DIAGRAMME UML DE MODELISATION STATIQUE

Chargé du cours : Mr APEKE

Membres du groupe :

AMOUZOU Kévin

KLOUTSE Daniel

NYAKOU Améyo Jessica

Sommaire

Introduction

Comme n'importe quel type de projet, un projet informatique nécessite une phase d'analyse, suivi d'une étape de conception. Dans la phase d'analyse, on cherche d'abord à bien comprendre et à décrire de façon précise les besoins des utilisateurs ou des clients. Que souhaitent-ils faire avec le logiciel ? Quelles fonctionnalités veulent-ils ? Pour quel usage ? Comment l'action devrait-elle fonctionner ? les diagramme UML nous permet alors de pourvoir faire cette phase d'analyse.

Pourquoi et comment modéliser

Un modèle est une représentation abstraite et simplifier du monde réel en vue de le décrire, de l'expliquer ou de le prévoir.

Concrètement un modèle permet de réduire la complexité d'un phénomène en éliminant les détails qui n'influencent pas le comportement de manière significative.

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence. Un modèle est un langage commun, précis, qui est connu par tous les membres de l'équipe et il est donc, à ce titre, un vecteur privilégié pour communiquer.

Historique

Au début des années 90, il existait une cinquantaine de méthode objet et tous avaient une idées communes (l'utilisation des classes, Objet, association, cas d'utilisation …) ce qui entraina des confusions. Au lieu de fait avancée la technologie des objets, ceux-ci au contraire le ralenti. L'objectif est de créer un langage de modélisation utilisable à la fois par les humains et les machines permettant d'identifier la sémantique des concepts de base, classer les concepts, construire un métamodèle, choisir une notation graphique et regrouper par niveau d'abstraction, complexité et domaine. Il fallait donc aller à la recherche d'un langage commun unique, utilisable par tous les méthodes, adapté à tous les phases de développement et compatible avec toutes les techniques de réalisation. Il y a eu alors une histoire d'imbrication : Booch a ammené ses catégories et sous-système, Embley les classes singletons et objets composites, Fusion la description des opération et numérotation des messages, Gamma et al. leur Frameworks, patterns et notes, Jacobson les cas d'utilisation, OMT les association, Odell les classification dynamique et éclairage sur les événements et d'autre encore L'OMG (Object Management Groupe fondé en 1989 pour standardiser et promouvoir l'objet) a alors fait un démarche d'unification. On a unifié tout d'abord OMT et Boosh en 1995 qui donnait Unified Methode 0.8 qui à son tour sera unifié avec OOSE en 1996 qui donnera naissance à la premier version d'UML 0.9, cette version va être à son tour unifier avec beaucoup d'autre méthode dont Catalysis et ROOM pour donner UML 1.1 en Novembre 1997 qui évoluera pour donner la version 1.3 en Juin 1999 qui lui aussi évoluera pour donner la version 1.4 en Fin 2001 qui contenait que 9 diagramme. La version 2.0 est alors apparu en 2005 qui a été développé avec un consortium élargi pour améliorer davantage le langage afin de refléter une nouvelle expérience sur l'utilisation. Elle contenait 13 diagrammes qui a ensuite évolué vers 14 avec la version 2.5.1.

UML dans sa version 2.5.1 propose 14 diagramme qui peuvent être utilisés dans la description d'un système. C'est diagramme sont regroupé dans deux grands ensembles les diagrammes structurels et les diagrammes de comportement.

- **Les diagrammes structurels :**
  Ces diagrammes, aux nombre de sept (7), ont vocation à représenter l'aspect statique d'un système (classes, objet, composant ...)
  - *Diagramme de classe* : ce diagramme représente la description statique du système en intégrant dans chaque classe la partie dédiée aux données et celle consacrée aux traitements. C'est le diagramme pivot de l'ensemble de la modélisation d'un système.
  - *Diagramme d'objet* : le diagramme d'objet permet la représentation d'instances des classes et des liens entre instances.
  - *Diagramme de composant* (modifié dans UML 2) : ce diagramme représente les différents constituants du logiciel au niveau de l'implémentation d'un système.
  - *Diagramme de déploiement* (modifié dans UML 2) : ce diagramme décrit l'architecture technique d'un système avec une vue centrée sur la répartition des composants dans la configuration d'exploitation.
  - *Diagramme de paquetage* (nouveau dans UML 2) : ce diagramme donne une vue d'ensemble du système structuré en paquetage. Chaque paquetage représente un ensemble homogène d'éléments du système (classes, composants...).
  - *Diagramme de profil* (nouveau dans UML2) : ce diagramme décrit un mécanisme d'extension léger à UML en définissant des stéréotypes personnalisés, des valeurs balisées et des contraintes.
- **Les diagrammes de comportement :**
  Ces diagrammes représentent la partie dynamique d'un système réagissant aux événements et permettant de produire les résultats attendu par les utilisateurs. Sept (7) diagrammes sont proposés par UML.
  - *Diagramme des cas d'utilisation* : ce diagramme est destiné à représenter les besoins des utilisateurs par rapport au système. Il constitue un des diagrammes les plus structurants dans l'analyse d'un système.
  - *Diagramme d'état-transition* (machine d'état) : ce diagramme montre les différents états des objets en réaction aux événements.
  - *Diagramme d'activités* (modifié dans UML 2) : ce diagramme donne une vision des enchaînements des activités propres à une opération ou à un cas d'utilisation. Il permet aussi de représenter les flots de contrôle et les flots de données.
  - *Diagramme de séquence* (modifié dans UML 2) : ce diagramme permet de décrire les scénarios de chaque cas d'utilisation en mettant l'accent sur la chronologie des opérations en interaction avec les objets.
  - *Diagramme de communication* (anciennement appelé collaboration) : ce diagramme est une autre représentation des scénarios des cas d'utilisation qui met plus l'accent sur les objets et les messages échangés.
  - *Diagramme global d'interaction* (nouveau dans UML 2) : ce diagramme fournit une vue générale des interactions décrites dans le diagramme de séquence et des flots de contrôle décrits dans le diagramme d'activités.

