Nancy-Université
Université Nancy 2

Compléments d'analyse et conception des systèmes d'information (ACSI)

COURS, TD, Etude de cas

Public concerné : DUT Informatique 2ème année

Jacques LONCHAMP

Date : 2008/2009

UNIVERSITE NANCY 2
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
2ter boulevard Charlemagne
CS 5227
54052 NANCY Cedex

Tél : 03.54.50.38.00
Fax : 03.54.50.38.01
http://www.iuta.univ-nancy2.fr

Table des matières

Présentation d'UML

Les méthodes systémiques comme Merise ont marqué une première évolution dans les années 80 autour des idées de **SI**, de **base de données**, de **niveaux de modélisation** (conceptuel, organisationnel, physique) et de **séparation données - traitements**.

Depuis la fin des années 90 l'ACSI connaît une deuxième évolution autour des idées d'**objet** (regroupant données et traitements), de **réutilisation** (code et conception), de **langage de haut niveau** permettant d'exprimer aussi bien l'analyse (la description du problème), la conception (la description de la solution) et l'implantation, **d'architectures complexes** à base de composants distribués et hétérogènes.

Un langage commun tout au long de la démarche

La représentation du monde réel se fait dans les mêmes termes que celle du logiciel : des **objets, des classes, des opérations, des attributs et des associations** permettent d'exprimer le modèle des besoins aussi bien que le modèle d'implantation.

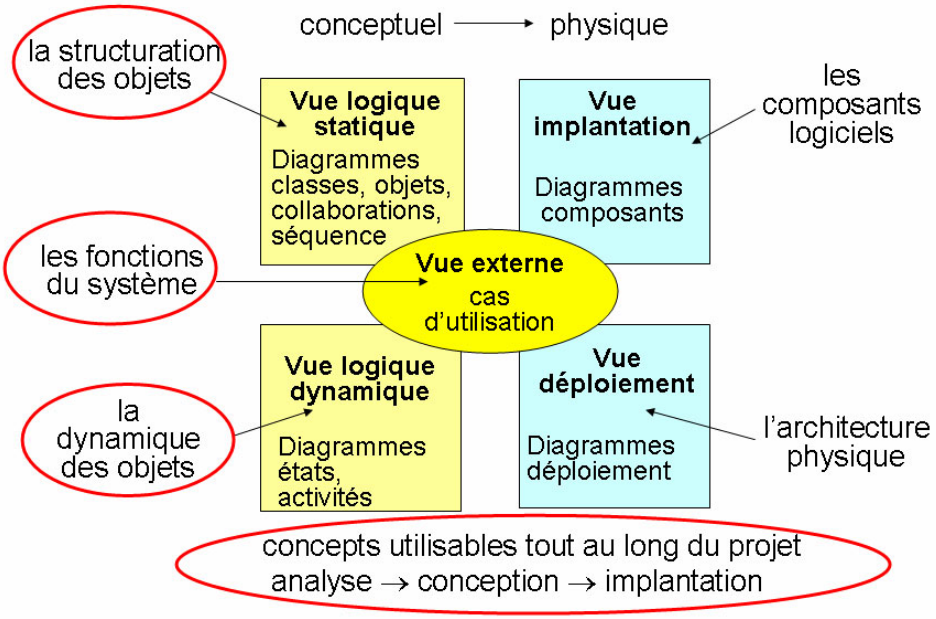
La démarche ne consiste plus à réécrire un modèle d'un certain niveau dans les termes du niveau suivant au moyen de règles de traduction. On passe d'un niveau à un autre par **enrichissement** des éléments existants et **adjonction** d'éléments nouveaux, en conservant **le même formalisme.**

Un langage unifié : UML

Le langage à objet permettant de décrire l'analyse, la conception et l'implantation des systèmes, c'est UML ('Unified Modeling Language'). En bref :

- UML est une notation, pas une méthode : UML définit des modes de représentation (diagrammes et notations) mais pas de démarche standardisée.
- UML est un langage de modélisation objet.
- UML a pour but de documenter les modèles objets.
- UML convient pour toutes les méthodes objet.
- UML est dans le domaine public (la version actuelle est UML 2.1). C'est l'OMG ('Object Management Group') chargé de la normalisation des technologies objets qui pilote UML.

UML est une proposition complexe et en constante évolution dont nous n'étudions que les bases.



