule ou un repaire, et la classe mère est abstraite. Il s'agit donc de l'un des cas simples de transformation : héritage par les classes filles.

L'héritage est non complet, un héritage par la classe mère serait une erreur. L'héritage par référence est une mauvaise solution qui complexifie pour rien le modèle.

Remarque : Clé candidate "identiteSecrete"

Nous pourrions décider que *identiteSecrete* est une clé candidate, bien que la diagramme UML ne le précise pas.

Remarque : Vue vFigurine

La vue vFigurine est optionnelle, la classe mère étant abstraite.

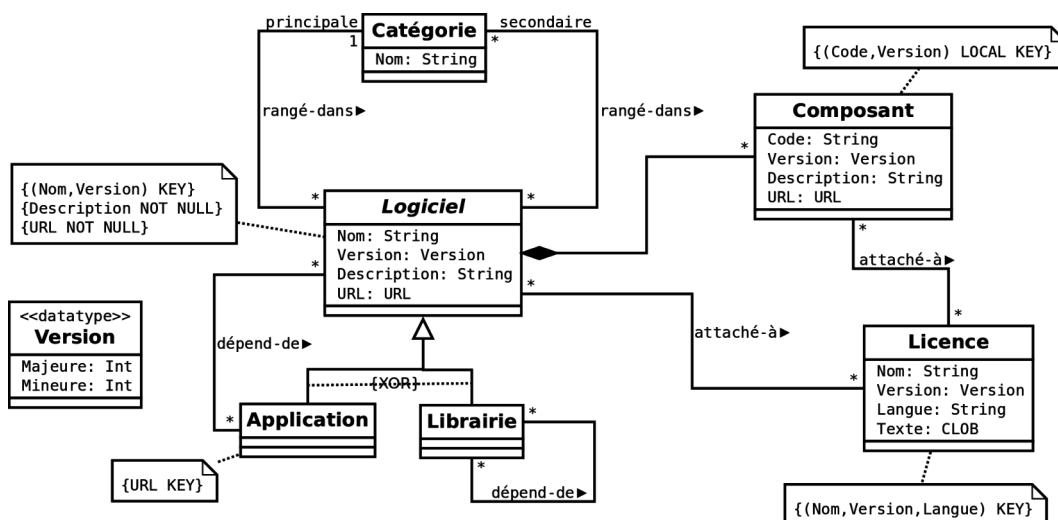
```

vFigurine = Union ( Union (Projection(Personnage,designation,prix),
  (Projection(Vehicule,designation,prix)), Projection(Repaire,designation,
  prix))

```

> **Solution n°9**

Exercice p. 33

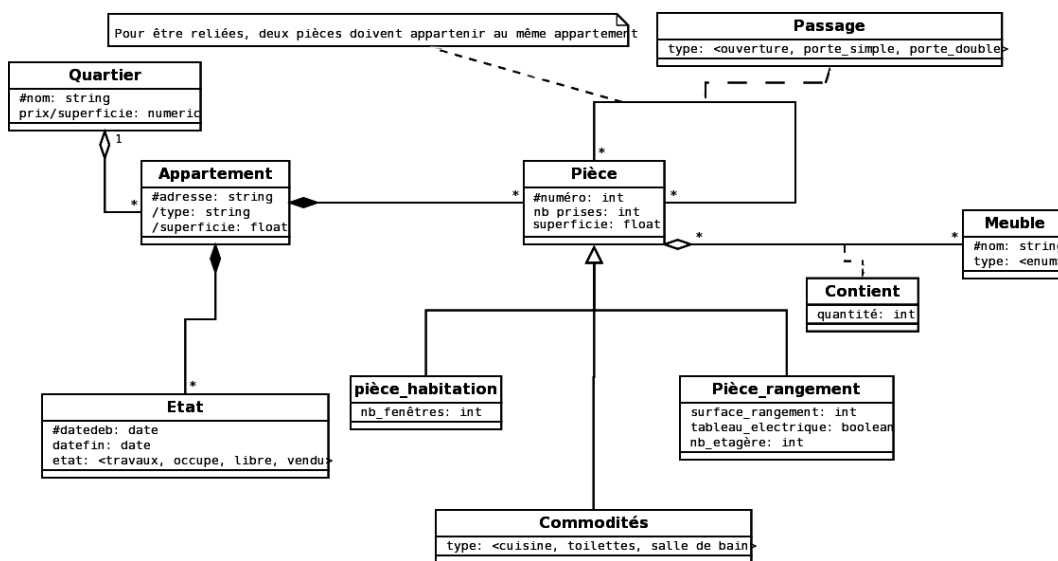


Remarque

On aurait pu ajouter une classe abstraite ONL dont hériterait Logiciel et Composant et qui factoriserait version et URL. L'intérêt principal serait de faire apparaître la notion d'ONL.

