DOCUMENTATION SUR UML

INTRODUCTION

Un système d'information est un ensemble de ressource organisé permettant de :stocker, de structurer de manière efficace et coherante , d'acquerrir et communiquer des informations sous forme de testes, des images , sons ou des données codé dans une organisation

II) NOTION D'ANALYSE ET DE CONCEPTION

1) Merise

Merise est une methode d'analyse et de conception des donnees qui prend ses souches dans l'approche de decomposition fonctionnelle. L'approche de decomposition fonctionnelle separe les donnee et les traitements.

Son principal avantage est sa simplicite et son inconvénient est une maintenance évolutive.

2) UML

UML est une methode d'analyse et de conception qui prend sa souche dans l'approche orientee objet. L'approche oriente objet se base sur une structure commune : la notion de classe. La classe regroupe les donnees et les traitements

III) UML

APPROCHE

La description de la programmation orientee object fait ressortir l'etendu du travail conceptuel : definition des classe , de leur relation, des attributs et des méthodes , etc . pour programmer une application il ne convient pas de se lancer directement dans l'ecriture du code . il faut d'abord organiser les idée , les documenter puis organiser la realisation en definissant les modules et étapes de la réalisation .

Nee de la convergence de 3 methodes UML (Unified Modeling Language) n'est pas une methode car ces acteurs ont estimé qu'il n'était pas opportun de le définir comme une methode en raison de la diversité de ses particularites, donc UML est defini comme un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système d'information.

2) LES 13 DIAGRAMMES DE UML

UML possede différents types de diagramme representant autant de vues distinct(UML 2.0). Ils sont repartis en 3groupes :

- **Diagrammes UML structurels** (
  - Diagramme de classe,
  - Diagramme de composants,

- Diagramme de deploiement,
- Diagramme d'objet,
- Diagramme de paquetages
- Diagramme de structure composite);

- **Diagrammes UML comportementaux(**
  - Diagramme d'activites,
  - Diagramme de cas d'utilisation,
  - Diagramme d'etat-transmission
- **Diagrammes d'interaction**
  - Diagramme d'interaction,
  - Diagramme de sequence,
  - Diagramme de temps,
  - Diagramme de communication);

III) ETUDE DETAILLE DES DIAGRAMMES D'UML

1) Diagramme de Use Case ou cas d'utilisation
   a) Objectif

Bien souvent, la maîtrise d'ouvrage et les utilisateurs ne sont pas des informaticiens. Il leur faut donc un moyen simple d'exprimer leurs besoins. C'est précisément le rôle des diagrammes de cas d'utilisation qui permettent de recueillir, d'analyser et d'organiser les besoins, et de recenser les grandes fonctionnalités d'un système. Il s'agit donc de la première étape UML d'analyse d'un système.

Un diagramme de cas d'utilisation capture le comportement d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit. Il scinde la fonctionnalité du système en unités cohérentes, les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système, ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique

   b) acteur

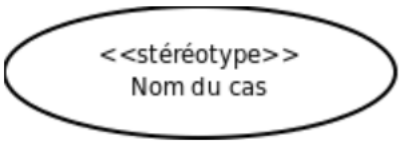
un acteur est tout ce qui interagit avec le système et qui est exterieur au système. Un acteur est representé en Uml par un petit bon homme

   c) Notion de use case

Un cas d'utilisation est une unité cohérente représentant une fonctionnalité visible de l'extérieur. Il réalise un service de bout en bout, avec un déclenchement, un déroulement

et une fin, pour l'acteur qui l'initie. Un cas d'utilisation modélise donc un service rendu par le système, sans imposer le mode de réalisation de ce service.

Un cas d'utilisation se représente par une ellipse (figure 2.3) contenant le nom du cas (un verbe à l'infinitif), et optionnellement, au-dessus du nom, un stéréotype



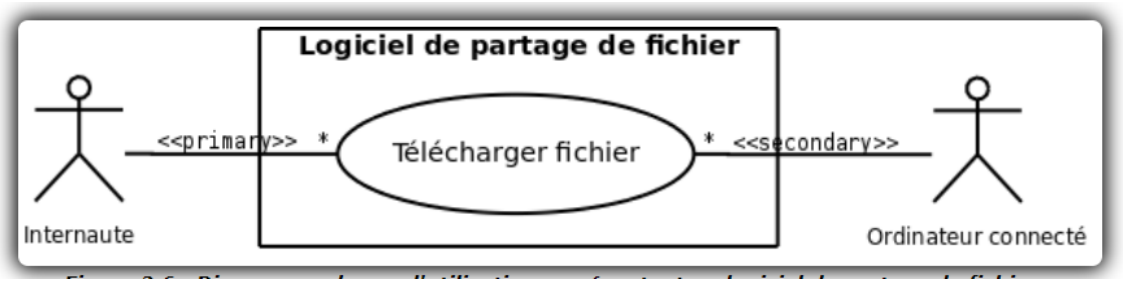
d) Type de cas d'utilisation

Il existe principalement deux types de relations :les dépendances stéréotypées, qui sont explicitées par un stéréotype (les plus utilisés sont l'inclusion et l'extension) ;et la généralisation/spécialisation.