Les cas d'utilisation

Définition

Les cas d'utilisation ('use cases') servent à exprimer le comportement du système selon le point de vue des utilisateurs.

Les cas d'utilisation **délimitent le système**, ses **fonctions** (ses cas), et ses **relations avec son environnement**. Ils modélisent à la fois des **activités** (fonctionnalités) et des **communications** (interactions).

Ils constituent en particulier un moyen de déterminer les **besoins** du système. Ils permettent d'impliquer les utilisateurs dès les premiers stades du développement pour exprimer leurs attentes (analyse des besoins).

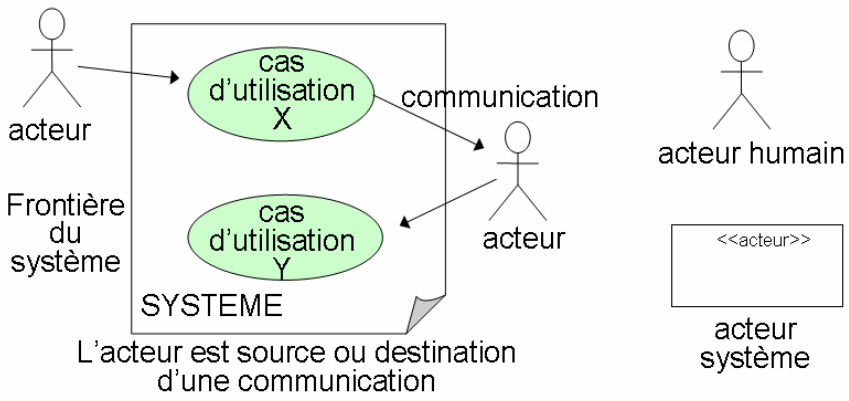
Ils sont utilisables pour tout projet, indépendamment d'UML et de l'approche objet.

Concepts

Un **acteur** est une personne ou un système qui interagit avec le système étudié, en échangeant de l'information (en entrée et en sortie). On trouve les acteurs en observant les utilisateurs directs du système, les responsables de sa maintenance, ainsi que les autres systèmes qui interagissent avec lui.

Un acteur représente un **rôle** joué par un utilisateur qui interagit avec le système. La même personne physique peut jouer le rôle de plusieurs acteurs. D'autre part, plusieurs personnes peuvent jouer le même rôle, et donc agir comme un même acteur.

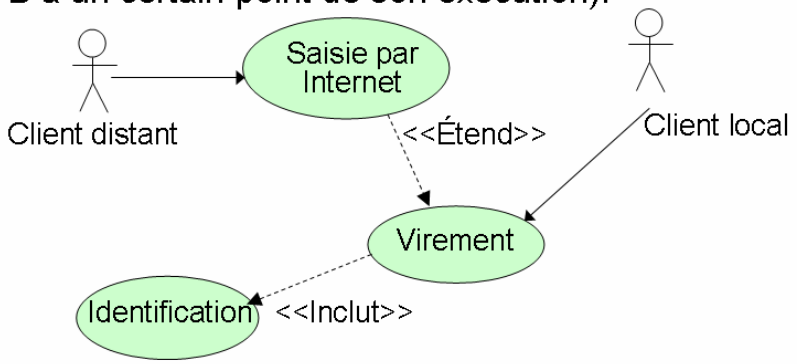
Les diagrammes de cas d'utilisation décrivent le dialogue entre les acteurs et le système représenté comme un ensemble de cas (fonctionnalités). Les communications sont orientées (avec une flèche) ou non. Les communications externes (entre acteurs ne sont pas représentées).



Les cas peuvent être structurés par les relations :

