Conception et modélisation orientée objet

I- Introduction

Historique

1970 : première méthode de développement (analyse structurée de yourdon 1970)
1988 : première méthode orientée-objet : Schlaer/Mellor (OOSA)
1991: coad/yourd (OOA), Booch (OOD), Rumbaugh (OMT), etc.
1992:Jacobson (Objectory), et de nombreuses autres (plus de 50)
1994 : tentative d'unification : Coleman (Fusion)
1995 : Booch+Rumbaugh, puis Jacobson (1995) : UML 0.8 (1995)
1996 : Appel d'offre de l'OMG pour une approche normalisée pour les développements objets
1997 : Choix et normalisation de UML 1.0 (actuellement 2.0)

Qu'est ce qu'un modèle

Un modèle est simplification de la réalité
On construit des modèles pour mieux comprendre le système en cours de développement
Pour un observateur A, M est un modèle de l'objet O, si M aide A à répondre aux questions qu'il pose sur O (Minsky)
On construit des modèles pour les systèmes complexes car on ne peut pas comprendre de tels systèmes dans leur ensemble.
Chaque modèle peut être décrit à différents niveaux de précision

Qu'est ce qu'UML ?

Un langage de modélisation visuel ;
Un langage pour spécifier, visualiser, construire et documenter les produits d'un développement logiciel ;
Une notation et une sémantique ;
Des diagrammes pour modéliser les aspects statiques, dynamiques et organisationnels ;
Une unification des connaissances acquises dans le domaine de l'orienté objet ;
Compatible avec toutes les méthodes de développement ;
UML n'est pas langage de programmation.

Pourquoi le terme « unifié » ?

UML se veut une notation unifiée par rapport ;
Aux méthodes et notation déjà proposées ;
Aux différentes phases du cycle de développement d'un logiciel ;
Aux domaines d'application ;
Aux langages et plateformes d'implantation ;
Aux différents procédés de développements ;

II- Concepts généraux de l'orienté objet

Actuellement la modélisation des systèmes, de plus en plus complexes, pose d'énormes problèmes de cohérence, de fiabilité, d'efficacité et de réutilisation.
Le génie logiciel et les outils de modélisation qu'il apporte, notamment avec l'approche orientée objet, a pour objectif de maîtriser la conception et le développement de ces systèmes complexes.

Paradigme objet

A fin de résoudre au mieux la problématique posée par la modélisation des systèmes complexes.
Une solution serait d'introduire, après la programmation numérique et après la programmation symboles, mais des structures complexes capables de représenter une entité du monde réel.
Ce sont ces structures qu'on appelle objets.

Regrouper des moyens de description et de manipulation de l'information au sein d'une même entité.

- ❖ Conception : l'identification de l'objet et de son interface avec les autres objets sera fondamentale.
- ❖ Réalisation : un objet pourra être implémenté de manière indépendante en respectant ses interfaces spécifiées.
- ❖ Un système logiciel développé selon l'approche orienté objet : un ensemble d'objets communiquant par message, chaque objet étant responsable de l'accomplissement de certaines tâches qui dépendent de son état.

Modélisation de système basée sur des concepts bien adaptés à cette activité :

⇒ L'objet : état+ comportements
⇒ L'encapsulation : interface visible +implémentation masquée
⇒ La communication entre objets : seul moyen l'envoi de message
⇒ L'abstraction : regrouper et hiérarchiser
⇒ La classe : famille d'objets similaires (attributs+méthodes)

Statiquement un système logiciel et un ensemble hiérarchisé de classes dont l'une au moins est appelée « racine » (ou classe d'entrée)

⇒ L'instance : création dynamique d'un objet exemplaire d'une classe (une classe abstraite et non instanciable)

La relation client /fournisseur : une classe offre des service à une autre (association+agrégation)

⇒ L'héritage : réutilisation « automatique » dans une classe (fille) des ressources d'une autre classe (parentes) (héritage simple et multiple)

⇒ Le polymorphisme : un nom peut désignes des objets de classe différentes à l'exécution ; tout objets désigné par ce nom est capable de répondre à sa façon à un ensemble commun de messages

⇒ Principe de la substitution : la valeur d'un objet d'un type donné peut être assignée à un objet d'un type ancêtre et non

L'inverse : une opération d'une classe donnée peut être redéfinie dans les classe descendantes

LA RELATION CLIENT DE (délégation)

Une classe A est cliente de classe B quand elle définit une entité de type B, sous l'une des formes suivantes :

Un attribut (variables statique)
La valeur de retour d'une fonction
Un argument d'opération
Une variable locale

Exemple :

```
Classe CLIENT
        Fournisseurs Frs ;
        String nom ;
        Acheter_produit (PRODUIT prdt : fournisseurs f) ;
        Entrer_caractéristiques () {
                        IDENTIFICATION id ;
                }
}
```

Cela induit une relation de dépendance car toute modification de la spécification de B peut entraîner une modification de A

La relation de dépendance entre classes exprime une dépendance potentielle entre les instances.

1- La relation d'association

Définition :

Une classe A est associée avec une classe B si A est cliente de B et si les instances de B sont indépendantes de celles de A ; une instance particulière de B peut appartenir à plusieurs associations donc être partagée.

Exemple :

```
Class AVION
        // Attributs de communication
        TOUR_CONTROLE contrôle ;
        Atterrir () {
                Contrôle demander_piste ;
                If (contrôle autorisation_atterissage)
        }
}
```

2- La relation de composition

Définition :

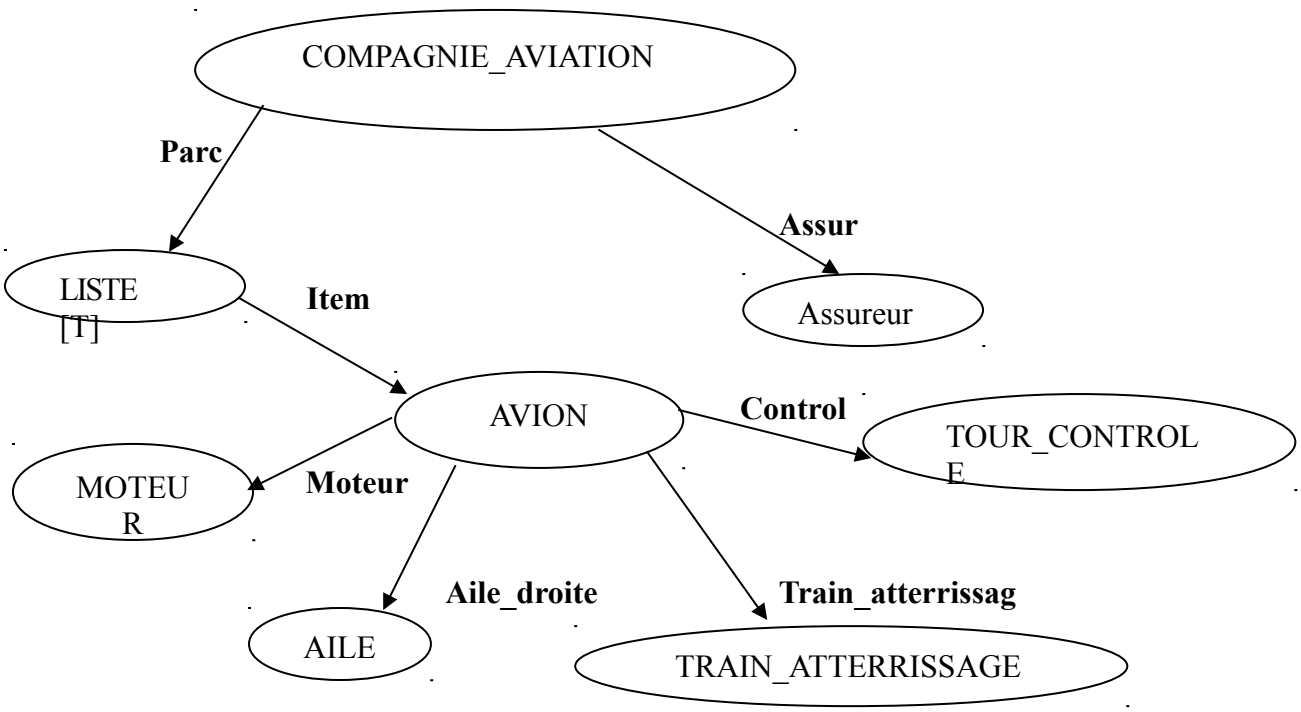
A est une relation d'agrégation avec B si A est cliente de B et si B est attachée à A comme un composant, l'instance de B ne peut être fournisseur d'autres instances de classe.

Exemple :

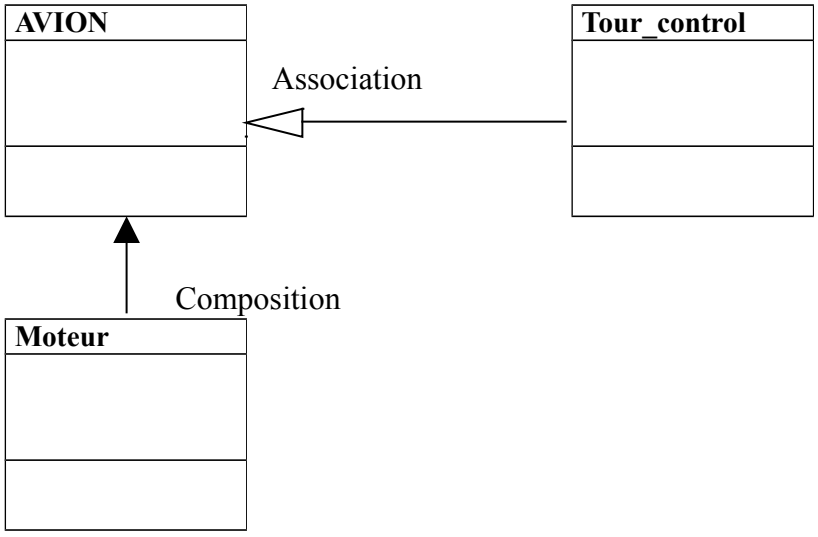
```
Class AVION
        // Attributs de communication
        TOUR_CONTROLE contrôle ;
        // Attributs de composition
        AILE aile_droits, aile_gauche ;
        MOTEUR_AVION moteur ;
        //méthodes
        Atterrir () {
                Contrôle demander_piste ;
                If (contrôle autorisation_atterissage)
                Train_atterrissage sortir ;
                Aile_droite .lever_volets ;
                Aile_droite .lever_volets ;
                Moteur.reduire_gaz ;


        }
}
```

RELATRION DE CLIENTELISME : REPRESENTATION GRAPHIQUE

On peut « oublier » les relations de dépendance secondaires (déclaration d'un attribut local ou privé. …..)



