Chapitre 3

Diagrammes UML
Vue Statique

2. Diagramme de classes

2.1 Introduction

Un diagramme de classes est un graphe d'éléments connectés par des relations.
Un diagramme de classes est une vue graphique de la structure statique d'un système (car on ne tient pas compte du facteur temporel dans le comportement du système), exprimée en termes de classes et de relations entre ces classes. Une classe décrit un ensemble d'objets et une association décrit un ensemble de liens; les objets sont instances des classes et les liens sont instances des associations.

Diagramme de classes = classes + relations

Intérêt

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet, il est le seul obligatoire lors d'une telle modélisation. Il permet de définir quelles seront les composantes du système final : il ne permet en revanche pas de définir le nombre et l'état des instances individuelles. Néanmoins, on constate souvent qu'un diagramme de classes proprement réalisé permet de structurer le travail de développement de manière très efficace; il permet aussi, dans le cas de travaux réalisés en groupe (ce qui est pratiquement toujours le cas dans les milieux industriels), de séparer les composantes de manière à pouvoir répartir le travail de développement entre les membres du groupe. Enfin, il permet de construire le système de manière correcte.

Si le diagramme de cas d'utilisation montre un système du point de vue des acteurs, le diagramme de classes en montre la structure interne. Il permet de fournir une représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas d'utilisation.
En UML, le modèle structurel ou statique d'un système est décrit à l'aide de deux sortes de diagrammes :

- Diagrammes de classes (description de tout ou d'une partie du système d'une manière abstraite).
- Diagrammes d'objets (description d'exemples de configuration de tout ou partie du système, en termes d'objets, de valeurs et de liens).

Pour un modèle complexe, plusieurs diagrammes de classes complémentaires peuvent être construits en focalisant sur :

- les classes qui participent à un cas d'utilisation ,
- les classes associées dans la réalisation d'un scénario précis,
- les classes qui composent un paquetage,
- la structure hiérarchique d'un ensemble de classes.

Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets

2.2 Eléments et notation d'un diagramme de classes

Métalangage des syntaxes :

Nous aurons régulièrement recours à ce métalangage pour décrire des syntaxes de déclaration. Ce métalangage contient certains méta caractères :

[ ] : les crochets indiquent que ce qui est à l'intérieur est optionnel ;

< > : les signes inférieur et supérieur indiquent que ce qui est à l'intérieur est plus ou moins libre; par exemple, la syntaxe de déclaration d'une variable comme compteur : int est <nom_variable> : <type> ;

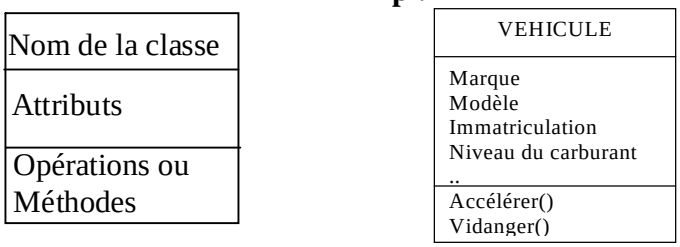
' ' : les cotes sont utiles quand on veut utiliser un méta caractère comme un caractère ; par exemple, pour désigner un crochet ([) il faut écrire '[' car [est un méta-caractère ayant une signification spéciale ;

... : permet de désigner une suite de séquence de longueur non définie, le contexte permettant de comprendre de quelle suite il s'agit.

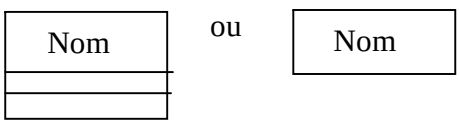
2.2.1 Classe

La classe intègre les concepts de type (en tant que «moule à instances ») et de module (avec l'idée d'interface + corps).
Une classe est un classeur représenté par un rectangle divisé en trois à cinq compartiments. Le premier indique une chaîne de caractères correspondant au nom de la classe qui doit commencer par un caractère alphabétique et ne pas contenir le caractère **'::'**. Le deuxième compartiment indique les attributs de la classe et le troisième ses opérations.