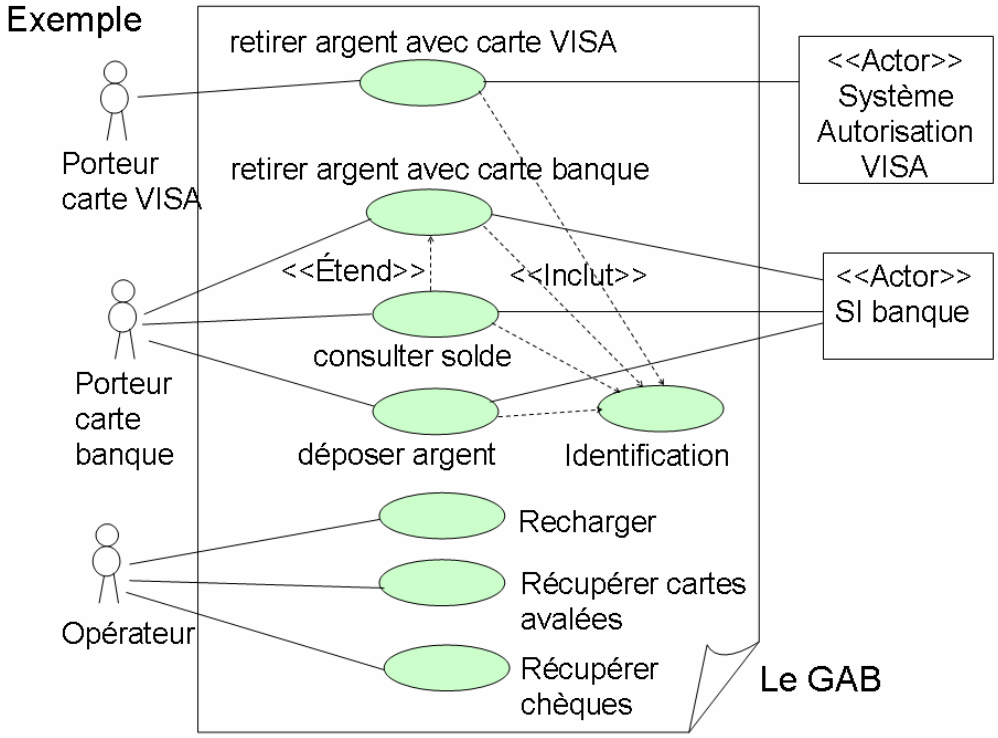
A <<Inclut>> B (le cas A inclut obligatoirement le cas B; permet de 'factoriser' des fonctionnalités partagées)

A <<Étend>> B (le cas A est une extension optionnelle du cas B à un certain point de son exécution).

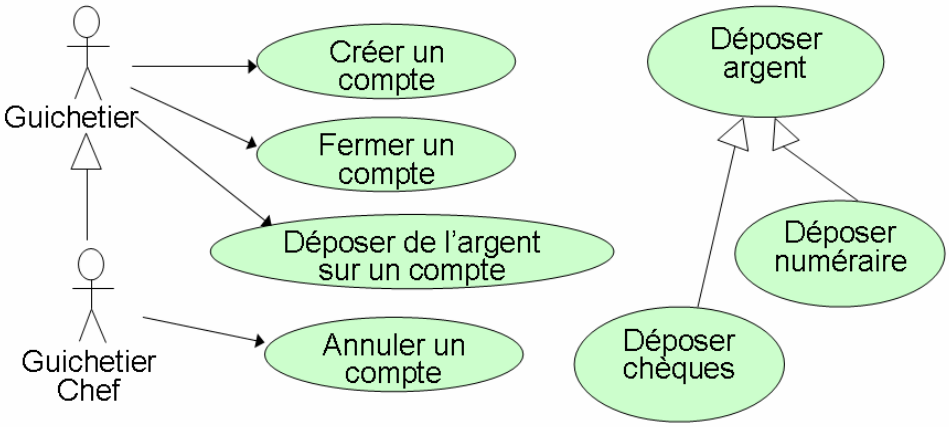


<<xxx>> est un **stéréotype** UML c.à d. un moyen de caractériser et classer des éléments des modèles UML; certains sont prédéfinis, mais les utilisateurs peuvent en définir d'autres.

Exemple



On peut également avoir de l'héritage entre acteurs et entre cas (généralisation/spécialisation).



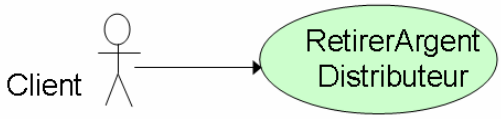
Un guichetier chef est un guichetier spécialisé qui peut faire tout ce que peut faire un guichetier et, en plus, il peut annuler un compte. Déposer chèques et Déposer numéraire sont des spécialisations de Déposer argent.

Spécification des cas

Chaque cas d'utilisation est précisé de manière **textuelle et structurée** en précisant :

- dans quelles conditions la cas peut démarrer (pré-conditions),
- dans quelles conditions la cas peut se terminer (post-conditions),
- les étapes du déroulement normal ('nominal'),
- les variantes possibles et les cas d'erreurs d'erreurs,
- les informations échangées entre acteur et système,
- les éventuelles contraintes non fonctionnelles.