Présentation du diagramme de classe en utilisant UML

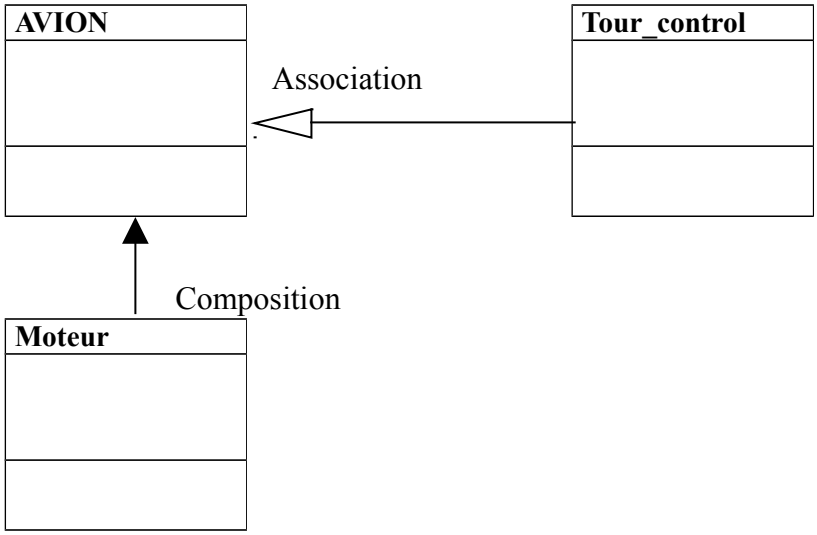


Exercice

Détermine la relation entre les classes suivantes :
Livre, couverture, paragraphe.
Présenter le diagramme de classe en utilisant UML.

Correction :

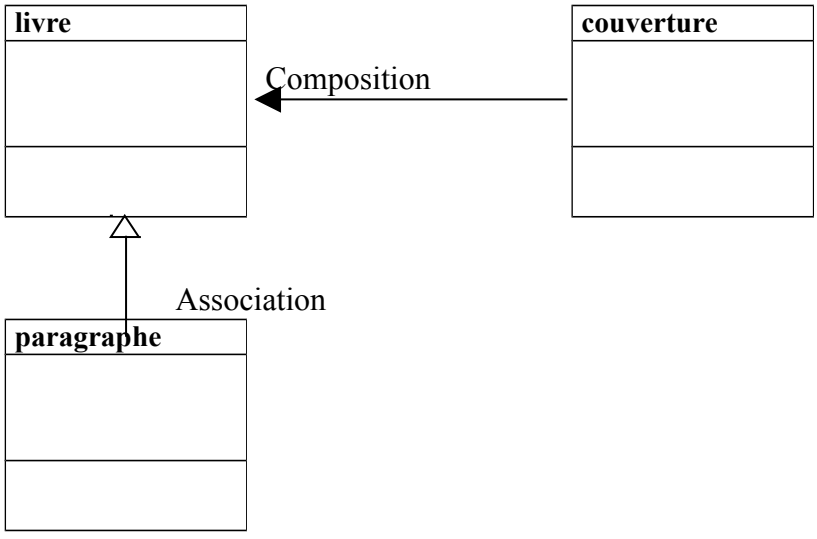
Présentation du diagramme de classe en utilisant UML



Exercice

Détermine la relation entre les classes suivantes :
Livre, couverture, paragraphe.
Présenter le diagramme de classe en utilisant UML.

Correction :

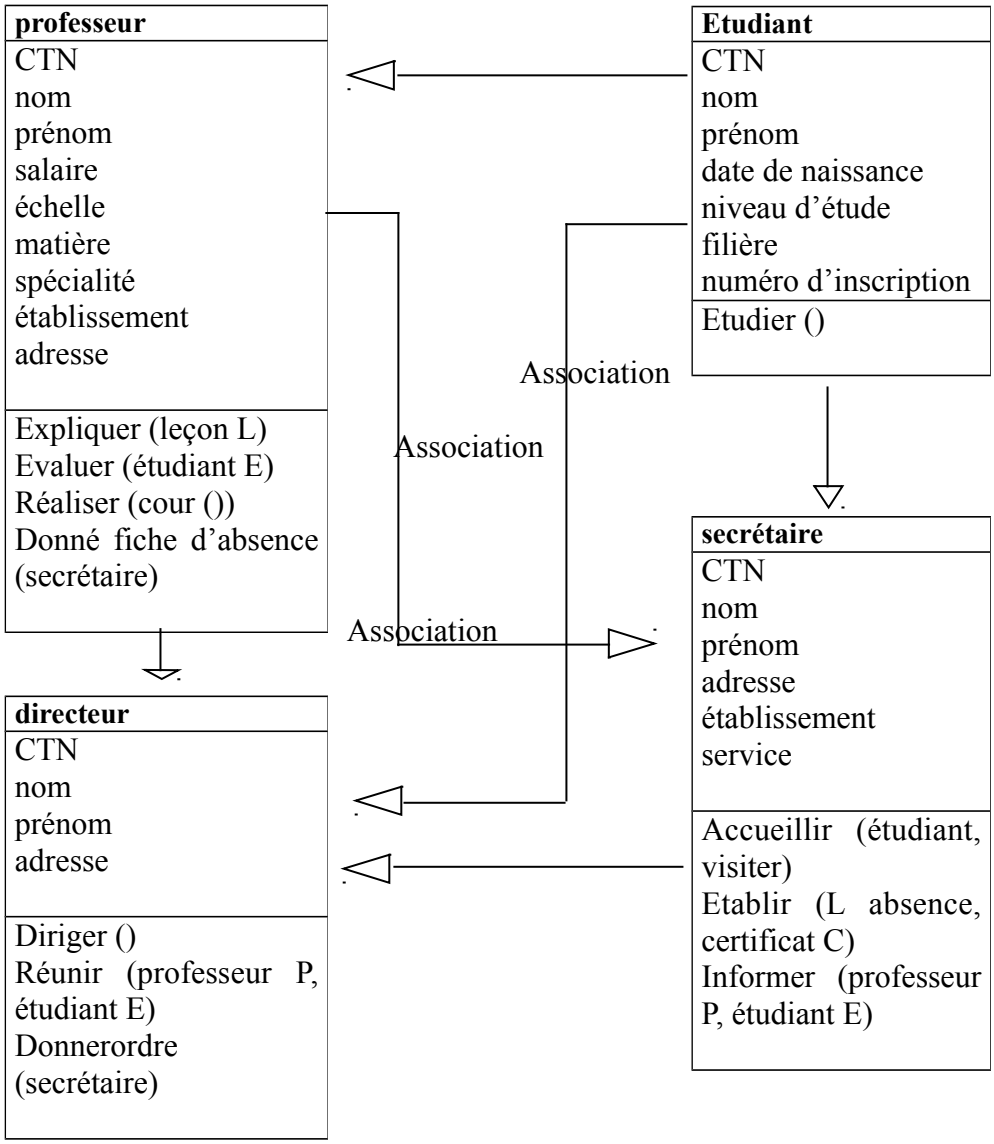


Exercice

On considère les classes suivantes :
Professeur, étudiant, secrétaire

1- déterminer les attributs et les comportements
2- déterminer les relations entre les classes

Correction



III- LE POLYMOPHISME

Idée de base :

- Pouvoir faire exécuter un message par un objet dont le type varie dynamiquement
- Le système doit déterminer dynamiquement l'implantation de l'opération à exécuter

Formalisation : définition d'entités flexibles pouvant référencer des objets de forme variable à l'exécution (d'où **polymorphisme)**
Exemple figure employer
Mise en œuvre : exploitation de l'héritage

- La valeur d'un objet d'un type donné peut être assignée à un objet d'un type ancêtre, pas l'inverse

- Redéfinition d'opérations dans les classes filles

Intérêt :
- Flexibilité sans perte du typage
- Généraliser

IV- DEVELOPPEMENT PAR OBJETS

- Liaison statique et liaison dynamique= la correspondance entre le message et cette l'opération à exécuter se fait en construisant une liaison entre le message et cette opération (à la compilation=liaison statique ; à l'exécution= liaison dynamique).
- Généricité= paramètre des classe par des types (classe générique)
- Assertion sémantiques= établissement de contrats entre une classe ses classe clientes. Chacune ayant des droits et obligations (pré-conditions et de post-conditions portant sur des opérations, et d'invariants de classe)
- Exceptions= lorsqu'un contrat échoue, une exception est générée et le mécanisme de gestion des exceptions permet de récupérer l'erreur, de la traiter et évite au programme de s'arrêter sur celle-ci (tolérance aux fautes et aux pannes).

V- OBJECTIFS QUALITE D'UN DEVELOPPEMENT PAR OBJETS

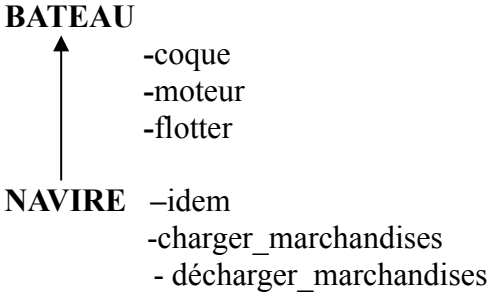
Qualités essentielles d'un développement de logiciel :
- Homogénéité
- Simplicité
- Réversibilité/ traçabilité
- Maintenabilité/ extensibilité
- Fiabilité/ contractualisation
- Rèutilisabilité

VI- L'HERITAGE

But : préciser, dans la phase conceptuelle, la sémantique des relations d'héritage
Intérêt : valider l'analyse/conception

1-Héritage pour spécialiser

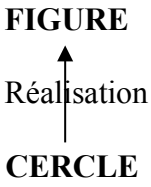
Approche descendante : spécialisation
Approche ascendante : généralisation
Définition : A est spécialisée par B si B hérite de A et offre des caractéristiques supplémentaires par rapport à A
Exemple :



2-Hériter pour réaliser une classe

Virtuelle

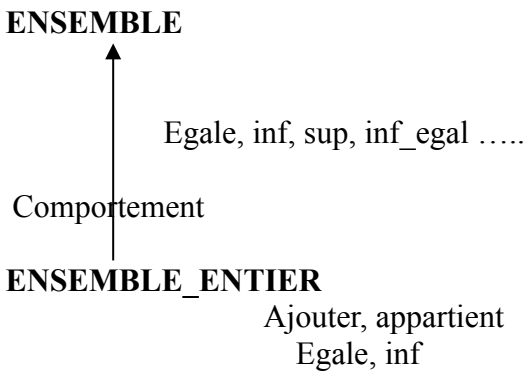
Définition : une classe virtuelle A est réalisée (concrétisée) par B si B décrit une implantation particulière de A
Exemple :



3-hériter d'un comportement (d'une propriété)

Définition : une classe B se comporte comme une classe A si B hérite de A et définit un comportement particulière de A, la classe A est virtuelle, B concrète ou virtuelle.

Exemple :

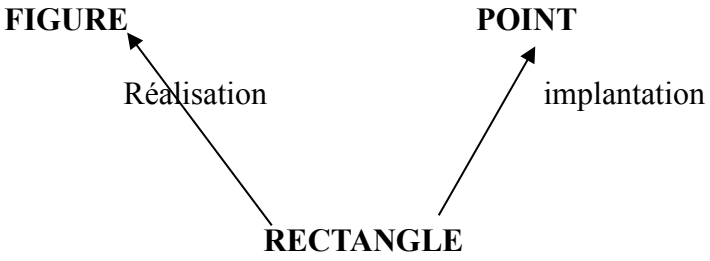


4-Hériter pour implanter (« incidental inhéritance »)

Définition : une classe B est implantée par une classe A si B hérite de A et si les caractéristiques de Ane sont utilisées que de manière interne dans B.