- *Diagramme de temps* (nouveau dans UML 2) : ce diagramme permet de représenter les états et les interactions d'objets dans un contexte où le temps a une forte influence sur le comportement du système à gérer.

1) Diagramme de classe

1. Définition

Dans le langage UML, les *diagrammes de classes* appartiennent à l'un des six types de diagramme structurel. Les diagrammes de classes sont fondamentaux pour le processus de modélisation des objets et modélisent la structure statique d'un système. Suivant la complexité d'un système, vous pouvez utiliser un seul diagramme de classes pour modéliser un système complet ou bien vous pouvez utiliser différents diagrammes de classes pour modéliser les composants d'un système.

Le diagramme de classes peut être utiliser pour visualiser, définir et documenter des fonctions structurelles dans vos modèles. Par exemple, pendant les phases d'analyse et de conception du cycle de développement, on peut créer des diagrammes de classes pour réaliser les fonctions suivantes :

- Capturer et définir la structure des classes et autres discriminants
- Définir les relations entre les classes et discriminants
- Illustrer la structure d'un modèle à l'aide d'attributs, d'opérations et de signaux
- Afficher les rôles et responsabilités de discriminant communs qui définissent le comportement du système
- Afficher les classes d'implémentation dans un package
- Afficher la structure et le comportement d'une ou plusieurs classes
- Afficher une hiérarchie d'héritage entre des classes et discriminants
- Afficher les travailleurs et entités comme des modèles objet métier

Le diagramme de classe est une représentation d'un ensemble de classes, d'interfaces et de paquetages ainsi que leurs relations.

Les rubriques suivantes décrivent les éléments de modèle dans les diagrammes de classes :

- Classes
  Dans le langage UML, une classe représente l'abstraction d'un objet ou un ensemble d'objets qui partagent une structure et un comportement communs. Les classes ou instances de classes, sont des éléments de modèle communs dans les diagrammes UML.
- Les packages
  Les packages regroupent des éléments de modèle associés de tout type, y compris d'autres packages.
- Les relations
  En langage UML, une relation est une connexion entre des éléments de modèle. Une relation UML est un type d'élément de modèle qui ajoute une sémantique à un modèle en définissant la structure et le comportement entre les éléments de modèle.
- Qualificateurs
  Dans UML, les qualificateurs sont des propriétés applicables aux associations binaires et des composants facultatifs des extrémités d'association. Un qualificateur contient une

liste d'attributs d'association, doté chacun d'un nom et d'un type. Les attributs d'association modélisent les clés utilisées pour indexer un sous-ensemble d'instances de relation.

2. Notation

Une classe comprend un nom (nom de la classe), un/des attribut(s) (un attribut est un propriété élémentaire d'une classe) et un/des méthodes(s) (une opération est une fonction applicable aux objets d'une classe). Le nom de la classe peut être qualifié par un « stéréotype ».

La notation d'une classe est la suivante :

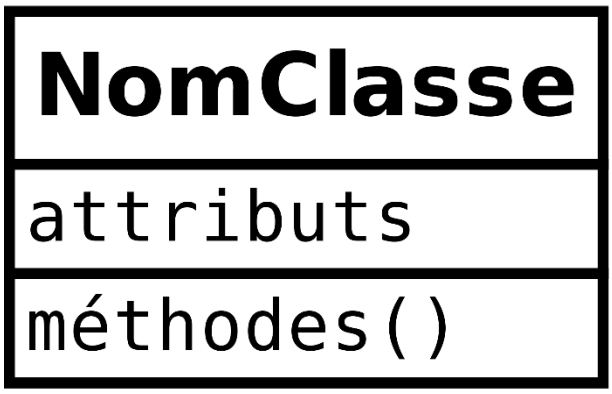


Figure 1:notation d'une classe

On a aussi des classes abstraites, des classes actives et des classe passives.