> **Solution n°10**

Exercice p. 35



MCD UML

Remarque

La notation des clés avec # n'est pas standard en UML on préférera exprimer la contrainte de type clé en notant : {key}

> **Solution n°11**

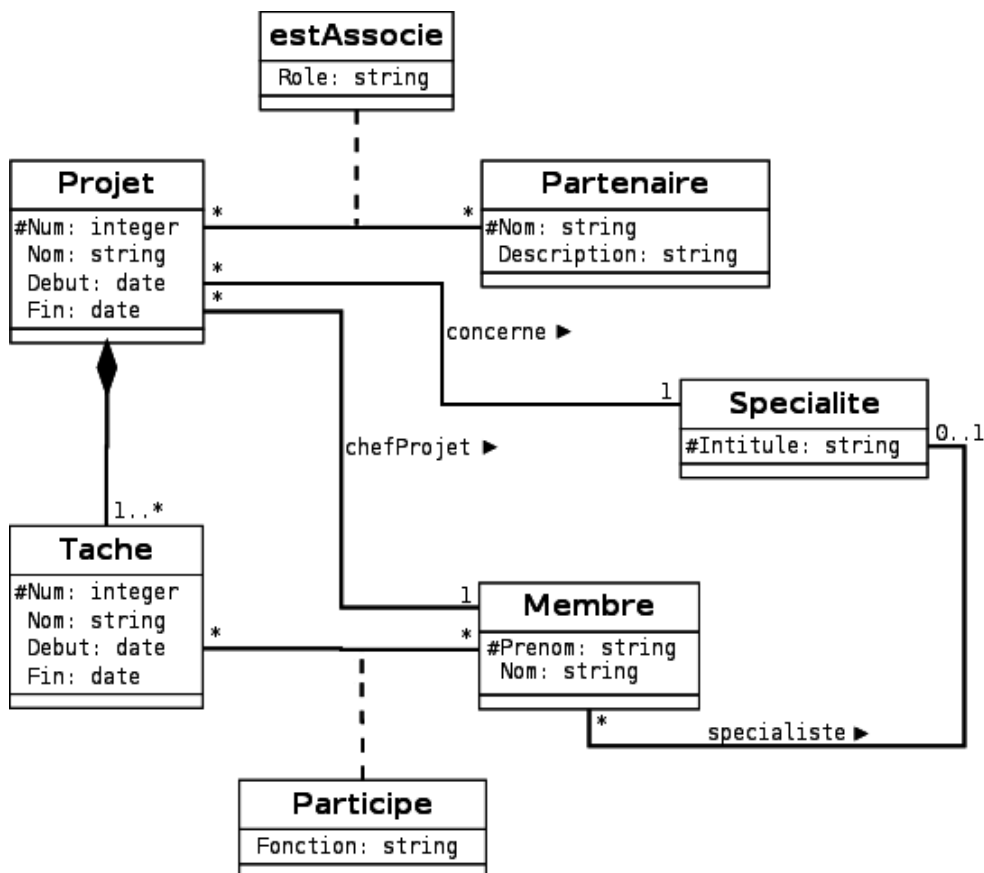
Exercice p. 37

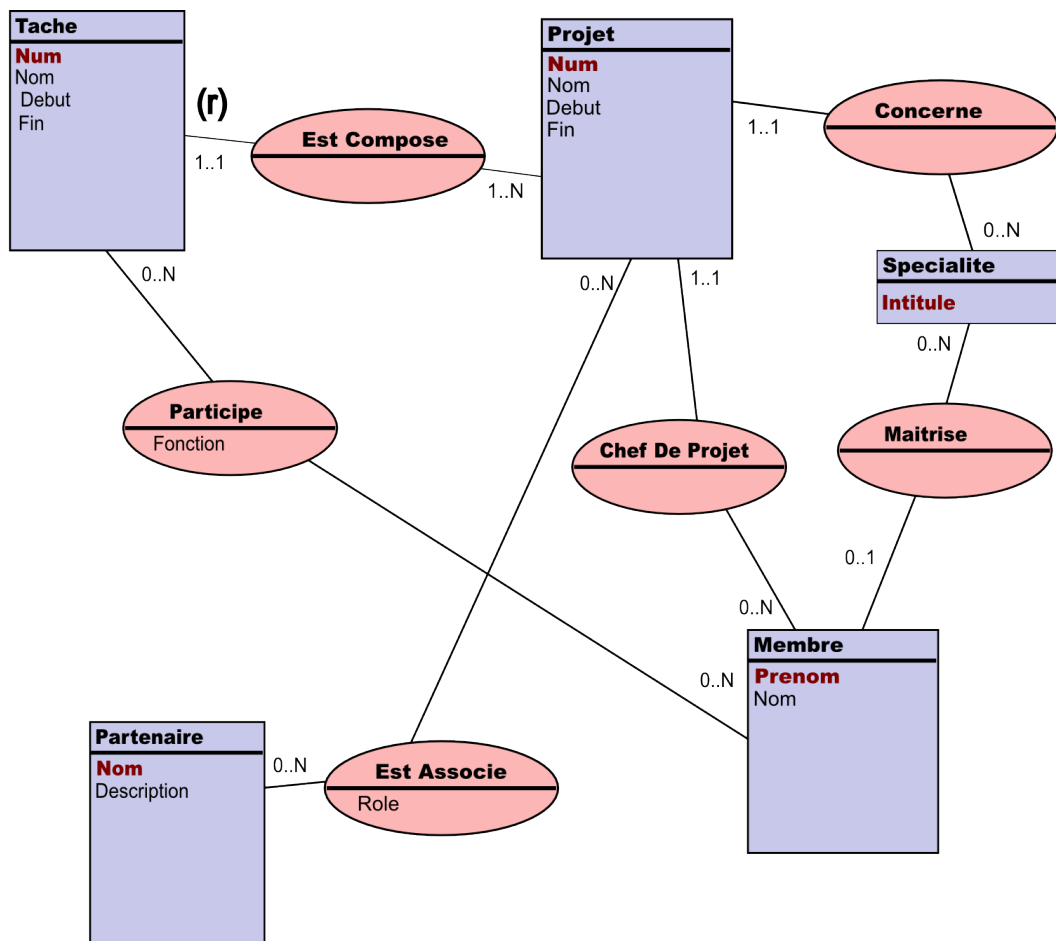
☞ *Exemple : Exemple d'analyse*

- Un projet est défini par un numéro, son nom, sa date de début et sa date de fin.
- Chaque projet est composé d'une ou plusieurs tâches, et chaque tâche est définie par son numéro, son nom, sa date de début et sa date de fin.
- Un projet a un chef de projet, membre de l'association, qui peut participer à la réalisation d'une ou plusieurs tâches.
- Un membre est défini par son prénom, son nom et sa spécialité.
- Un projet a une spécialité bien définie.
- La réalisation d'un projet se fait en collaboration avec un ou plusieurs partenaires, chacun de ces partenaires a un rôle.
- Un partenaire est défini par son nom et sa description.

> **Solution n°12**

Exercice p. 37

Exemple de modélisation UML et E-A*Modèle UML "gestion de projets"*



Modèle E-A "gestion de projets"

🔍 Remarque

Le choix du prénom comme clé ici n'est pas un bon choix en général.

🔧 Conseil

On préférera la notation {key} plutôt que # pour repérer les clés.