Exp :



Représentation simplifiée :



Ne pas représenter les attributs ou les méthodes d'une classe sur un diagramme, n'indique pas que cette classe n'en contient pas. Il s'agit juste d'un filtre visuel, destiné à donner un certain niveau d'abstraction à son modèle.

Remarque : Un compartiment des responsabilités peut être ajouté pour énumérer l'ensemble de

tâches devant être assurées par la classe. Un autre compartiment pour des exceptions peut également être ajouté pour énumérer les situations exceptionnelles devant être gérées par la classe.

La notion de classe est essentielle en programmation orientée objets : elle définit une abstraction, un type abstrait qui permettra plus tard d'instancier des objets. On distingue généralement entre classes abstraites (qui ne peuvent pas être instanciées) et classes "normales", qui servent à définir des objets.

Visibilité des éléments : La visibilité déclare la possibilité pour un élément de modélisation de référencer un élément qui se trouve dans un espace de noms différents de celui de l'élément qui établit la référence. Elle fait partie de la relation entre un élément et le conteneur qui l'héberge, ce dernier pouvant être un paquetage, une classe ou un autre espace de noms. Dans une classe, le marqueur de visibilité se situe au niveau de chacune de ses caractéristiques (attributs, terminaisons d'association et opération). Il permet d'indiquer si une autre classe peut y accéder.

- La visibilité est:
  '+' pour public (l'élément est visible pour tous les clients de la classe)
  '#' pour protected (l'élément est visible pour les sous-classes de la classe)
  '-' pour private (l'élément n'est visible que par les objets de la classe dans laquelle il est déclaré)

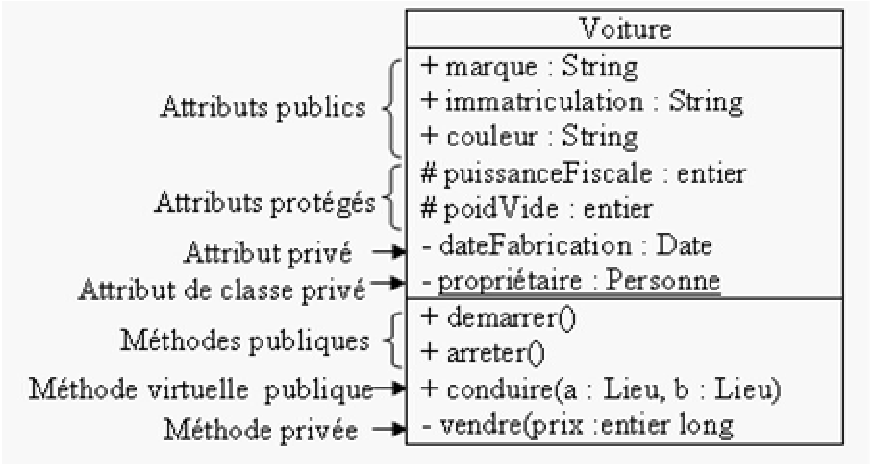
2.2.1.1 Attribut

Les attributs définissent des informations qu'une classe ou un objet doivent connaître. Chacune de ces informations est définie par un nom, un type de données, une visibilité et peut être initialisé. Le nom de l'attribut doit être unique dans la classe. La syntaxe de la déclaration d'un attribut est la suivante :

La syntaxe d'un attribut est :

<visibilité> [/] <nom_attribut>:
<Type> ['[' <multiplicité> ']' ['{' <contrainte > '}']] [= <valeur_initiale>]

- Le type de l'attribut (<Type>) peut être un nom de classe, un nom d'interface ou un type de donné.
- UML définit son propre ensemble de types Integer, real, string, …
- La multiplicité (<multiplicité>) d'un attribut précise le nombre de valeurs que l'attribut peut contenir. Par défaut, la multiplicité est **1,1**(c-a-d. une et une seule valeur).
  Exp: Dans une classe Module, l'attribut nom_module[1,10], signifie que l'on peut indiquer de 1 à 10 noms de modules.
- Lorsqu'une multiplicité supérieure à 1 est précisée, il est possible d'ajouter une contrainte (<contrainte>) pour préciser si les valeurs sont ordonnées ({ordered}) ou pas ({list}).
- Un attribut peut être un attribut de classe, il est alors souligné. Il garde une valeur unique et partagée par toutes les instances de la classe. Les instances ont accès à cet attribut mais n'en possèdent pas une copie.
- Un attribut peut être dérivé (calculés à partir d'autres attributs et de formules de calcul), il est alors préfixé par le caractère '/'.