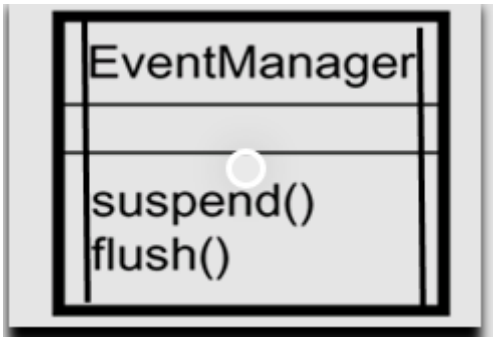


Figure 2:classe utile

Figure 3: classe abstraite

3. Format de description d'un attribut

La description complète des attributs d'une classe comporte un certain nombre de caractéristique qui doivent respecter le formulisme suivant :

visibilité nom : type [multiplicité] = valeur_initiale {propriétés}.

- La visibilité : se reporter aux explication données plus loin sur ce point.
- Nom d'attribut : nom unique dans la classe.
- Type : type primitif (entier, chaine de caractère ..) dépend des types disponibles dans le langage d'implémentation ou type classe matérialisant un lien avec une autre classe.
- Multiplicité : permet de données plus d'information sur les type primitifs comme les tableaux (exemple couleur : Saturation [3], points : Points [2..*]).
- Valeur initiale : valeur facultative données à l'installation d'un objet de la classe.
- Propriétés : valeurs marquées facultative (exemple : « interdit » pour mise à jour interdite)

Un attribut peut avoir des valeurs multiples. Dans ce cas, cette caractéristique est indiquée après le nom de l'attribut (ex. : prénom [3] pour une personne qui peut avoir trois prénoms).

Un attribut dont la valeur peut être calculée à partir d'autres attributs de la classe est un attribut dérivé qui se note « /nom de l'attribut dérivé ».

❖ Relation entre les classes

Un lien est une connexion physique ou conceptuelle entre instances de classes donc entre objets. Une association décrit un groupe de liens ayant une même structure et une même sémantique. Un lien est une instance d'une association. Chaque association peut être identifiée par son nom. Une association entre classes représente les liens qui existent entre les instances de ces classes.

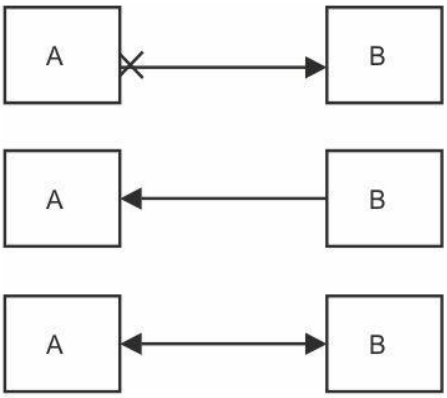
1 La multiplicité.

La multiplicité indique un domaine de valeurs pour préciser le nombre d'instance d'une classe vis-à-vis d'une autre classe pour une association donnée. La multiplicité peut aussi être utilisée pour d'autres usages comme par exemple un attribut multivalué. Le domaine de valeurs est décrit selon plusieurs formes :

- Intervalle fermé – Exemple : 2, 3 ..15.
- Valeurs exactes – Exemple : 3, 5, 8.
- Valeur indéterminée notée * – Exemple : 1..*.
- Dans le cas où l'on utilise seulement *, cela traduit une multiplicité 0..*.
- Dans le cas de multiplicité d'associations, il faut indiquer les valeurs minimale et maximale d'instances d'une classe vis-à-vis d'une instance d'une autre classe.

2 Navigabilité

La navigabilité indique si l'association fonctionne de manière unidirectionnelle ou bidirectionnelle, elle est matérialisée par une ou deux extrémités fléchées. La non-navigabilité se représente par un « X ».



- Navigabilité unidirectionnelle de A vers B.
- Pas de navigabilité de B vers A (non définie explicitement).

- Navigabilité unidirectionnelle de B vers A.
- Navigabilité de A vers B (non définie explicitement).

- Navigabilité bidirectionnelle entre A et B. Habituellement représentée sans flèche.

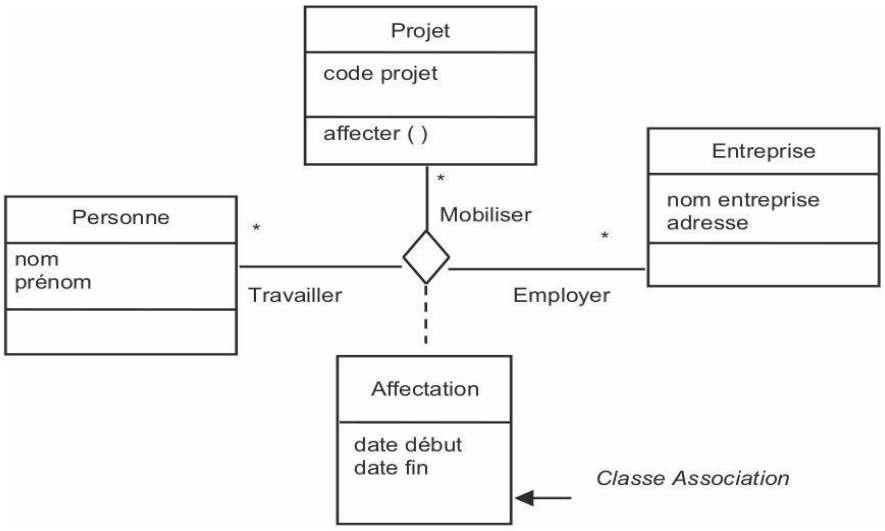
3 Association de dimension supérieure à 2 et classe-association