A est une classe concrète.

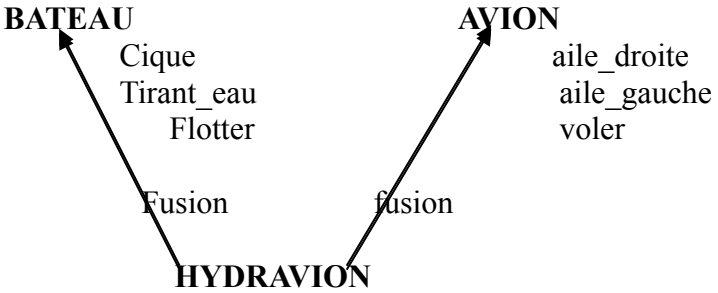
Exemple :



5- Hériter pour fusionner :

Définition : C fusionne 2 classes A et B si C hérite de A et B et si les 2 liens peuvent etre vus comme des liens de spécialisation indépendants.

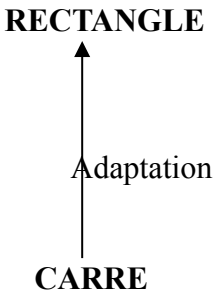
Exemple :



6- hériter pour adapter (proche de la spécialisation)

Définition : B adapte A si B hérite de A et affine A sans rajouter de nouvelles caractéristiques.

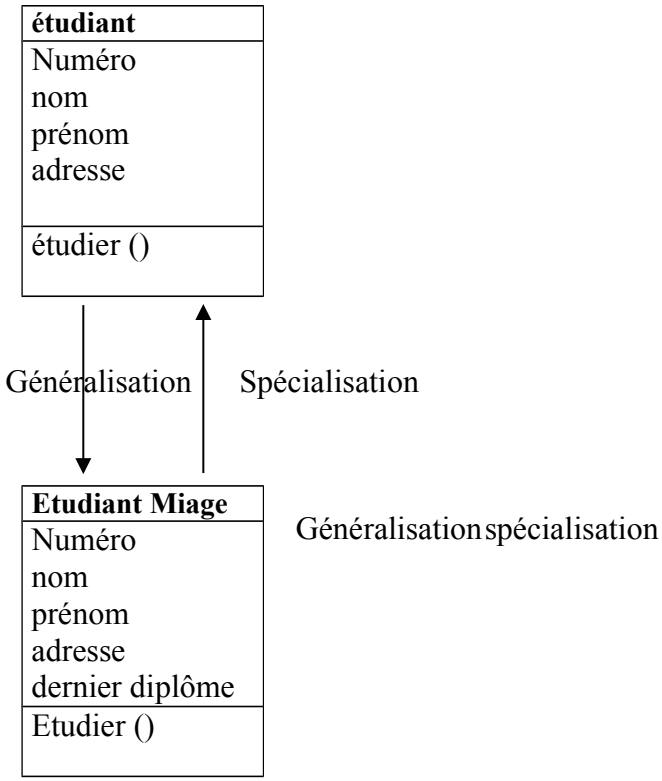
Exemple :



Polymorphisme

Définition : c'est la capacité pour plusieurs classes pour définir la même fonction (méthode)

Exemple : héritage pour spécialiser



Une classe abstraite définis ou moins une fonction nom inplimenté

Exercice :

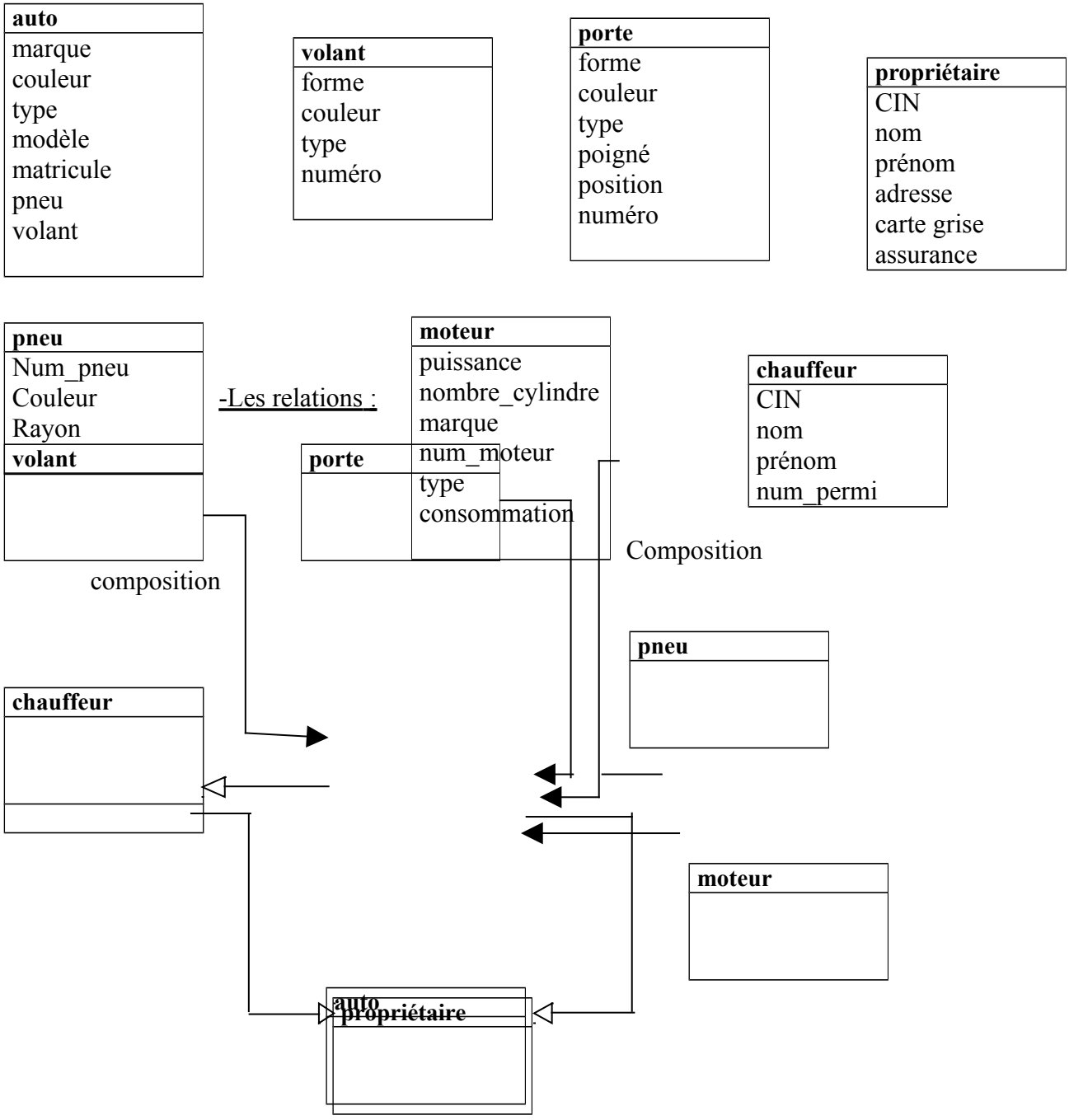
On considère les classe suivant : auto, volant, port, pneu, moteur, chauffeur, propriétaire

Question :

1-determiner les propriétés de chaque classe

2-déterminer la relation entre les classes

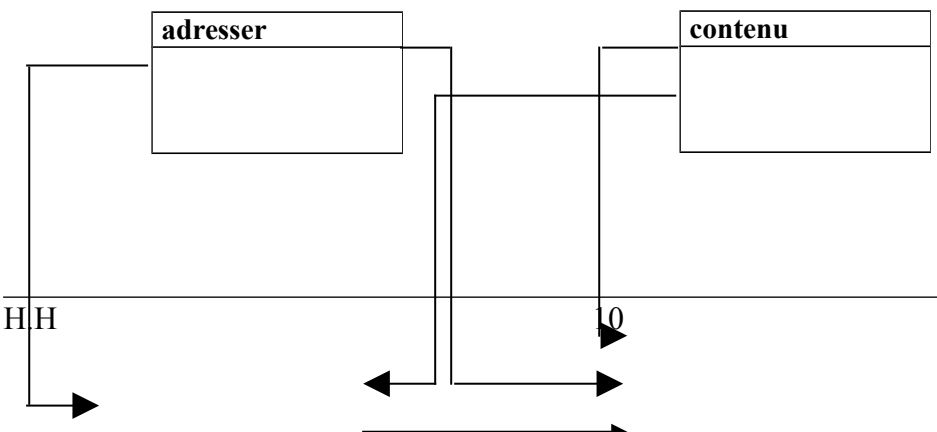
Correction :

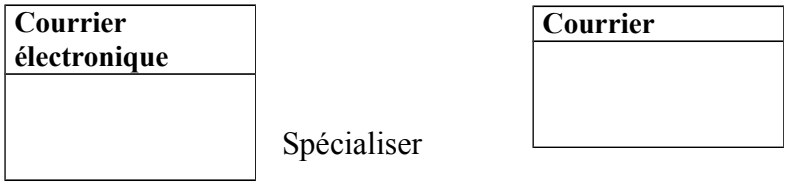


Exercice :

Considérant les classes suivantes : courrier, courrier électronique, adresse, contenu

Correction :





Exercice :

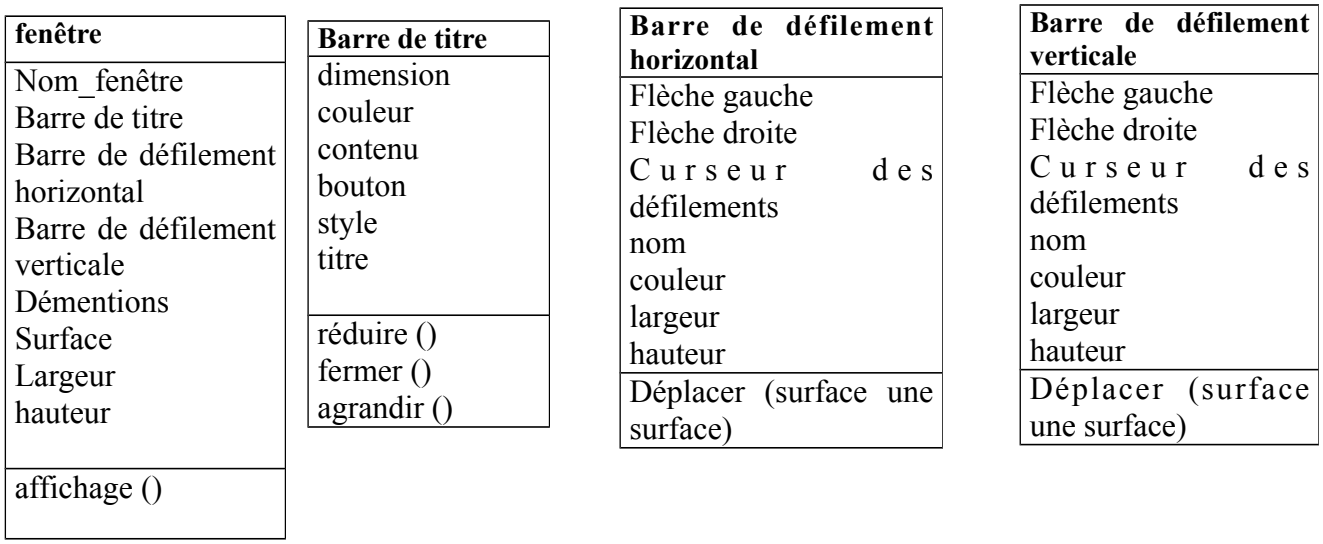
Soient la classe suivante : fenêtre, barre de titre, barre de défilement horizontal, barre de défilement verticale, surface, fenêtre, éditeur du code, onglet, programmeur.