2.2.1.2 Méthode

Dans une classe, une opération doit être unique. Quand le nom d'une opération apparaît plusieurs fois avec des paramètres différents, on dit que l'opération est surchargée. En revanche, il est impossible que deux opérations ne se distinguent que par leur valeur retournée.

La syntaxe:d'une opération est la suivante :

<visibilité> <nom_méthode> ([<paramètre_1>, ... , <paramètre_N>]) : [<type_renvoyé>] [{<propriétés>}]

La syntaxe de définition d'un paramètre (<paramètre>) est la suivante :

[<direction>] <nom_paramètre>:<type> ['['<multiplicité>']'] [=<valeur_par_défaut>

La direction peut prendre l'une des valeurs suivante :

in : Paramètre d'entrée passé par valeur. Les modifications du paramètre ne sont pas disponibles pour l'appelant. C'est le comportement par défaut.
out : Paramètre de sortie uniquement. Il n'y a pas de valeur d'entrée et la valeur finale est disponible pour l'appelant.
inout : Paramètre d'entrée/sortie. La valeur finale est disponible pour l'appelant.

Le type du paramètre (**<type>**) peut être un nom de classe, un nom d'interface ou un type de donné prédéfini.

Les propriétés (**<propriétés>**) correspondent à des contraintes ou à des informations complémentaires comme les exceptions, les préconditions, les postconditions ou encore l'indication qu'une méthode est abstraite (mot-clef *abstract*), etc.

Méthode de classe

Comme pour les attributs de classe, il est possible de déclarer des méthodes de classe. Une méthode de classe ne peut manipuler que des attributs *de classe* et ses propres paramètres. Cette méthode n'a pas accès aux attributs *de la classe* (*i.e.* des instances de la classe). L'accès à une

méthode de classe ne nécessite pas l'existence d'une instance de cette classe. Graphiquement, une méthode de classe est soulignée.

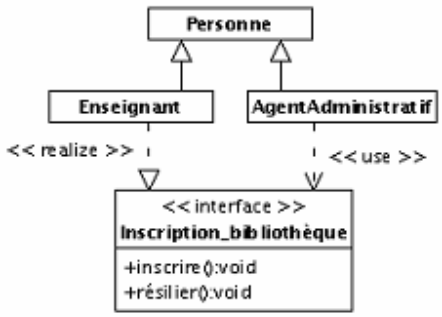
2.2.1.3 Les interfaces

Les interfaces représentent l'élément le plus abstrait du diagramme de classes. L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet. Une interface est formellement équivalente à une classe abstraite sans attributs ayant uniquement des opérations abstraites.

Un classeur, stéréotypé « interface » permet de regrouper un ensemble de propriétés et d'opérations assurant un service cohérent. Il peut s'agir de l'interface complète d'un objet, ou simplement d'une partie d'interface qui sera commune à plusieurs objets.

Une interface doit être réalisée par au moins une classe. Graphiquement, cela est représenté par un trait discontinu terminé par une flèche triangulaire et le stéréotype « realize ». Une classe (classe cliente de l'interface) peut dépendre d'une interface (interface requise). On représente cela par une relation de dépendance et le stéréotype « use ».