Une association de dimension supérieure à 2 se représente en utilisant un losange permettant de relier toutes les classes concernées.

Une classe-association permet de décrire soit des attributs soit des opérations propres à l'association. Cette classe-association est elle-même reliée par un trait en pointillé au losange de connexion. Une classe-association peut être reliée à d'autres classes d'un diagramme de classes.

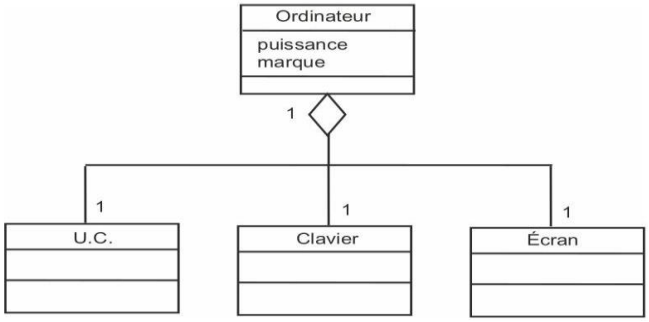
Exemple

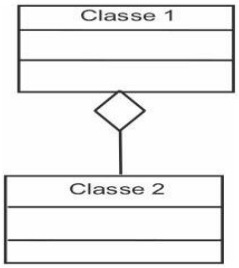
Un exemple d'une association de dimension 3 comprenant une classe-association « Affectation ». La classe-association Affectation permet de décrire les attributs propres à l'association de dimension 3 représentée.



4 Agrégation

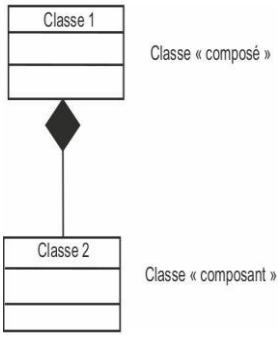
L'agrégation est une association qui permet de représenter un lien de type « ensemble » comprenant des « éléments ». Il s'agit d'une relation entre une classe représentant le niveau« ensemble » et 1 à n classes de niveau « éléments ». L'agrégation représente un lien structurel entre une classe et une ou plusieurs autres classes

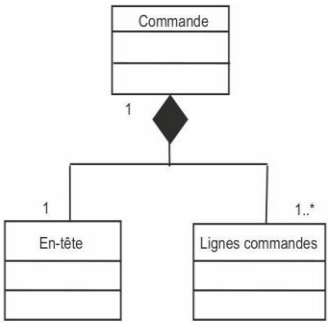


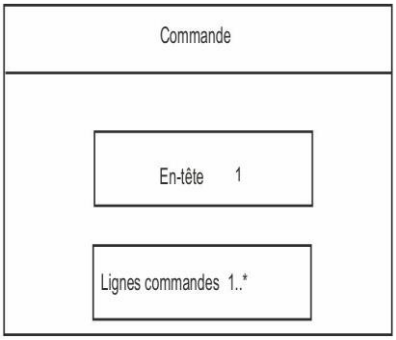


5 Composition

La composition est une relation d'agrégation dans laquelle il existe une contrainte de durée de vie entre la classe « composant » et la ou les classes « composé ». Autrement dit la suppression de la classe « composé » implique la suppression de la ou des classes « composant ».

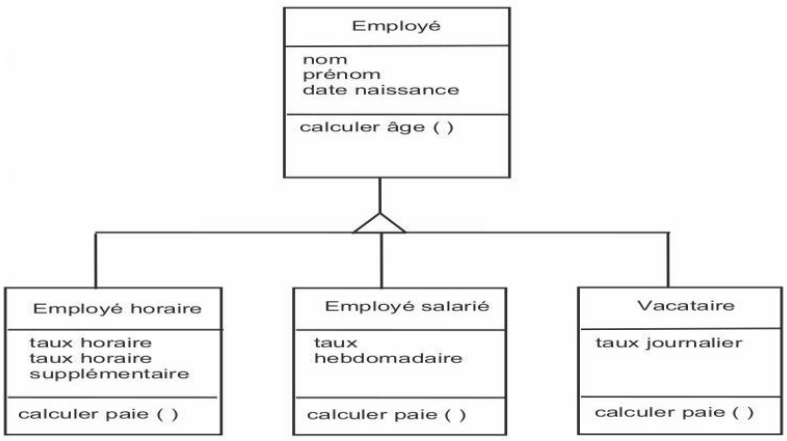






6 L'héritage

L'héritage permet à une sous-classe de disposer des attributs et opérations de la classe dont elle dépend. Un discriminant peut être utilisé pour exploiter le critère de spécialisation entre une classe et ses sous-classes. Le discriminant est simplement indiqué sur le schéma, puisque les valeurs prises par ce discriminant correspondent à chaque sous-classe.