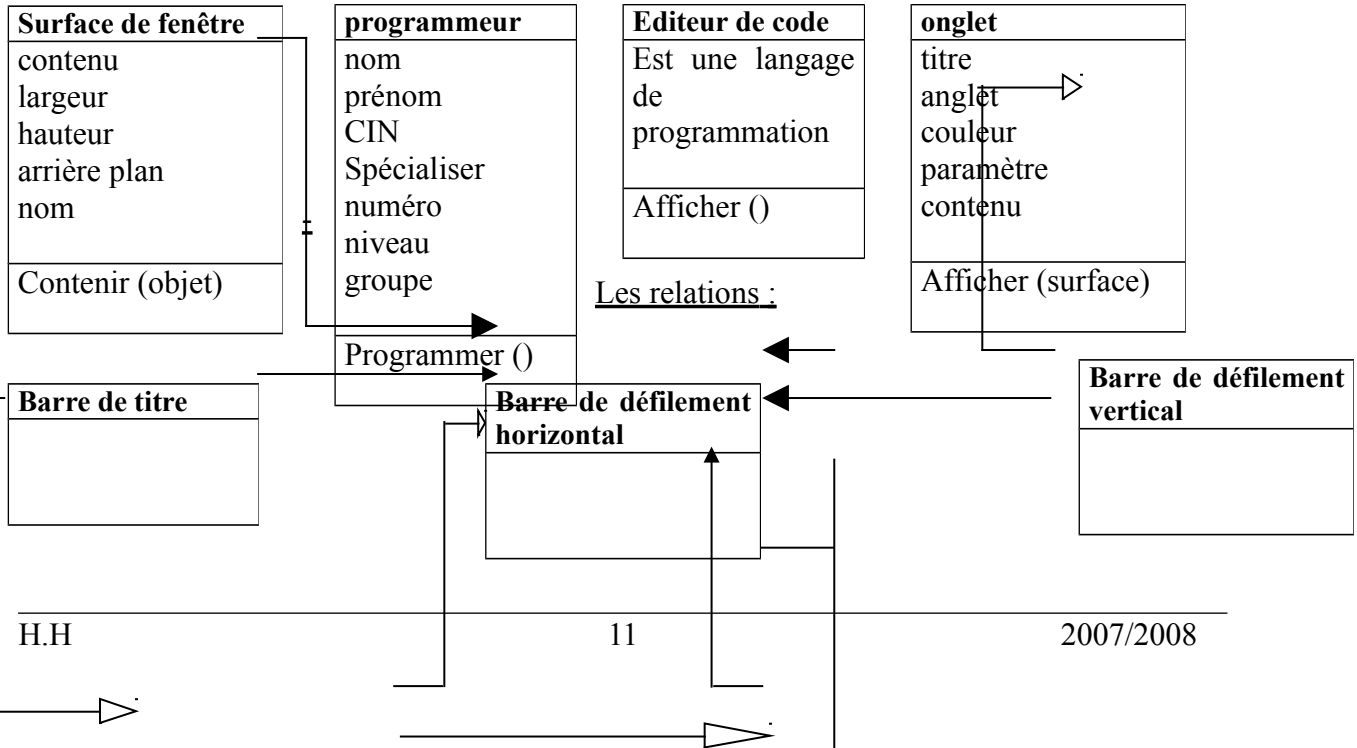
Question :

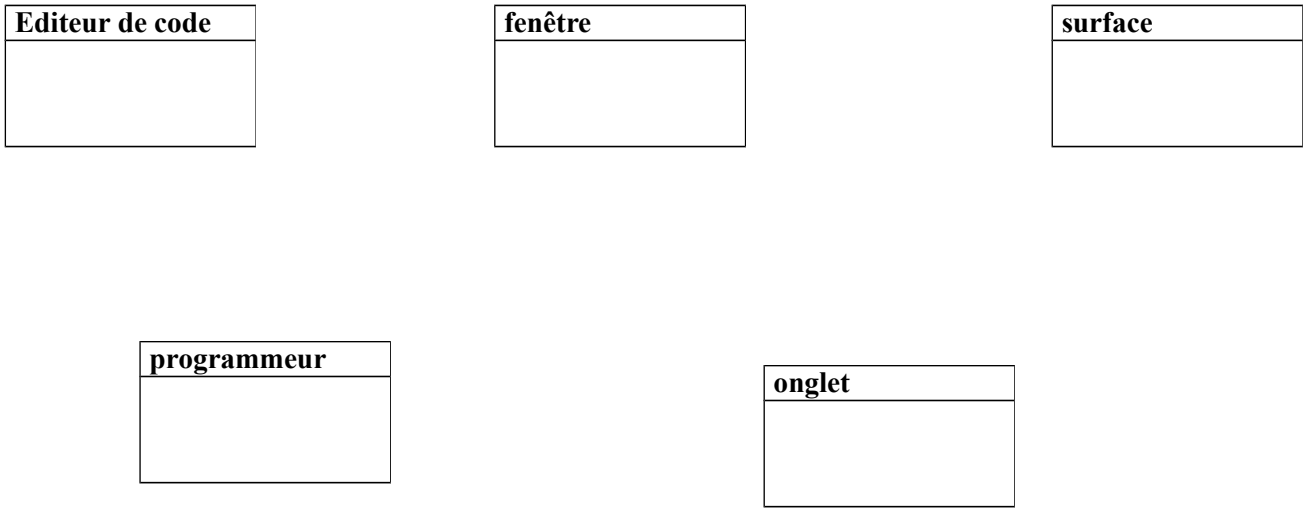
1)-déterminer les propriétés et les méthodes pour chaque classe

2)- déterminer les relations entre les classes.

Correction :







Module : Analyse et conception orientée objet

I. Introduction

1. Modèle :

L'objectif d'un modèle et de créer un schéma significatif du monde réel.
Se schéma doit être plus simple que la réalité, mais aussi la reflète fidèlement.

2. Méthode Orientée Objet :

Une méthode est un langage de modélisation et un processus, il existe plusieurs méthode parmi lequel la méthode BOOCH celle de IVAR JACOBSON et la méthode OMT de GAMES RUMBAUGH c'est trois hommes ont crée au sein de Rational Software, la méthode objectory, aujourd'hui design sous le nom RUP (Rational Unified Process).
L'analyse et conception orientée objet impose la construction des modules efficace, elle met en jeu deux éléments important, un langage et un processus de modélisation.

a. **Langage de modélisation** : c'est simplement une convention des modèles qui permet de définir notre modèle sur le papier, il reste bien entendre à faire coller ses conventions à toutes les personnes travaillant avec nous, il serait encore plus pratique d'avoir recours défini un en fois pour toute.

Le langage universel en matière de développement logiciel et la notation UML (Unified Modelign Langage).
Le rôle du langage de modélisation est de reprendre à des questions telles que comment représenter une classe ? Comment représenter une relation d'héritage et comment représenter un acteur ? Comment représenter une opération ?

Les tailles sur le langage UML (en sera plus tard).

b. Le processus :

Le processus d'analyse et conception orientée objet est plus important et plus complexe que le langage adopté.

Le processus de conception est itératif, se qui ségnifié qu'au cours la totalité du processus est répété au for et mesure de l'adjonction d'amélioration.

Dans une conception itératif de concept (une idée de ce que l'on veut construire) le concept constitué le point de départ, cette vision s'étend et évolue avec l'examen des détailles des que nous avons une idée des besoins la conception est démarré, toutes en sachons que les questions qui vont se posé en entraîneront des modifications qui auront probablement un effet sur le besoin.

Bien que les étapes de processus soient répétés, et il est pratiquement impossible de d'écrire d'une tel manière cylindre, ils seront d'écrites, séquentiellement (ligne par ligne) :

*Les étapes du processus de la conception orientée objet sont :

- ☛Conceptualisation (vision)
- ☛Analyse des besoins.
- ☛ Conception.
- ☛ Implémentation.
- ☛Test.
- ☛Livraison.

❶ : La conceptualisation et la « vision » :c'est une simple phase qui d'écrit l'idée de ce que vous voulez faire construire.

❷ : L'analyse : est le processus de compréhension de besoin.

❸ : Conception : est la phase de création du modèle de classe à partir de la quel le code sera générer.

❹ : L'implémentation : c'est la transcription (transformation) dans le code. Exemple : code en java, en c++, visual basic…

❺ : Test : c'est la vérification que tous se passe comme privée.

❻ : Livraison : c'est la remise de client.

✱**La vision** : tout bon logiciel (application) commence par une vision.

La vision d'un projet est une simple phase qui le résume.

✱**Analyse des besoins** : la phase de conceptualisation, dans la quel s'articule la vision et brève, il ne doit pas être confondue avec les besoins pour passé à un analyse, vous devez comprendre e que doit faire le produit et comment in doit le faire.

Uses cases= (cas d'utilisation) :

Les cas d'utilisation (uses cases) : sont tous simplement une description en niveau supérieur de la façon dont l'application sera utilisée, les cas d'utilisation orienté seulement l'analyse mais aident à déterminer les classes.

Les élaborations d'un jeu complet de cas d'utilisation peut représenter le travail de plus important de l'analyse, les exprès du domaines sont les plus qualifier pour vous aidez dans cette tache.

✱**Cas d'utilisation** : une description de la façon dont logiciel sera utilisée.

✱**Les experts du domaines** : se sont les personnes qualifiés du domaines au quel le produit est distinè.

✱**Acteur :** tout utilisateur au système interagissant avec le système que vous développer.

Avant de déterminer les cas d'utilisation il faut identifier les acteurs.

A. **Identification des acteurs** : les acteurs ne sont pas forcements des gens, les systèmes le sont aussi leur caractéristiques sont :

- D'être externe de système

- D'interagir avec le système.

Écrive la liste des personnes avec nouveau système et le meilleur point de départ

B. Les cas d'utilisation sont déterminées on répondant au question suivant :

① Pourquoi un acteur donné utilise-il le système ?
② Quelles la sortie que l'acteur attend ?
③ Qui incite l'acteur à utiliser le système maintenant ?
④ Que doit faire l'acteur pour utiliser le système ?
⑤ Quelles informations l'acteur doit-il fournit au système ?