Exp :

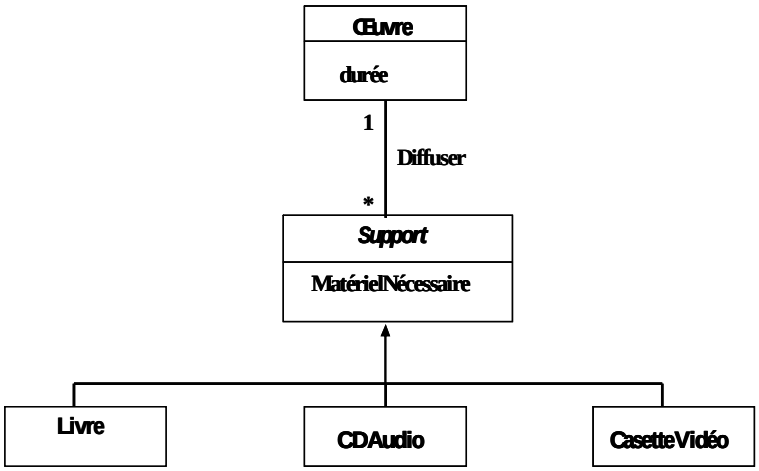


2.2.1.4 Méthodes et classes abstraites

Une méthode est dite abstraite lorsqu'on connaît son entête mais pas la manière dont elle peut être réalisée (*i.e*. on connaît sa déclaration mais pas sa définition).

Une classe est dite abstraite lorsqu'elle définit au moins une méthode abstraite ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée. On ne peut instancier une classe abstraite : elle est vouée à être spécialisée par des classes concrètes (instanciables) qui précisent les méthodes abstraites. Elle doit toujours être suivie de ***classes dérivées***.

Exp :



Classe abstraite : *Support* **Classes dérivées :** Livre, CD Audio, Cassette Vidéo

Une classe abstraite peut très bien contenir des méthodes concrètes.

Une classe abstraite pure ne comporte que des méthodes abstraites. En programmation orientée objet, une telle classe est appelée une interface.

Pour indiquer qu'une classe est abstraite, on ajoute le mot-clef *abstract* derrière son nom.

La syntaxe de base de la déclaration d'un nom d'une classe est la suivante :

<Nom_de_la_classe> [{ [abstract], [<auteur>], [<date>], ... }]

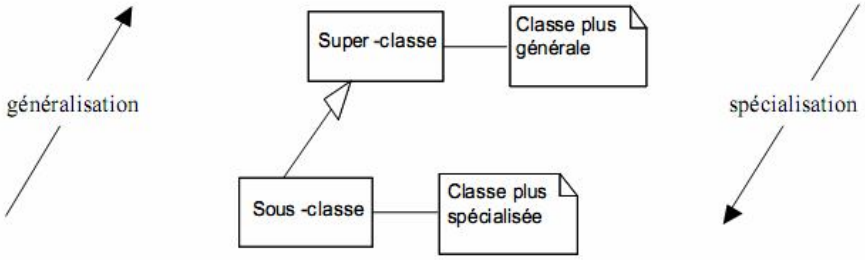
2.3 Relations entre classes

2.3.1 Héritage (Généralisation/Spécialisation)

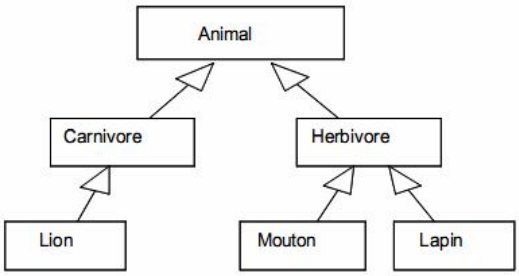
La généralisation décrit une relation entre une classe générale (super classe ou classe parent) et une classe spécialisée (sous-classe ou classe enfant). La classe spécialisée est intégralement cohérente avec la classe de base, mais comporte des informations supplémentaires (attributs, opérations, associations). L'héritage multiple (plusieurs super classes) est possible.

Dans le langage UML, ainsi que dans la plupart des langages objet, cette relation de généralisation se traduit par le concept d'héritage. On parle également de relation d'héritage. Ainsi, l'héritage permet la classification des objets. Le symbole utilisé pour la relation d'héritage ou de généralisation est une flèche avec un trait plein dont la pointe est un triangle fermé

désignant le cas le plus général.