7 Dépendance

La dépendance entre deux classes permet de représenter l'existence d'un lien sémantique. Une classe B est en dépendance de la classe A si des éléments de la classe A sont nécessaires pour construire la classe B.

2) Diagramme d'objet

En modélisation UML, les diagrammes d'objet offrent une image instantanée des instances d'un système et des relations entre ces instances. L'instanciation des éléments de modèle dans un diagramme de classes vous permet d'explorer le comportement d'un système à un point de cohérence. Un diagramme d'objet est composé d'objets (instances de classes) et de lien (instance d'association).

Les diagrammes d'objet sont utiles dans les situations suivantes :

- Pendant la phase d'analyse d'un projet, vous pouvez créer un diagramme de classes pour décrire la structure d'un système puis créer un ensemble de diagrammes d'objets en tant que jeux d'essai pour vérifier si le diagramme de classes est complet et exact.
- Avant de créer un diagramme de classes, vous pouvez créer un diagramme d'objets pour prendre connaissance de certains faits concernant des éléments de modèles particuliers et leurs liens ou pour illustrer des exemples précis des discriminants requis.

Les rubriques suivantes décrivent les éléments compris dans les diagrammes d'objets :

- Spécifications d'instances dans la modélisation UML

  Dans les modèles UML, les spécifications d'instances sont des éléments qui représentent une instance dans le système modélisé. Lorsque vous instanciez un discriminant dans un modèle, vous créez une spécification d'instance qui représente une entité du système modélisé à un point de cohérence, similaire à image instantanée de l'entité. Vous pouvez modéliser les changements apportés à l'entité en créant plusieurs spécifications d'instances, une pour chaque image instantanée.

- Relations de lien dans la modélisation UML

  Dans les diagrammes UML, une relation de lien est une instance d'une association ou du chemin d'une communication. Alors qu'une association est une relation entre deux discriminants, un lien est une relation entre des objets, des instances de discriminants ou des nœuds.

- Relations de dépendance

  Dans la modélisation UML, une relation de dépendance est une relation dans laquelle un élément, le client, tire parti ou dépend d'un autre élément de modèle, le fournisseur. Vous pouvez utiliser des relations de dépendance dans les diagrammes de classes, les diagrammes de composants, les diagrammes de déploiement et les diagrammes de cas d'utilisation pour indiquer qu'un changement apporté au fournisseur pourrait nécessiter un changement dans le client.

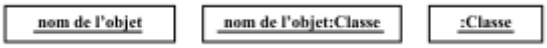
- Relations de déploiement

Dans la modélisation UML, les relations de déploiement indiquent qu'un type de nœud particulier prend en charge le déploiement d'un type d'artefact.

1. Notation

Un objet est une classe : il représente ‘‘l’état ’’ d’une classe à un instant précis. Un objet peut se représenter de 3 différentes manières :

- Nom de l’objet (souligné)
- Non de la classe(souligné)
- Nom de l’objet : nom de la classe (souligné)

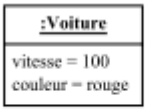


Il est possible aussi de faire une représentation d’un groupe d’objet d’instance d’une même classe.

L’état d’un objet est déterminé par les valeur de ses attributs. Il est possible de nommer un état afin d’indiquer clairement dans quel état se trouve un objet.

La représentation des objets peut contenir des attributs significatifs.

Représentation de l’état d’un objet



Représentation des attributs significatifs

2. Associations

Les objets sont reliés par des instances d’association : les liens. Un lien est une relation entre objet à une instante donnée. La multiplicité des extrémités des liens est toujours de 1.