Exemple :

Posant ses questions pour l'acteur clients :

✷Le client utilise le système pour obtenir le liquide, faire un dépôt, ouvrir un compte, fermer compte, vérifier son solde.

✷Ajout d'argent dans un compte optantion du liquide ou l'affichage des informations sur le solde.

✷Avoir l'intension de faire un achat, il peut avoir reçus de l'argent sur son compte.

✷Insérer une carte dans la machine.

✷Enter un code.

Après avoir explorer en détail les cas d'utilisations d'un acteur ils convient de développer ceux des autres acteurs.

C. Créer le modèle du domaine :

Cette première approche veut permet de développer vous spécification des besoins avec un modèle du domaine détailler. Elle vous permet de capturées tous les objets du domaine décrivant les objets mentionner dans vos cas d'utilisation.

Pour chacun de ces objets du domaine nous allons capturés des données essentielle comme le nom de l'objet (client, compte, etc…) et si l'objet est un acteur il faut capturer les attributs et les comportements principaux de l'objet (description d'une classe).

Il faut déterminer aussi les relations entre les objets du domaine.

Remarque :

Les objets d'écrits ne sont pas des objets de la conception mais des objets du domaine (objets d'analyse).

D. Etablir les scénarios :

-Avec nos cas d'utilisation préliminaires ne pouvant formaliser les cas et les approfondir

-Un scénario est une description d'un jeu spécifique de serquenstance qui font la distinction parme les éléments du cas d'utilisation

Exemple :

☛Le cas « le client retire de l'argent de son compte » peut avoir les scénarios suivants :

-Le client demande un retrait de 3000dh de son compte, il permet les billets et les systèmes impriment un reçu.

-Le client demande un retrait de 3000dh de son compte, mais son solde est de 2000dh, il est avisé que son compte n'est pas suffisamment approvisionne.

-Le client demande un retrait de 3000dh de son compte, mais il est déjà retirer 10000dh aujourd'hui, il a limité de 3000dh par jour, il est informé des problème et doit décider s'il accepte de ne prélève que 2000dh.

-Le client demande un retrait de 3000dh de son compte, mais le reçu ne peut être imprimer par manque de papier, il est informé des problèmes, il doit décidé s'il veut ou non pour suivre l'opération son recevoir de reçus.

Exercice

On veut automatiser la gestion des notes des étudiants pour l'école MIAGE.

Question :

Réaliser les étapes de l'analyse des besoins :

1) Identifier les acteurs :
   - *Professeur
   - *Directeur
   - *Secrétaire
   - *Etudiant
2) Les cas d'utilisations
   - **a.** Cas d'utilisation pour professeur
     - *Saisir les notes
     - *Mettre à jour les notes
     - *Se connecter
     - *Enregistrer
   - **b.** Cas d'utilisation pour le directeur :
     - *Donner des observations
     - *Saisir le résultat final.
     - *Se connecter.
   - **c.** Cas d'utilisation pour le secrétariat :
     - *Donner le bulletin des notes
     - *Consulter les notes
     - *Imprimer le relever des notes
     - *Se connecter
   - **d.** Cas d'utilisation pour l'étudiant
     - *Consulter les notes
     - *Se connecter
     - *Entrer me mot de passe
     - *Vérifier les notes
3) Créer le modèle du domaine :

- ◆ Note
- ◆ Matières
- ◆ Relever des notes
- ◆ Observation
- ◆ Résultat final
- ◆ Directeur
- ◆ Professeur
- ◆ Secrétariat
- ◆ Etudiants
- ◆ Bulletin

Direction

-Caractéristiques (numéro, type directeur)

-Comportement (donner observation, saisir résultat)

Professeur :

-Caractéristiques (numéro, prénom…)

-Comportement (saisir les notes, mettre à jour)

Secrétariat :

-Caractéristiques (numéro, nom)

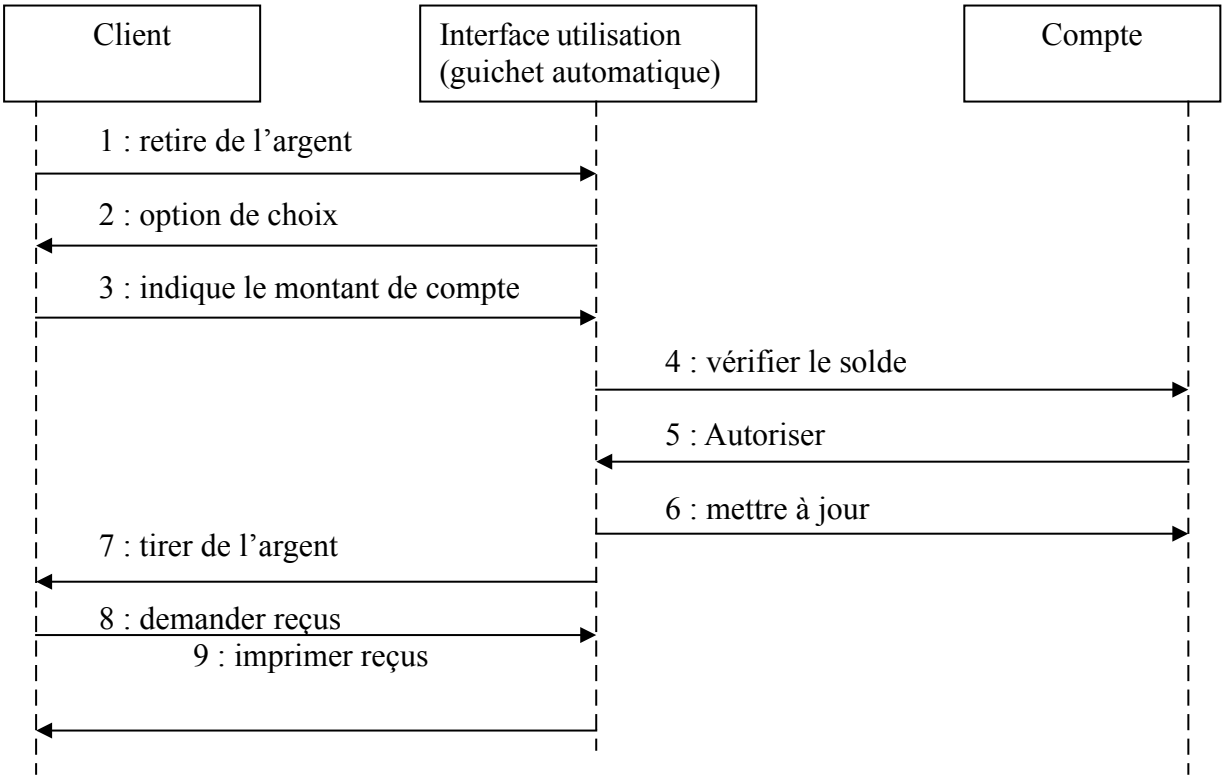
-Comportement (imprimer)

Etudiant :

-Caractéristiques (numéro, nom)
-Comportement (consulter les notes…)

E. Diagramme d'interaction :

Le diagramme d'interaction capture les détails de scénarios pour un cas d'utilisation donné.
Les objets qui interagissent sont des objets des domaines.
Exemple : Pour le cas d'utilisation retire de l'argent.
Représente les interactions des objets suivant des domaines client, interface, et compte utilisateur (guichet automatique).
Le digramme d'interaction montrant la représentation de cette interaction :



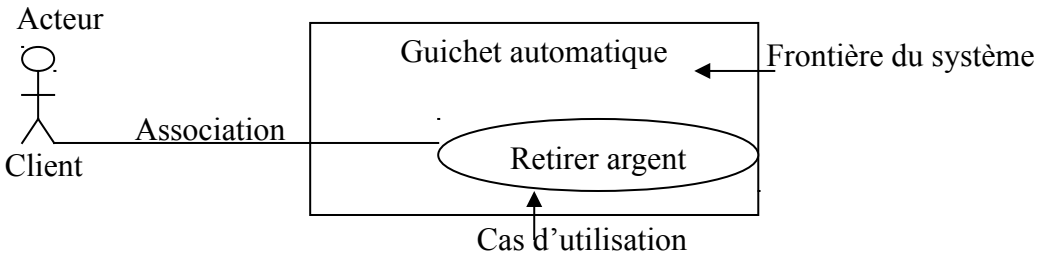
F. Analyse de l'application :

Les besoins de l'application son les conditions préliminaire (contrainte du client besoin…).

Idialement, vous analysez les problèmes consoler la solution, puis choisir les plates formes et des systèmes d'exploitations qui sont le mieux adapté. Dans les pratiques ses clients qui décident en fonction de existant et vous doivent en tenir compte su système

G. Représentation graphique de cas d'utilisation (uses cases) :

Ce cas d'utilisation retire argent peut se représenter sous forme du schéma suivant :

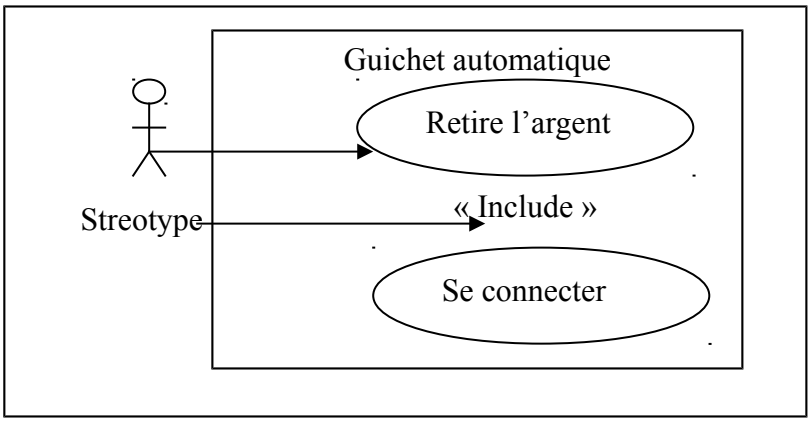


La seule information capturée ici est l'interaction abstraite entre l'acteur « client » et le système, mais en peut détaille le représentation en l'intredipant les relations entre les cas d'utilisations (uses cases).

i. Relation d'inclusion « include » :

Un cas d'utilisation A inclus un cas d'utilisation B, c'est le cas A dépend de B lorsque A solusité, B est obligatoirement comme une partie de A (l'exécution de A nécessite l'exécution de B).