Exemple :



Les propriétés principales de l'héritage sont :

- La classe enfant possède toutes les propriétés de ses classes parents, mais elle ne peut accéder aux propriétés privées de celle-ci.
- Une classe enfant peut redéfinir (même signature) une ou plusieurs méthodes de la classe parent.
- Sauf indication contraire, un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes.
- Toutes les associations de la classe parent s'appliquent aux classes dérivées.
- Une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue.

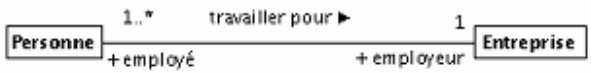
2.3.2 Association

Une association est une relation entre deux classes (association binaire) ou plus (association n-aire), qui décrit les connexions structurelle entre leurs instances ("est produit par", "est affilié à","se trouve à", "est conduit par"…).

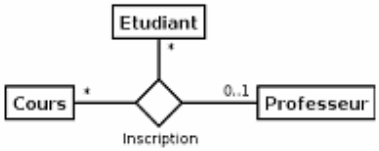
Exp :

Association Binaire : Une association binaire est matérialisée par un trait plein entre les classesssociées qui peut être orné d'un nom, avec éventuellement une précision du sens de

lecture(► ou ◄). Si les deux extrémités de l'association pointent vers une même classe, l'association est dite réflexive.



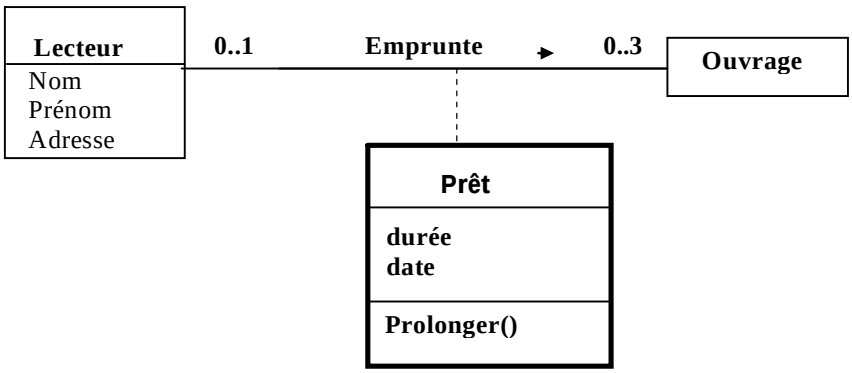
Association n-aire



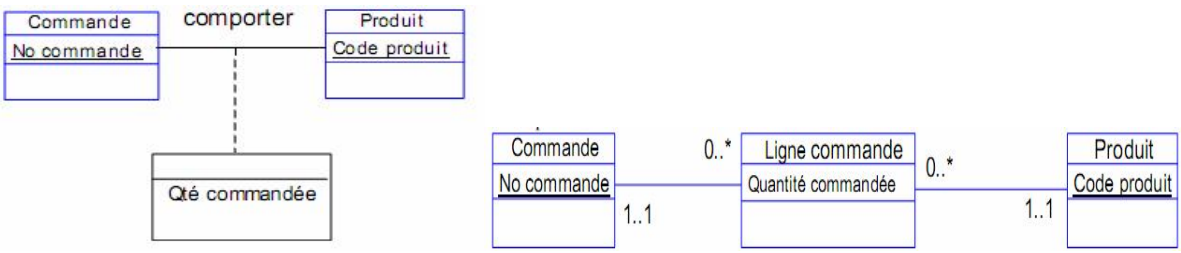
2.3.3 Classe-association

Une classe-association possède les propriétés des associations et des classes : elle se connecte à deux ou plusieurs classes et possède également des attributs et des opérations. Une classe-association est caractérisée par un trait discontinu entre la classe et l'association qu'elle représente.

Exp :



Remarque : Il est toujours possible de se passer des classes associations. Toute classe-association peut être remplacée par une classe intermédiaire et qui sert de pivot pour une paire d'association.