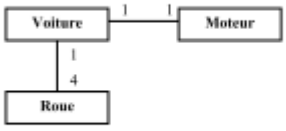


Diagramme de classes

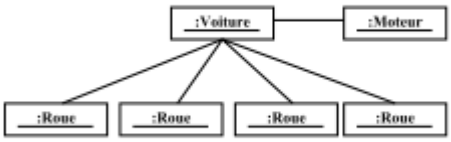
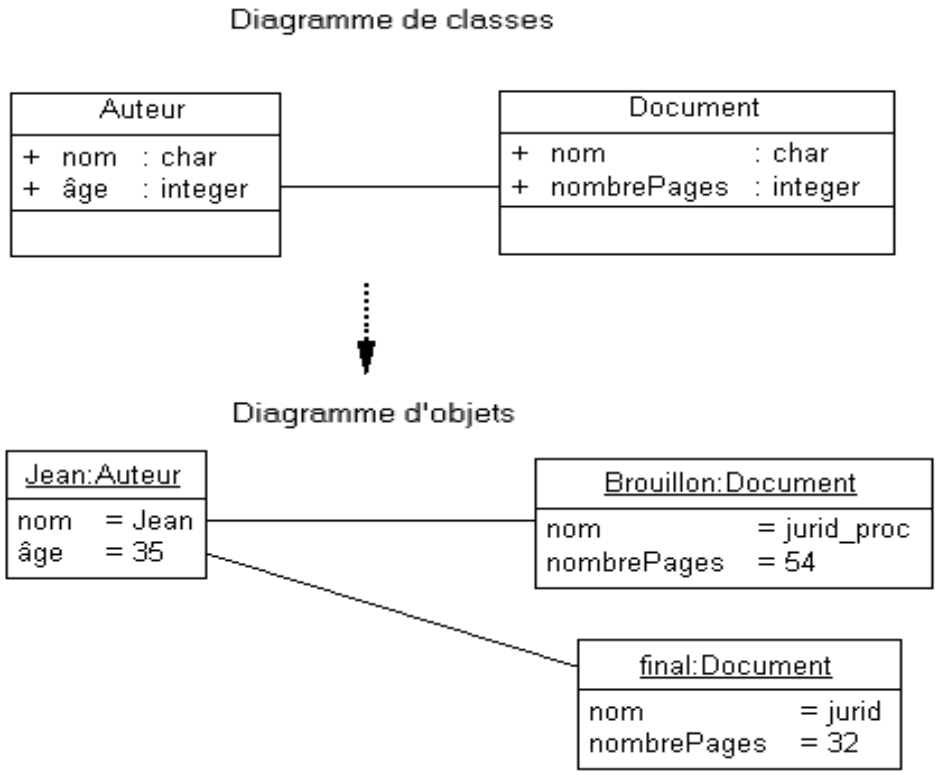


Diagramme d’objets

Les stéréotypes peuvent être utilisé pour qualifier les extrémités des liens comme « local », « global », « paramètre », « self ».

3. Diagramme de classe à diagramme d'objet



3) Diagramme de structure composite

Un diagramme de structure composite est un diagramme UML qui joue le même rôle qu'un diagramme de classes, mais permet d'approfondir la description de la structure interne de plusieurs classes, à l'intérieur de celles-ci des instances collaborent par le biais de liens de communication afin de parachever des objectifs communs.

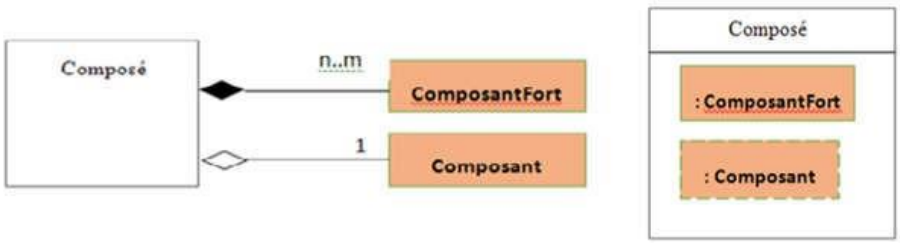
Le diagramme de structure composite permet d'afficher :

- La structure interne d'un classificateur
- Les interactions avec l'environnement par le biais des ports
- Un comportement d'une collaboration

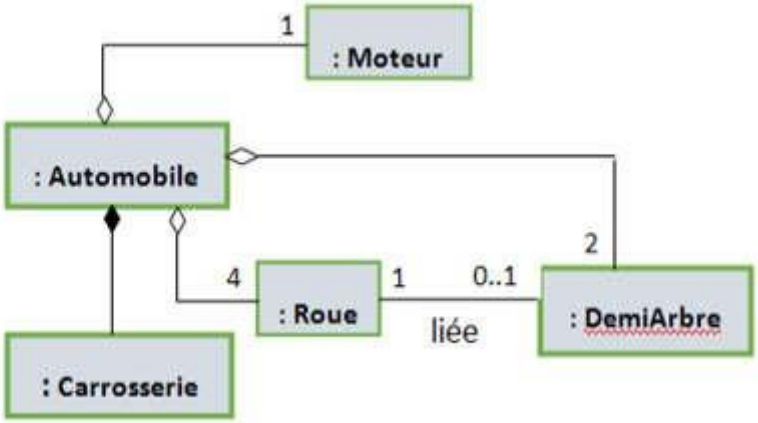
1. Notation

Le diagramme de structure composite ne remplace pas un diagramme de classe mais le complète. Dans le diagramme de structure composite l'objet composé est décrit par un classifieur tans dit que ses composants sont décrits par des parties.