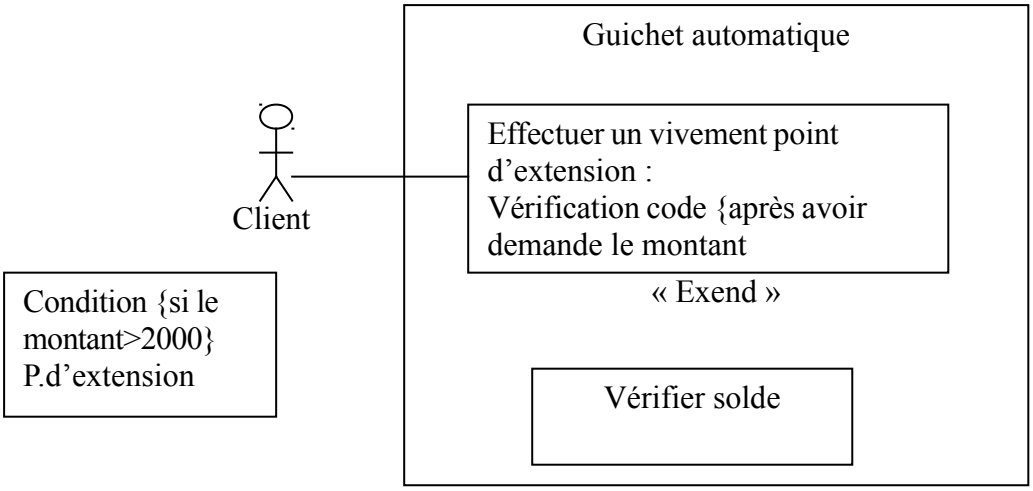
Exemple :



ii. Relation d'extension « Extend » :

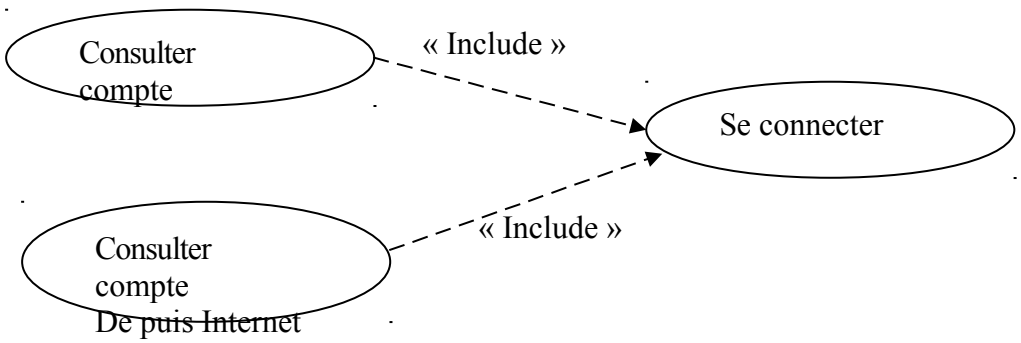
Un cas d'utilisation B étend un cas d'utilisation A lorsque le cas d'utilisations B peut être appelé au cour d'exécution. Le cas d'utilisation A, exécution A peut éventuellement entériner l'exécution de B contrairement à l'inclusion l'exécution est l'optionnel. Cette dépendance est symbolisée par les stéréotypes « étend » l'exécution peut intervenir à un point précis de point prisse du cas étendu. Ce point s'appelle « point d'extension », « extension point », il porte un nom qui figure dans un compartiment du cas étendu (A) ce la ré brique point d'extension, et éventuellement associe à une condition indiquant le moment ou l'extension intervient graphiquement une condition exprimé se forme d'une note.

Exemple :



iii. Relation de généralisation :

Un cas A généralise un cas B c'est B est un cas particulier de A



Exercice :

On veut automatiser la gestion d'une agence de voyage.

Question :

-Déterminer les acteurs ?
-Déterminer graphique les cas des utilisations,
-Représenter graphiquement les cas d'utilisation ?
-Déterminer le diagramme d'interaction.

Acteur :

- ♦Directeur
- ♦Récepteur
- ♦Voyage
- ♦Chauffeur

Les cas d'utilisation :

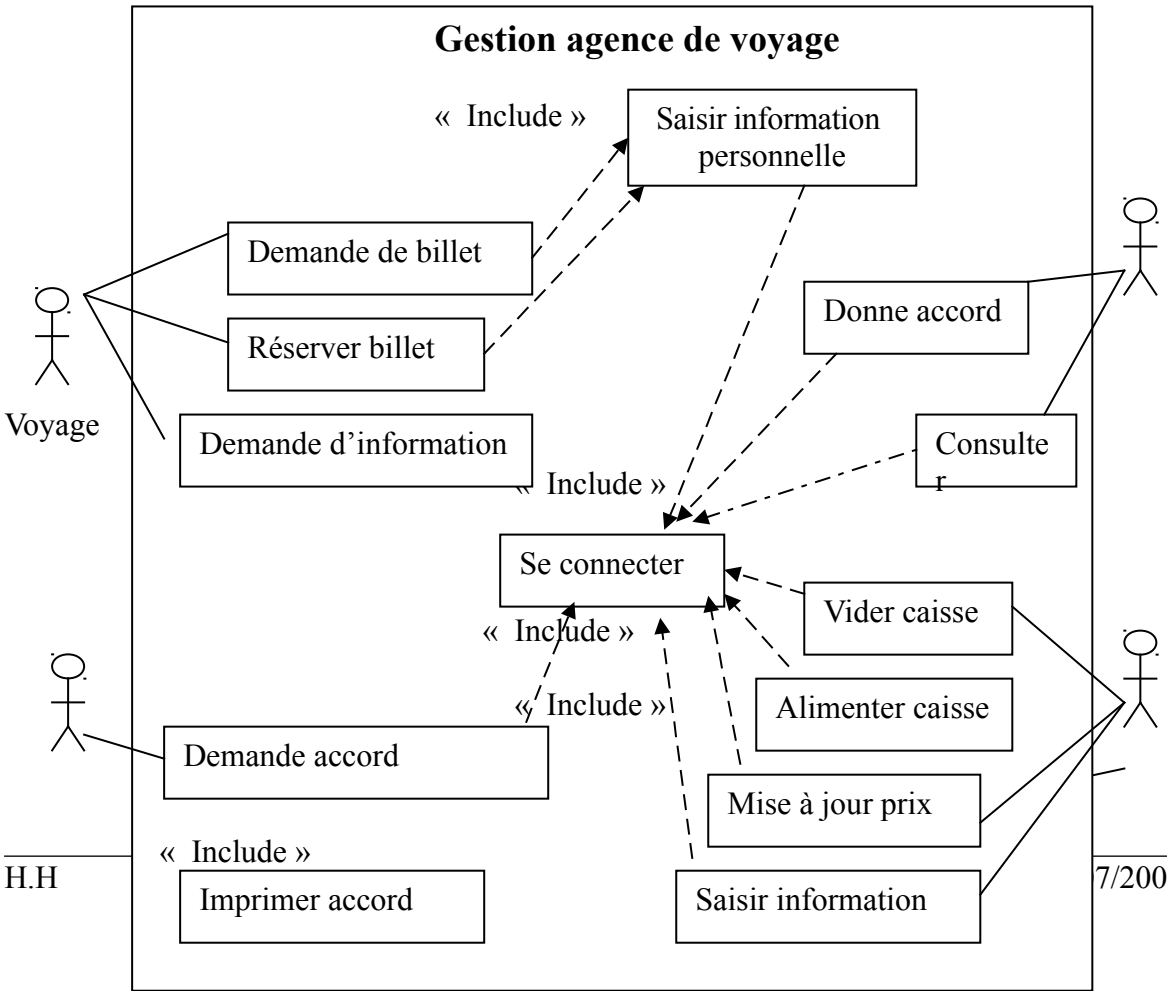
- ● Directeur : consultation
  - ✷ Se connecter
  - ✷ Donner accorde
- ● Réception : mise à jour des prix
  - ✷Se connecter
  - ✷Saisir les informations exceptionnelles.
  - ✷ Alimenter le système par papier
  - ✷ Vider la caisse du système
- ● Voyage : se connecter
  - ✷Demande billet

- ✷Demande les informations personnelles
- ✷Réserver billet
- ✷ Imprimer billet

● Chauffeur : se connecter

- ✷Demande d'information (accord)
- ✷Imprimer accord

Diagramme d'interactions :



Réception

Chauffeur

Acteur	Cas d'utilisation
Stagiaire	Ouvrir la session
Cardien	surveiller
Technicien	configurer

Exercice :

Étude de cas.

L'école MIAGE décide de restreindre les accès à certaines salles (les salles de TP) au moyen de mots de passe pour chaque salle de TP. L'accès à un PC est commandé par le mot de passe de la salle où se trouve ce PC.

Les droits d'accès sont alloués entre les filières et les PC des salles de TP. Un PC donné ne peut appartenir à plusieurs filières.

A chaque filière, on affecte les différents OC d'une salle.

Un technicien est responsable de la configuration initiale du système et la mise à jour des différentes informations de définition des filières et de PC. Le technicien doit être identifié par le système.

Un gardien dispose d'un écran de contrôle et est informé des tentatives infructueuses.

Questions :

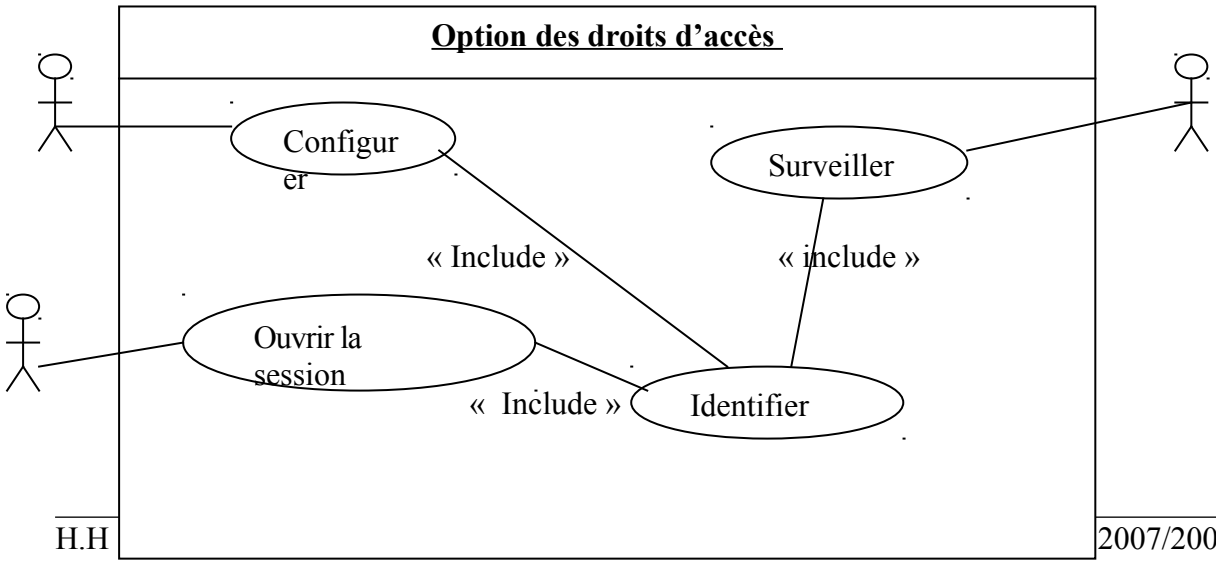
1. rechercher les acteurs (utilisateurs à l'extérieur du système) ?
2. déterminer les cas d'utilisation ?
3. élaborer le diagramme des cas d'utilisation
4. décrire textuellement chaque cas d'utilisation (scénario) ?
5. élaborer le diagramme d'interaction.