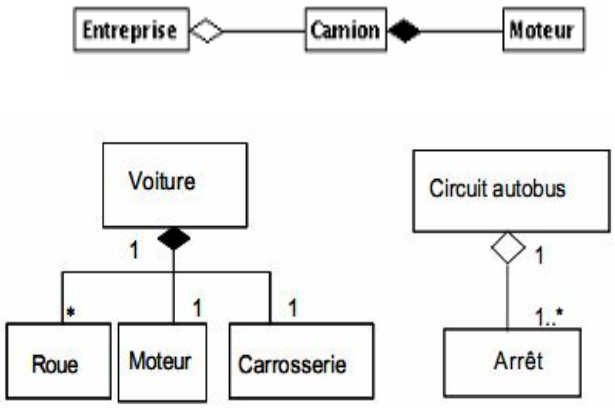


2.3.4 Agrégation et composition

Une agrégation est une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. Graphiquement, on ajoute un losange vide (◊) du côté de l'agrégat.

La composition, également appelée agrégation composite, décrit une contenance structurelle (contenance physique) entre instances. Ainsi, la destruction de l'objet composite implique la destruction de ses composants. Graphiquement, on ajoute un losange plein (♦) du côté de l'agrégat.

Exp :



Agrégation **Composition**

2.3.5 Relation de dépendance

Une dépendance est une relation unidirectionnelle exprimant une dépendance sémantique entre les éléments du modèle. Elle est représentée par un trait discontinu orienté. Elle indique que la modification de la cible implique une modification de la source.

Exp :

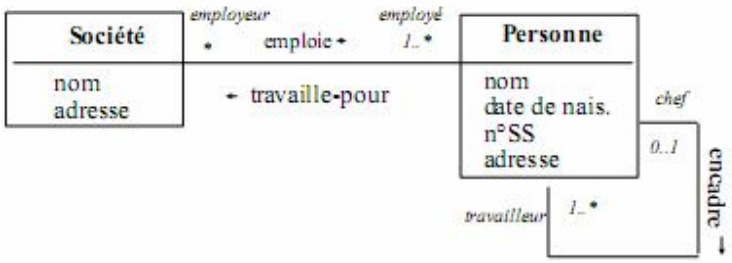
La dépendance est souvent stéréotypée pour mieux expliciter le lien sémantique entre les éléments du modèle (« derive », « refine », « use », « realize », « trace », « create », « instantiate », …).

2.3.6 Multiplicité ou cardinalité

La cardinalité associée à une terminaison d'association, d'agrégation ou de composition déclare le nombre d'objets susceptibles d'occuper la position définie par la terminaison considérée (montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe).

n	Exactement n (entier naturel)
m..n	de m à n (Au moins m et au plus n)
0..1	De zéro à un (0 ou 1)
*	De zéro à plusieurs (zéro ou plusieurs)
0..*	
n..*	De n à plusieurs (Au moins n)

Exp :