Exemple : cas RetirerArgentDistributeur



Précondition le distributeur contient des billets ; il est en attente d'une opération : il n'est ni en panne, ni en maintenance.

Postcondition si de l'argent a pu être retiré, la somme d'argent sur le compte est égale à la somme d'argent qu'il y avait avant moins le retrait. Sinon, la somme d'argent sur le compte est inchangée.

Déroulement normal

(1) le client introduit sa carte bancaire, (2) le système lit la carte et vérifie si la carte est valide, (3) le système demande au client de taper son code, (4) le client tape son code confidentiel, (5) le système vérifie que le code correspond à la carte, (6) le client choisit une opération de retrait, (7) le système demande le montant à retirer, etc.

Variantes

(A) Carte invalide : au cours de l'étape (2), si la carte est jugée invalide, le système affiche un message d 'erreur, rejette la carte et le cas d'utilisation se termine.

(B) …

Contraintes non fonctionnelles

(A) Performance : le système doit réagir dans un délai inférieur à 4 secondes, quelque soit l'action de l'utilisateur.

(B) …

Les cas d'utilisations peuvent aussi être vus comme des **classes de scénarios.** Chaque scénario correspond à une utilisation particulière, par un acteur donné, dans des circonstances données. On peut décrire les principaux.

SCENARIO 1

Jacques insère sa carte dans le distributeur x233.

Le système accepte la carte et lit le numéro de compte.

Le système demande le code confidentiel.

Jacques tape 'xwrzhj'.

Le système détermine que ce n'est pas le bon code.

Le système affiche un message et propose à l'utilisateur de recommencer.

Jacques tape 'xwrzhi'.

Le système affiche que le code est correct.

Le système demande le montant du retrait.

Jacques tape 300 €.

Le système vérifie s'il y a assez d'argent sur le compte.

etc.

Mode d'emploi

Le processus exploitant les cas d'utilisation peut donc se résumer ainsi :

- trouver les acteurs, les décrire brièvement,
- trouver les cas d'utilisation, les décrire brièvement,
- construire le diagramme des cas d'utilisation,
- structurer éventuellement le modèle (relations <<Inclut>> et <<Étend>>, généralisation/spécialisation),
- spécifier chaque cas (description textuelle structurée); éventuellement donner les scénarios les plus importants.

Conclusion

Le diagramme de cas d'utilisation est plus riche que le diagramme acteurs/flux de Merise.

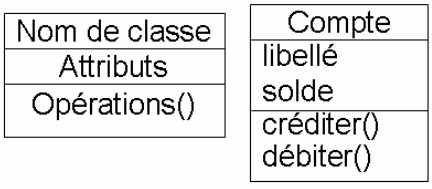
En plus des acteurs et des communications, il liste les principales fonctionnalités attendues. Il permet de les organiser grâce aux relations d'héritage, d'inclusion et d'extension.

Avec la notion de description textuelle et de scénario, l'analyste dispose d'une riche panoplie pour exprimer de manière semi-formelle les **besoins fonctionnels et non fonctionnels du système étudié** (son cahier des charges).

Les diagrammes de classes

Concepts

Les diagrammes de classes expriment la structure **statique** du système. Ils décrivent l'ensemble des classes et leurs associations. Une **classe** décrit un ensemble d'objets (instances de la classe). Elle peut être vue comme la factorisation des **éléments communs** à un ensemble d'objets (les différences sont ignorées). Durant l'analyse et la conception, il est plus économique de raisonner en termes de classes que d'objets individuels.



Indicateurs de visibilité des attributs et opérations :

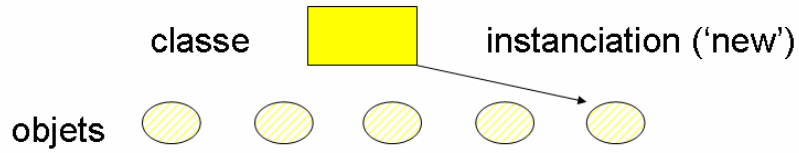
+ public (visible par tous)

- privé (visible dans la classe uniquement)