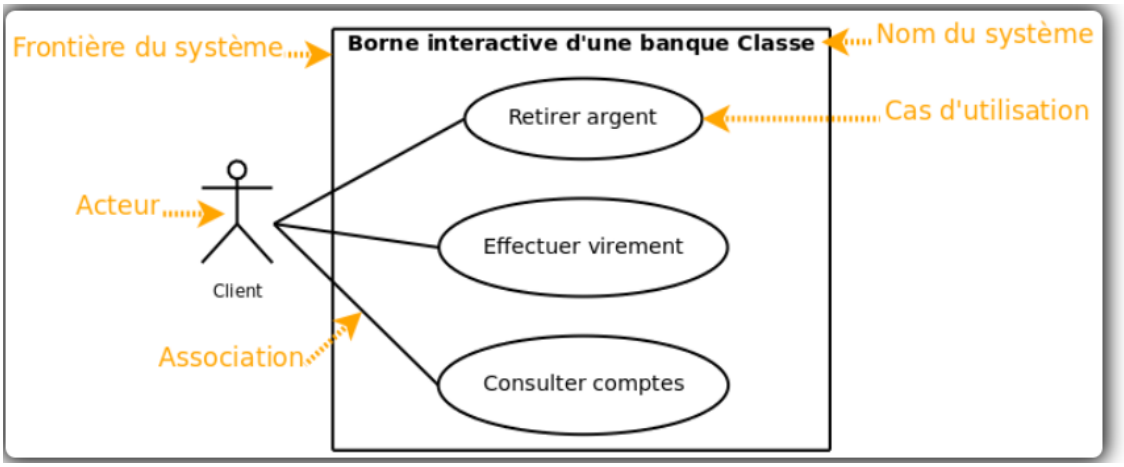
Une dépendance se représente par une flèche avec un trait pointillé . Si le cas A inclut ou étend le cas B, la flèche est dirigée de A vers B.

Le symbole utilisé pour la généralisation est un flèche avec un trait plein dont la pointe est un triangle fermé désignant le cas le plus général.

e) Les liens ou relations entre cas d'utilisation



f) Formalisme d'un diagramme de cas d'utilisation



g) Description textuel d'un diagramme de cas d'utilisation

2) Diagramme de classe

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.

a) Objectif

Alors que 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 pour réaliser les cas d'utilisation. Il est important de noter qu'un même objet peut très bien intervenir dans la réalisation de plusieurs cas d'utilisation. Les cas d'utilisation ne réalisent donc pas une partition(7) des classes du diagramme de classes. Un diagramme de classes n'est donc pas adapté (sauf cas particulier) pour détailler, décomposer, ou illustrer la réalisation d'un cas d'utilisation particulier.

Il s'agit d'une vue statique, car on ne tient pas compte du facteur temporel dans le comportement du système. Le diagramme de classes modélise les concepts du domaine d'application ainsi que les concepts internes créés de toutes pièces dans le cadre de l'implémentation d'une application. Chaque langage de Programmation orienté objet donne un moyen spécifique d'implémenter le paradigme objet (pointeurs ou pas, héritage multiple ou pas, etc.), mais le diagramme de classes permet de modéliser les classes du système et leurs relations indépendamment d'un langage de programmation particulier.

b) Notion de classe et d'objet

Une *instance* est une concrétisation d'un concept abstrait. Par exemple :

- la Ferrari *Enzo* qui se trouve dans votre garage est une instance du concept abstrait *Automobile ;*
- l'amitié qui lie Jean et Marie est une instance du concept abstrait *Amitié ;*

Une classe est un concept abstrait représentant des éléments variés comme :

- des éléments concrets (ex. : des avions),
- des éléments abstraits (ex. : des commandes de marchandises ou services),

Tout système orienté objet est organisé autour des classes. Une classe est la description formelle d'un ensemble d'objets ayant une sémantique et des caractéristiques communes.

Un objet est une instance d'une classe. C'est une entité discrète dotée d'une identité, d'un état et d'un comportement que l'on peut invoquer. Les objets sont des éléments individuels d'un système en cours d'exécution.

c) Attributs

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

<visibilité> [/] <nom_attribut> :
<type> ['['<multiplicité>']' [{<contrainte>}]] [= <valeur_par_défaut>]

d) Méthodes

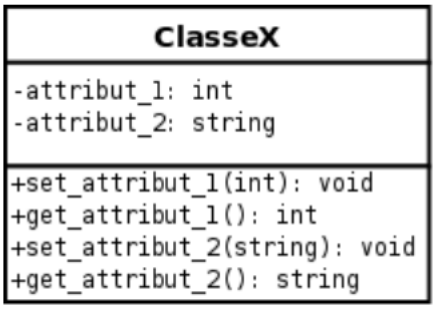
Dans une classe, une opération (même nom et mêmes types de paramètres) doit être unique. Quand le nom d'une opération apparaît plusieurs fois avec des paramètres différents, on dit que l'opération est surchargée. En revanche, il est impossible que deux opérations ne se distinguent que par leur valeur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