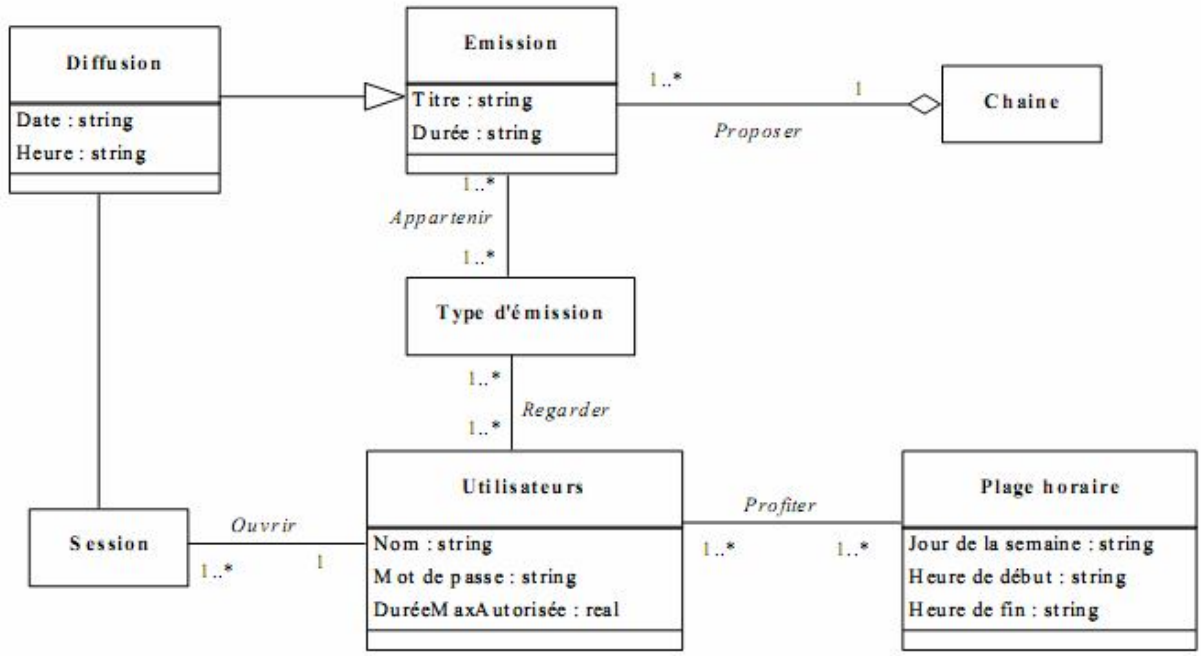


Dans cet exemple, toute société emploie au moins un employé. Toute personne peut avoir plusieurs emploies dans une société (elle peut ne pas en avoir) et a au plus un seul chef. La personne chef encadre au moins un travailleur.

2.4 Elaboration d'un diagramme de classe

Une démarche couramment utilisée pour élaborer un diagramme de classes consiste à :

- **Trouver les classes du domaine étudié**. Cette étape empirique se fait généralement en collaboration avec un expert du domaine. Les classes correspondent généralement à des concepts ou des substantifs du domaine.
- **Trouver les associations entre classes.** Les associations correspondent souvent à des verbes, ou des constructions verbales, mettant en relation plusieurs classes, comme « est composé de », « pilote », « travaille pour ». Il faut se méfier de certains attributs qui sont en réalité des relations entre classes.
- **Trouver les attributs des classes.** Les attributs correspondent souvent à des substantifs, ou des groupes nominaux, tels que « la masse d'une voiture » ou « le montant d'une transaction ». Les adjectifs et les valeurs correspondent souvent à des valeurs d'attributs.
- **Organiser et simplifier le modèle** en éliminant les classes redondantes et en utilisant l'héritage.
- **Vérifier les chemins d'accès aux classes**.
- **Itérer et raffiner le modèle**. Un modèle est rarement correct dès sa première construction. La modélisation objet est un processus non pas linéaire mais itératif.

Exp : Pour la réalisation d'un ensemble de services relatifs à la télévision (accès à des

chaînes thématiques, contrôle des utilisations et des utilisateurs, programme électronique détaillé des chaînes accessibles...) :
Les chaînes proposent des émissions.
Une émission peut être l'objet de plusieurs diffusions. Une émission a une certaine durée.
Une diffusion a une certaine date/heure de début.
Une plage horaire est relative à un certain jour de la semaine, elle commence et se termine à des heures rondes.
Une session est l'utilisation du système par un utilisateur, d'une certaine date/heure de début à une certaine date/heure de fin, au cours de laquelle il peut zapper d'une chaîne à une autre.
Une session n'est autorisée que si l'utilisateur qui la déclenche est un utilisateur autorisé pour le type d'émission, correspondant à la diffusion regardée, Si l'heure de début de session appartient à une plage horaire autorisée pour cet utilisateur, et Si le crédit hebdomadaire de cet utilisateur n'est pas atteint.
Proposer une première version du diagramme de classe faisant apparaître les classes entités et les attributs nécessaires à la vision statique de ce diagramme de classe.