> Solution n°13


Exercice p. 37

Exemple de schéma relationnel

```

1 Specialite (#Intitule:string)
2 Membre (#Prenom:string, Nom:string, Specialite=>Specialite(Intitule))
3 Projet (#Num:integer, Nom:string, Debut:date, Fin:date, ChefDeProjet=>Membre
  (Prenom), Specialite=>Specialite(Intitule))
4 WITH ChefDeProjet, Specialite NOT NULL
5 Tache (#Num:integer, #Projet=>Projet(Num), Nom:string, Debut:date, Fin:date)
6 Partenaire (#Nom:string, Description:string)
7 Participe (#Prenom=>Membre(Prenom), #Tache=>Tache(Num), #Projet=>Tache(Projet),
  Fonction:string)
8 EstAssocie (#Nom=>Partenaire(Nom), #Projet=>Projet(Num), Role:string)

```


 **Fondamental**

La clé *num* de *Tâche* est locale à la classe composant, donc deux objets de cette classe *peuvent* avoir le même *num*. En revanche deux objets de cette classe avec la même clé étrangère vers *Projet* ne peuvent pas avoir le même *num*, car *(num, Projet)* est une clé.

En français :

- il peut exister deux tâches numéro 1 dans deux projets différents
- il ne peut pas exister deux tâches numéro 1 au sein du même projet

num	projet	...	
1	1		
1	2		
4	2		

 **Complément**

Contraintes : PROJ(Projet, Num) IN PROJ(Tache, Projet)

Glossaire

Clé (key)

En UML et relationnel on parle de clé pour désigner un groupe d'attributs qui permet d'identifier un objet.

Clé alternative (alternate key)

En relationnel les clés alternatives sont les clés candidates non retenues comme clés primaires.

Clé artificielle (surrogate key)

En relationnel une clé artificielle est un attribut artificiel (qui ne représente aucune donnée) ajouté à la relation pour servir de clé primaire. On mobilise cette technique lorsque l'on n'a pas pu ou voulu choisir une clé naturelle pour clé primaire.

Clé candidate (candidate key)

Clé candidate est un synonyme pour clé, toutes les clés sont candidates. En fait on parle juste de clé au niveau conceptuel. En relationnel on parle de clés candidates pour différencier les clés alternatives (qui n'ont pas été retenues comme clé primaire) de la clé primaire (la clé candidate qui a été élue pour effectuer les références par clé étrangère).

Clé étrangère (foreign key)

Une clé étrangère est en relationnel un ensemble d'attributs qui référence les attributs de la clé primaire de la même relation ou d'une autre relation. La clé étrangère est plutôt mal nommée car ce n'est pas une clé (elle n'identifie pas un objet), mais une référence à une clé.

Clé locale (local key)

Dans certaines constructions au niveau conceptuel (association N:M et composition) la clé peut être locale, c'est à dire qu'au lieu d'identifier pleinement un objet (comme une clé classique), elle identifie un objet étant donné un contexte (les membres de l'association N:M ou l'objet composite).

Une clé locale n'est donc pas une clé au sens relationnel, elle ne permet pas d'identifier un enregistrement, mais elle deviendra une partie d'une clé lors du passage au relationnel.

Clé naturelle (natural key, business key)

Clé naturelle est synonyme de clé au niveau conceptuel (toutes les clés sont naturelles à ce stade). En relationnel, les clés naturelles sont toutes les clés candidates non artificielles.

Clé primaire (primary key)

Une clé primaire est une clé candidate qui a été choisie pour être référencée par les autres relations par le mécanisme des clés étrangères.

Abréviations



BD : Base de Données

E-A : Entité-Association

Bibliographie

Roques Pascal, Vallée Franck. *UML 2 en action : De l'analyse des besoins à la conception J2EE*. ISBN 2-212-11462-1 (3ème édition). Paris : Eyrolles, 2004. 385 p. architecte logiciel.

Pascal Roques, *UML 2 par la pratique*, 7e édition, Eyrolles, 2009 [ISBN 978-2212125658]

Webographie



Dia, <http://live.gnome.org/Dia>

Morlon Jérôme, *UML en 5 étapes*, http://developpeur.journaldunet.com/dossiers/alg_uml.shtml, 2004.

Borderie Xavier, *Cinq petits conseils pour un schéma UML efficace*, http://developpeur.journaldunet.com/tutoriel/cpt/031013cpt_uml5conseils.shtml, 2004.

Objecteering software. <http://www.objecteering.com>. [2002-septembre].

UML en Français, <http://uml.free.fr>, consulté en 2002.