Correction :

1) Les acteurs

2) Les cas d'utilisations

3) le diagramme des cas d'utilisation :



4) Scénario :

a. Configurer :

Identification : le technicien donne son nom et son mot de passe.

- Le système vérifier le mot de passe
- Si le mot est validé alors le système autorise l'accès.
- Si non le système affiche un message pour ressaisir le mot de passe et le nom.

Configuration :

1) Le technicien demande la liste des stagiaires d'une filière donnée.
- Le système affiche la liste des stagiaires.
- Le technicien choisi un stagiaire donnée.
- Le système affiche les informations concernant le stagiaire.
- Le technicien modifier les données concernant le stagiaire
- Le système sauvegarde les données modifiées.

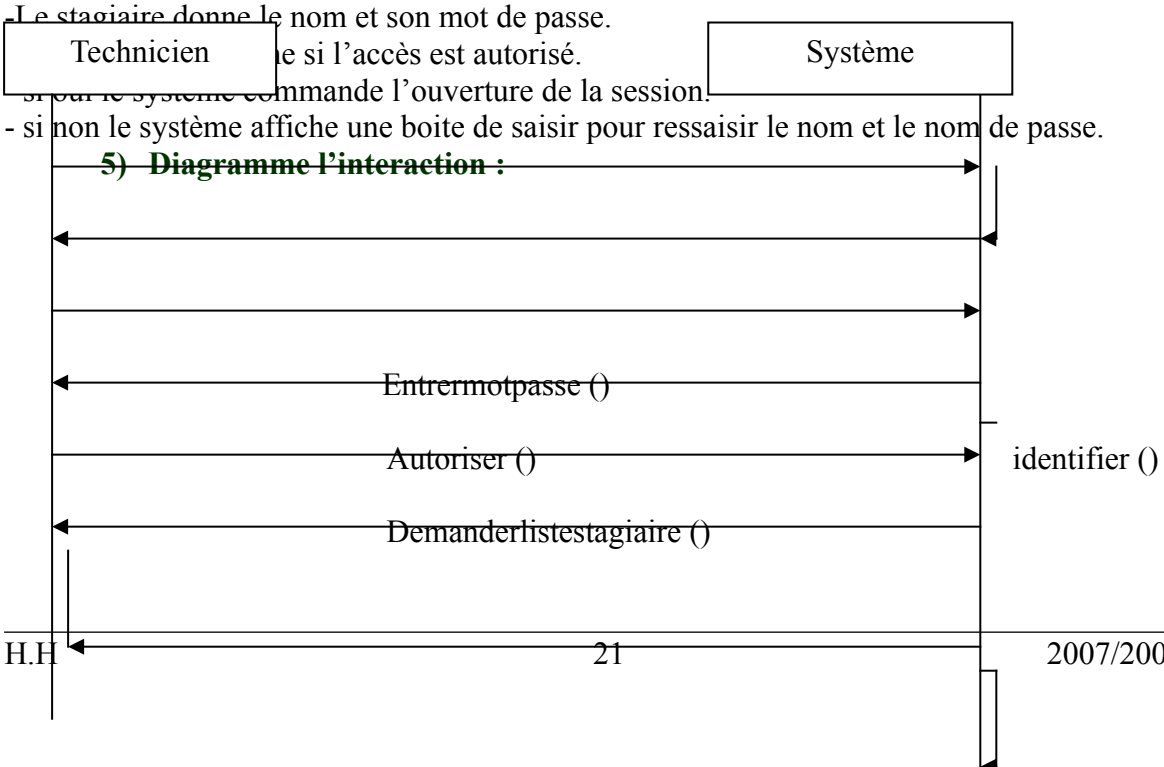
2) le technicien demande la liste des PC d'une salle de TP données.
- le système affiche la liste des PC.
- le technicien choix un PC donnée.
- le système affiche les informations concernant le PC.
- le technicien modifier les données concernant le PC.
- le système sauvegarde les données modifiées.

b. Surveillance :
- le gardien demande la liste des stagiaires.
- Le système affiche la liste des stagiaires.
- Le gardien choisi le stagiaire en question.
- Le système affiche les informations sur le stagiaire.
- Le gardien saisi le rapport d'événement.
- Le système sauvegarde le rapport.

c. Ouvrir une session :
- Identification
- Ouverture d'une session

-Le stagiaire donne le nom et son mot de passe.
- si le système commande l'ouverture de la session.
- si non le système affiche une boite de saisir pour ressaisir le nom et le nom de passe.

5) Diagramme l'interaction :



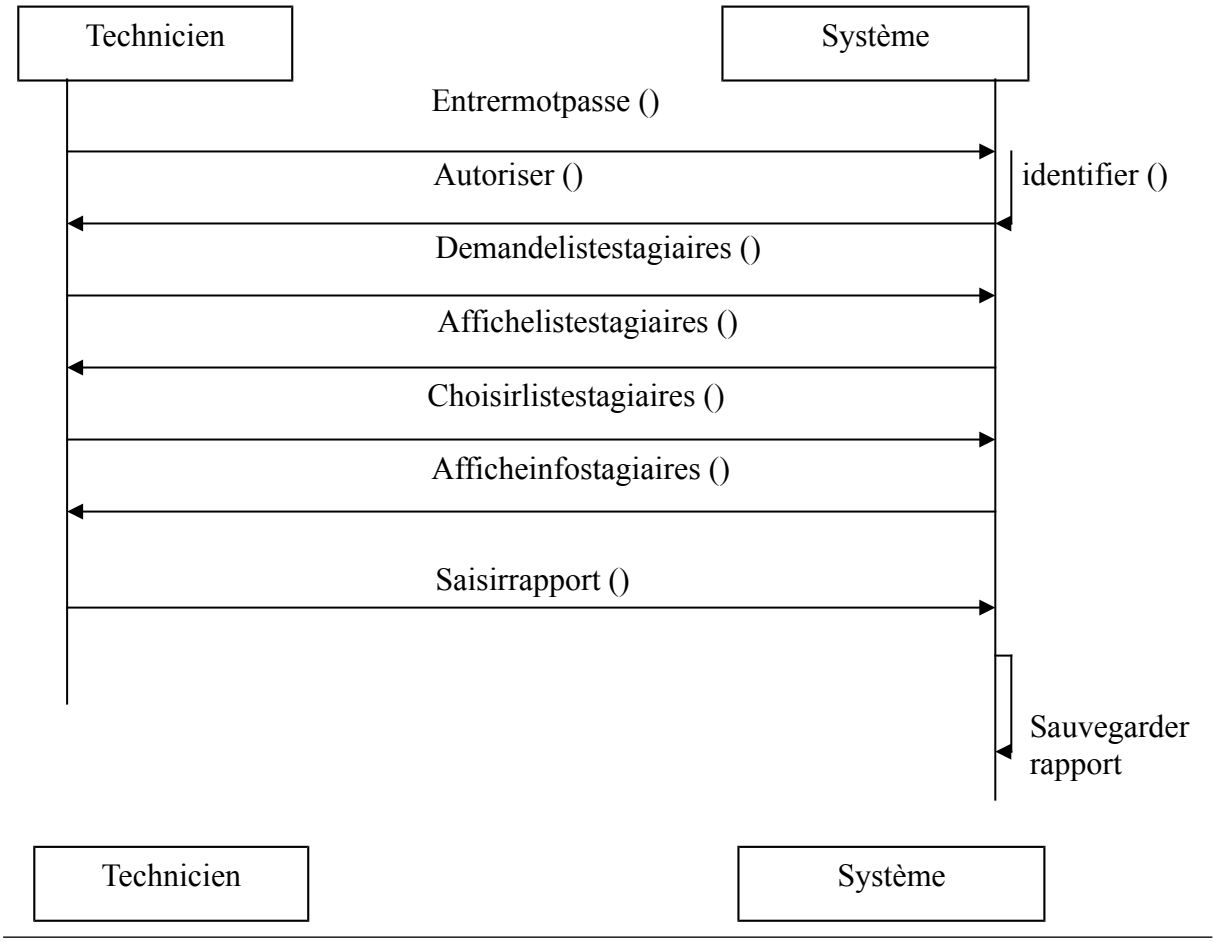
Afficherlistestagiaires ()

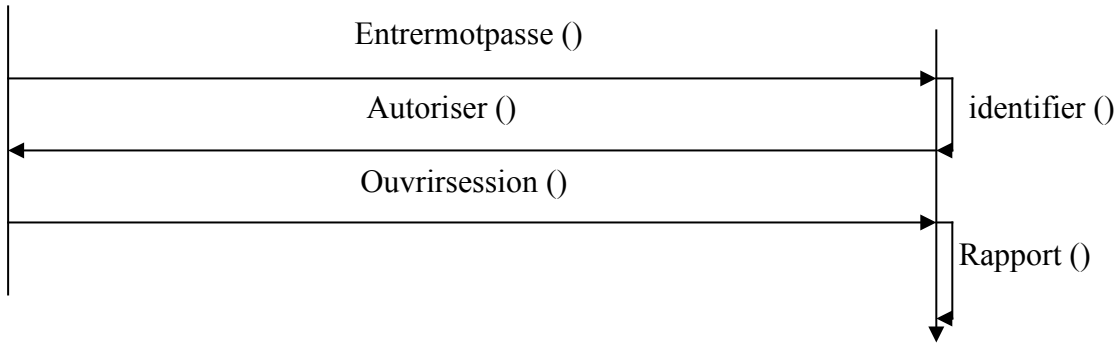
Choisirsatagiaire ()

Afficherinfostagiaires ()

Modifier () validemodofication ()

Sauvegarder ()





Conception

L'analyse met sur le domaine du problème, alors que la conception se concentre sur la création de la solution, c'est-à-dire la transformation de notre compréhension des besoins en un modèle pouvant être mis en œuvre dans le code.
Ce modèle est divisé en deux sections : les mécanismes d'architecture et de conception de classe.

Cette dernière est à son tour divisée en conception statique (détail des classe, de leurs relations et caractéristiques), et conception dynamique (détail des interactions entre classes).

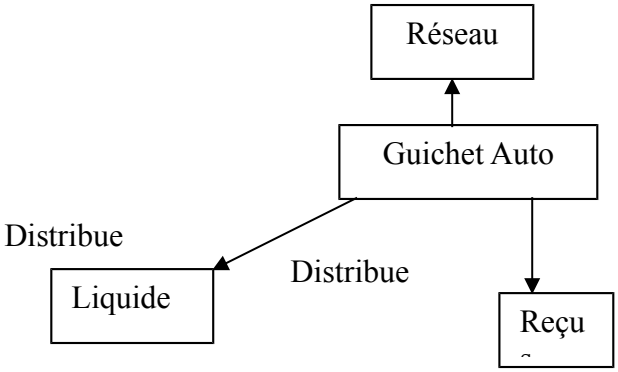
I. Mécanismes d'architecture :

Le mécanisme d'architecture fournit des détails sur la façon dont on implémente la persistance d'objets, les conflits d'accès etc.