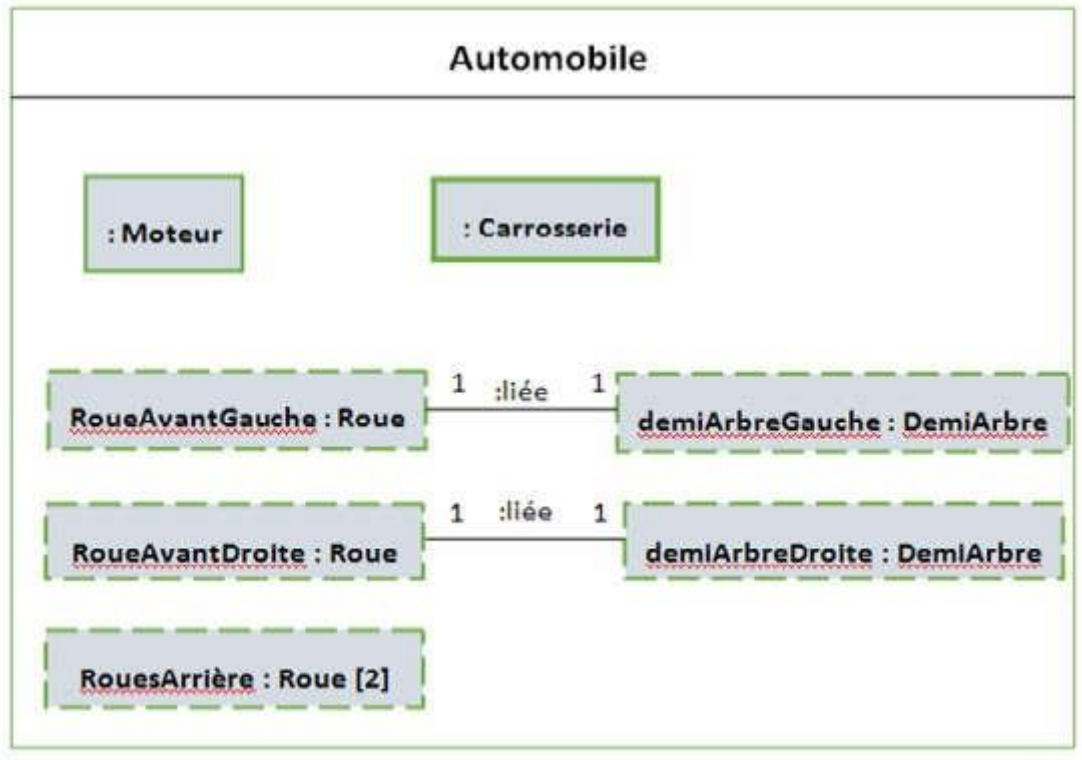
L'objet composé décrit par le diagramme de classe possède un composant issu d'une composition forte et un autre issu d'une agrégation. Dans le diagramme de structure composite les composants sont intégré au sein du classifieur qui décrit l'objet composé, la cardinalité est indiquée entre crochets. Un composant issue d'une agrégation est représenté par une ligne en points tillés, un composant issu d'une composition forte est représentée par une ligne continue.



2. Diagramme de classe-diagramme de structure composite



Ce diagramme de classes illustre une automobile en tant qu'objet composé il introduit l'association liée entre les roues et les demi arbres qui assurent la transmission entre le moteur et les roues avant qui sont les roues motrices. La cardinalité de l'association liée est 0...1 un pour les roues avant et zéro pour les roues arrière. Cette information ne peut pas être introduite dans le diagramme de classes à moins d'introduire deux sous classes de roue : roue avant et roue arrière mais cela alourdira le diagramme de classes.



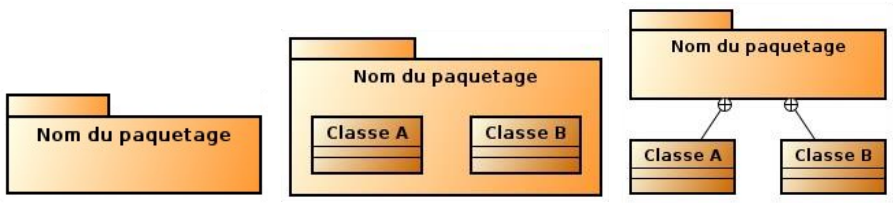
4) Diagramme de paquetage

Lorsque nous sommes en présence d'un système de grande taille, il peut être intéressant de le décomposer en plusieurs parties (appelées paquetage).

Un paquetage est donc un regroupement de différents éléments d'un système (regroupement de classes, diagrammes, fonctions, interfaces…). Cela permet de clarifier le modèle en l'organisant. Le diagramme de paquetages représente les paquetages (ou espaces de noms) composant un système, ainsi que les relations qui lient ces différents paquetages.

1. Notation

Il est représenté par un dossier avec son nom à l'intérieur. Il est possible de représenter les éléments du système appartenant au paquetage à l'intérieur de celui-ci ou à l'extérieur de celui-ci.