La déclaration d'une opération contient les types des paramètres et le type de la valeur de retour, sa syntaxe est la suivante :

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

e) Encapsulation

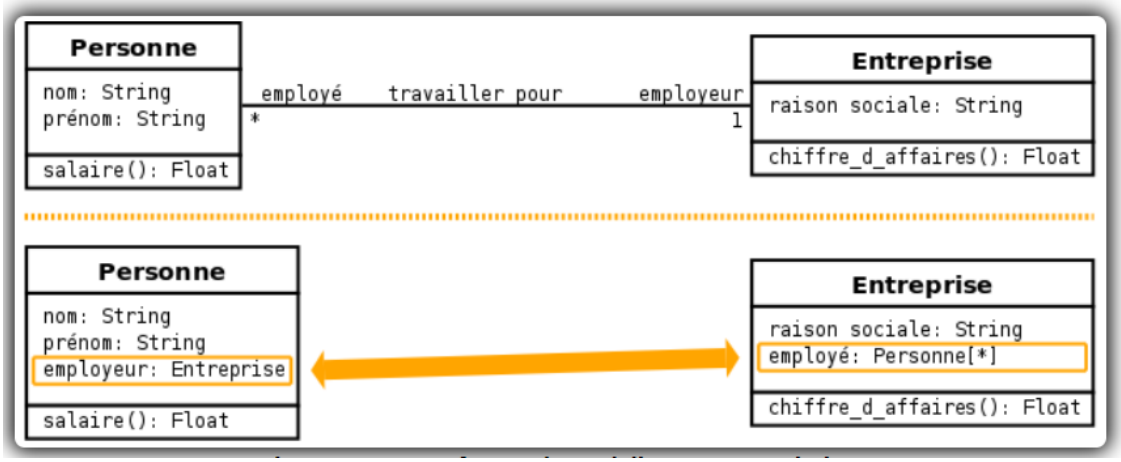
L'encapsulation permet de définir des niveaux de visibilité des éléments d'un conteneur. La visibilité déclare la possibilité pour un élément de modélisation de référencer un élément qui se trouve dans un espace de noms différent de celui de l'élément qui établit la référence. Elle fait partie de la relation entre un élément et le conteneur qui l'héberge, ce dernier pouvant être un paquetage, une classe ou un autre espace de noms. Il existe quatre visibilités prédéfinies: ***Public*** **ou +,** ***Protected*** **ou # ,** ***Private*** **ou - ,** ***Package*** **ou ~ ou rien**



f) Surcharge

g) Relation entre classe

Une association est une relation entre deux classes (association binaire) ou plus (association n-aire), qui décrit les connexions structurelles entre leurs instances. Une association indique donc qu'il peut y avoir des liens entre des instances des classes associées.



Il existe différents types de relation : l'association ; la classe d'association ; l'association n-aire ; la navigabilite d'une association ; l'agregation ; la généralisation.

3) Diagramme de sequence
   a) Objectif

Le **diagramme de séquence** permet de montrer les interactions d'objets dans le cadre d'un scénario d'un **Diagramme** des cas d'utilisation. ... Le but étant de décrire comment se déroulent les actions entre les acteurs ou objets.

   b) Ligne de vie

Une ligne de vie se représente par un rectangle, auquel est accroché une ligne verticale pointillée, contenant une étiquette dont la syntaxe est :

[<nom_du_rôle>] : [<Nom_du_type>]

   c) Message

Un message définit une communication particulière entre des lignes de vie. Plusieurs types de messages existent, les plus communs sont :

- l'envoi d'un signal ;
- l'invocation d'une opération ;
- la création ou la destruction d'une instance.
  - Message asynchrone

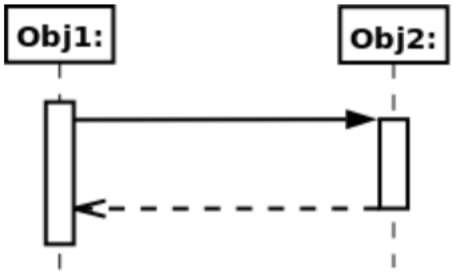
Une interruption ou un événement sont de bons exemples de signaux. Ils n'attendent pas de réponse et ne bloquent pas l'émetteur qui ne sait pas si le message arrivera à destination, le cas échéant quand il arrivera et s'il sera traité par le destinataire. Un signal est, par définition, un message asynchrone.

Graphiquement, un message asynchrone se représente par une flèche en traits pleins et à l'extrémité ouverte partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible

- Message synchrone

L'invocation d'une opération est le type de message le plus utilisé en programmation objet. L'invocation peut être asynchrone ou synchrone. Dans la pratique, la plupart des invocations sont synchrones, l'émetteur reste alors bloqué le temps que dure l'invocation de l'opération.

Graphiquement, un message synchrone se représente par une flèche en traits pleins et à l'extrémité pleine partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible (figure 7.8). Ce message peut être suivi d'une réponse qui se représente par une flèche en pointill



4) Diagramme de composant
5) Diagramme de deploiement