La plus grande difficulté est de déterminer le jeu initial de classes et de comprendre ce qui fait une classe bien conçue. Une technique très simple suggère d'écrire les scénarios des cas d'utilisation puis de créer une classe pour chaque nom.
On considère le scénario suivant :

Le client choisit de retirer du liquide de son compte courant. Le compte est suffisamment approvisionné, le guichet automatique est alimenté en liquide et en reçus, et le réseau est opérationnel. Le guichet automatique demande au client d'indiquer un montant pour le retrait, et le client demande 300F ce qui est autorisé. La machine distribue 300F et imprime un reçu, le client prend l'argent et le reçu.
On peut déduire de ce scénario les classes suivantes :
Client, liquide, compte courant, compte, reçus, guichet Auto, réseau, montant, retrait, machine et argent.

On peut regrouper les synonymes pour créer la liste suivante, puis créer des classes pour chacun de ces noms :
Client
Liquide (argent, montant, retrait)
CompteCourant
Reçus
Guichet auto (machine)
Réseau
On peut ensuite schématiser les relations évidentes parmi certaines de ces classes comme suit :



L'extraction des noms du scénario est surtout le début de la transformation des objets provenant de l'analyse du domaine. Bien souvent, nombre des objets du domaine auront des substituts dans la conception. Un objet est appelé substitut pour distinguer le reçu physique distribue par la machine, de l'objet dans la conception qui est une abstraction implémentée dans le code.

On va découvrir que la plupart des objets du domaine ont une représentation un pour un entre l'objet du domaine et l'objet de conception.

Après avoir transformé les objets du domaine, vous pouvez commencer à rechercher d'autres objets utiles au moment de la conception, à commencer par les interfaces. Chaque interface entre votre nouveau système et tous les systèmes existants doit être encapsulée dans une classe de l'interface.

Ses classes d'interface offrent l'encapsulation su protocole d'interface et protégent donc votre code des altérations par autre système. Elles vous permettent de modifier votre propre conception, on de satisfaire les Changements dans la conception des autres systèmes, sans altérer le reste du code. Tant que les deux systèmes Continuent de gère l'interface convenue, ils peuvent évoluer indépendamment l'un de l'autre.

II. Modèle statique

On peut commencer à modéliser les relation et interaction entre les classe les trois points essentiels du modèle statique sont : les responsabilités, attributs et relation.
Chaque objet doit être une instance d'une classe bien définie chargée d'une responsabilité particulière.

Les classes délèguent généralement les responsabilités sans rapport à d'autres classes connexes, ce qui facilite la maintenance d'un programme

Le but est définir les classes à l'aide de fiches, d'identifier leurs responsabilités et de comprendre leurs interactions.

On peut prévoir une série de fiche, sur lesquelles on inscrit le nom d'une classe en haut, puis on sépare la fiche en deux colonnes : responsabilités et collaboration. On commence par compléter les fiches pour les classes les plus importantes qu'on a identifiées. On inscrit également « superclasse » sous le nom de la classe et les noms des classes dérivées.

A ce stade, on ne préoccupe pas des relations, ni de l'interface, pas plus que de savoir si telle ou telle méthode sera publique ou privée.

Une fois cette série de fiches établie, on commence à reprendre les scénarios.

Par exemple :

Le client choisit de retirer du **liquide** de son **compte courant. Le compte** est suffisamment approvisionné**, le guichet automatique** est alimenté en liquide et en **reçus, et le réseau est** opérationnel. Le guichet automatique demande au client d'indiquer **un montant** pour **le retrait**, et le client demande 300 F ce qui est autorisé**. La machine** distribue 300 F et imprime un reçu, le client prend l'argent et le reçu.

La suite ressemble à un jeu de rôles ou chaque participant se met dans la peau du personnage qui donnerait

Vie à chaque classe. Par exemple, le participant possédant la fiche « Compte Courant » va dire : « je dit au client la somme qui disponible sur le compte. Il me demande 300 F, j'envoie un message à la machine pour lui demande de la faire. » Le titulaire de la fiche Guichet Auto prend alors la parole : « je suis le distributeur, je donne 300 F et j'envoie à Compte Courant un message pour lui dire de débiter le compte de 300 F. la machine dispose de 300 F de moins. A qui dois-je le dire est ce que je dois en garder la trace », etc.

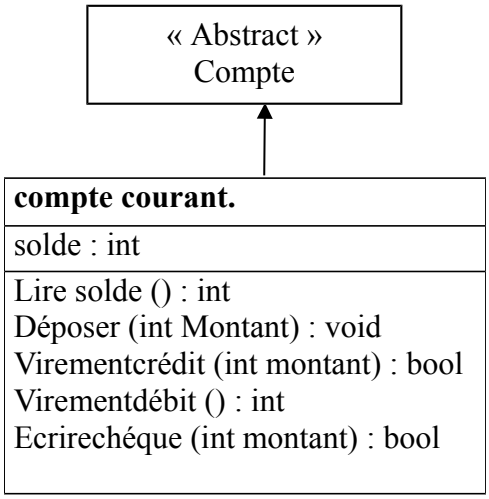
Cette première approche permet de clarifier les responsabilités de chaque et leurs interactions. Il est difficile de passer des fiches au code.

Pour pouvoir modéliser un modèle complet et fiable, on doit évoluer vers la notation UML.

Chaque fiche peut être traduite directement en une classe UML, les responsabilités en méthodes de la classe, et les attributs capturés ajoutés. La définition de classe, inscrite au dos de la fiche, devient la documentation de la classe.

Fiche compte courant

Classe :	**compte courant**
Superclasse :	**compte**
responsabilités	**collaborations**
Garder la trace du solde Accepter les dépôts et virement à créditer Ecrire les chèques Transférer le liquide demandé Mettre à jour le solde	**A**utres comptes Distributeur de liquide



La relation être la fiche Compte Courant et la classe UML correspondante.

III. Modèle dynamique

En plus des relations entre classes, il est important de modéliser leurs interactions. Par exemple, les classes compte courant GuicherAuto et Reçus peut interagir avec le client par l'intermédiaire du cas d'utilisation « retirer liquide ».

Diagramme séquentiel

Ce diagramme montre l'interaction entre plusieurs classes.

Exemple

La classe GuicherAuto va déléguer à la CompteCourant tout la responsabilité de gestion du solde, alors que CompteCourant va déléguer à GuicherAuto la gestion de l'affichage à l'utilisateur.

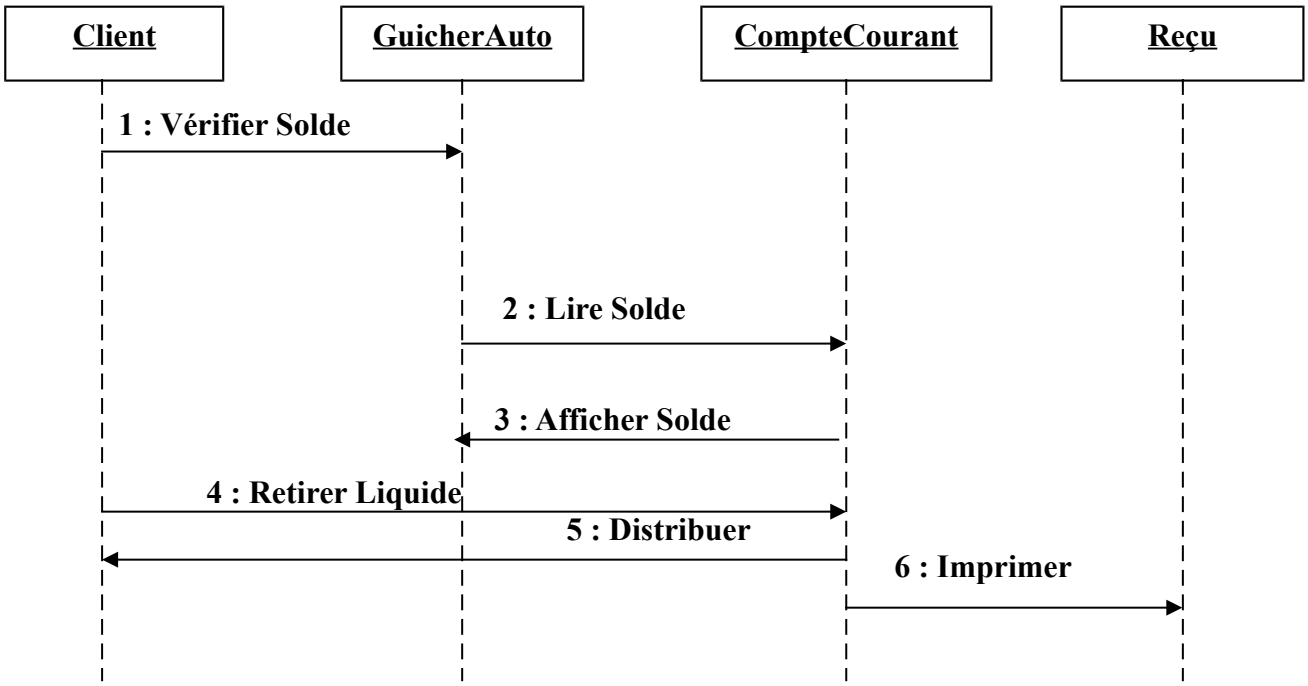
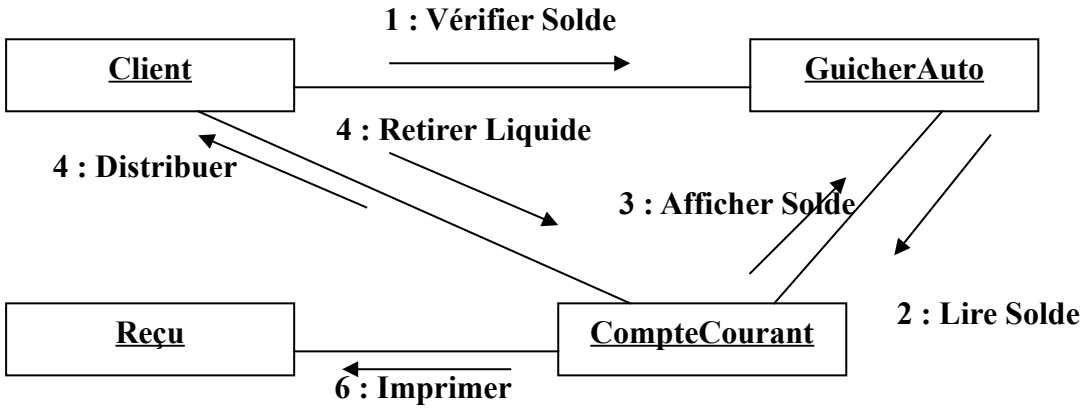


Diagramme en collaboration

Ce diagramme insiste sur les interactions entre classes


1 : Vérifier Solde
Client
GuicherAuto
4 : Retirer Liquide
4 : Distribuer
3 : Afficher Solde
2 : Lire Solde
Reçu
CompteCourant
6 : Imprimer