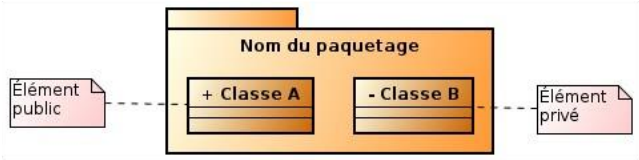


2. Dépendances entre paquetages

❖ Visibilité

Chaque éléments d'un paquetage est soit :

- privé, c'est-à-dire encapsulé dans le paquetage et invisible à l'extérieur de celui-ci. Un élément privé est désigné par un signe – devant lui.
- public, c'est-à-dire visible et accessible de l'extérieur du paquetage. Un élément public est désigné par un signe + devant lui.



Par défaut, les éléments d'un paquetage sont publics.

❖ Dépendance de type « *import* »

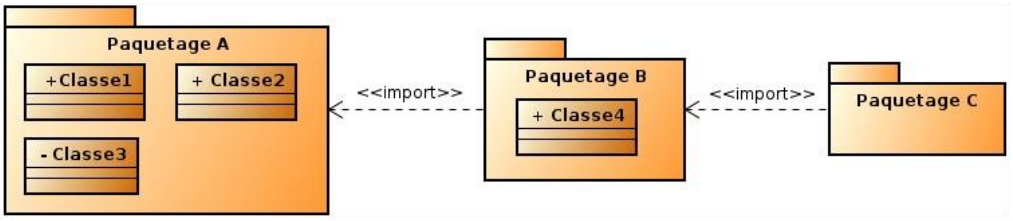
Elle correspond à l'importation par un paquetage ***B*** de tous les éléments *publics* d'un paquetage ***A*.**

Ces éléments :

- auront la visibilité *« public »* dans le paquetage ***B*** (et seraient donc aussi transmis à un paquetage *C* qui ferait une importation du paquetage ***B***).

- seront accessibles au paquetage ***B*** sans avoir à utiliser explicitement le nom du paquetage ***A***.

La dépendance de type « *import* » est représentée par une flèche pointillée muni du stéréotype *<<import>>*.



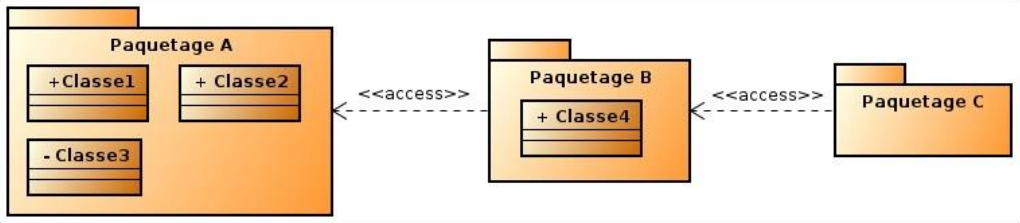
Le paquetage B importe Classe1 et Classe2 (pas Classe3 qui a une visibilité de type privée).

Classe1 et Classe2 ont une visibilité de type public dans paquetage B. Le paquetage C importe Classe1, Classe2 et Classe4.

❖ Dépendance de type « *access*»

Elle correspond à l'accès par un paquetage B de tous les éléments publics d'un paquetage A. Ces éléments auront la visibilité *privée* dans le paquetage B, ils ne peuvent donc pas être transmis à un paquetage C qui ferait une importation ou un accès au paquetage B (pas de transitivité).

La dépendance de type *« access »* est représentée par une flèche pointillée muni du stéréotype *<<access>>*.



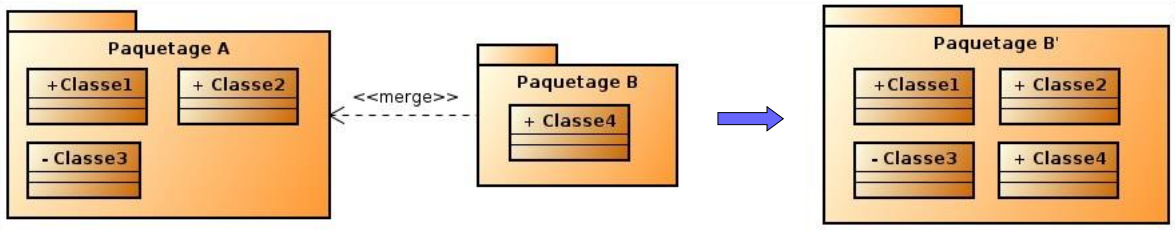
Le paquetage B a accès à Classe1 et Classe2 (pas à Classe3 qui a une visibilité de type privée).

Classe1 et Classe2 ont une visibilité de type privé dans paquetage B. Le paquetage C a accès à Classe4 (pas à Classe1 et Classe2 qui ont une visibilité de type privée dans paquetage B).

❖ Dépendances de type « *merge* »

Elle correspond à la fusion de 2 paquetages en un seul.

La dépendance de type « *merge* » est représentée par une flèche pointillée muni du stéréotype



Le paquetage A est fusionné dans le paquetage B (le paquetage A n'est pas modifié alors que le paquetage B est écrasé pour accueillir la fusion des 2 paquetages).