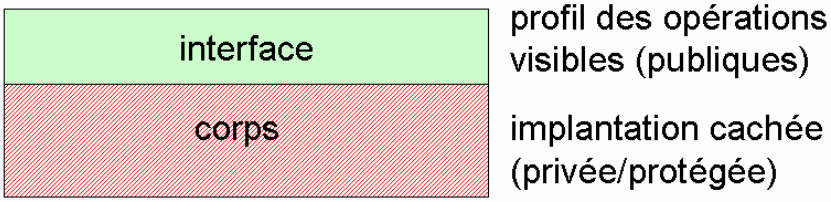
protégé (visible dans la classe et ses sous classes)

Classe = type + module

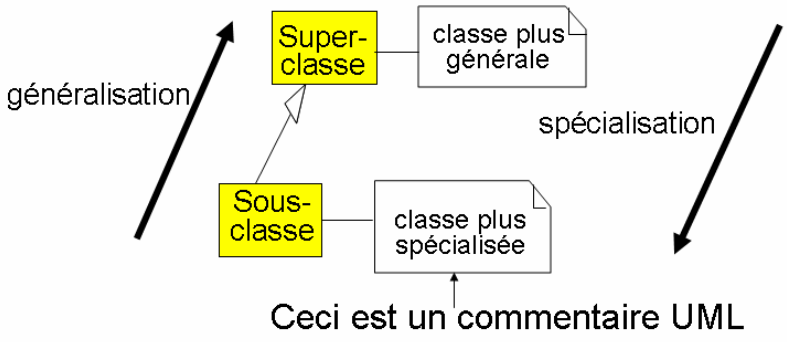
Type : 'moule' à instances (objets) ayant les mêmes propriétés et les mêmes comportements



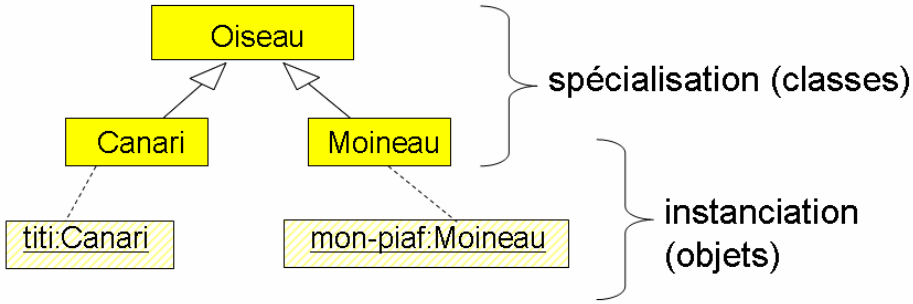
Module : interface visible + corps caché (utilisation possible sans connaître l'implantation; si le corps évolue sans impact sur l'interface le reste du système n'est pas touché)



La hiérarchisation des classes permet de gérer la complexité. Dans un sens, la **généralisation** correspond à la factorisation des éléments communs de classes (attributs, opérations) ce qui favorise la réduction de la complexité. Dans l'autre sens, la **spécialisation** permet d'adapter une classe générale à un cas particulier ce qui favorise la réutilisation et la modification incrémentielle.



Il ne faut pas confondre spécialisation et instanciation.

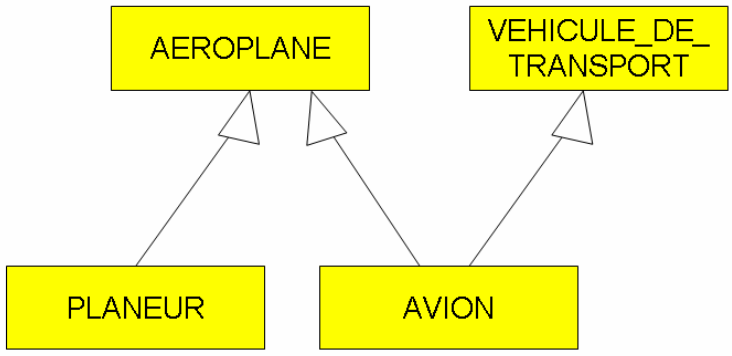


Notation des objets en UML: identificateur:classe

Les objets de la classe spécialisée **héritent** de la description des attributs (variables) et des opérations (méthodes) de la super-classe. Elles peuvent en **ajouter** d'autres et/ou en **redéfinir** certaines.

L'héritage peut être 'essentiel', s'il reflète une relation de spécialisation, ou 'accidentel', s'il est juste exploité pour réutiliser le code des opérations.

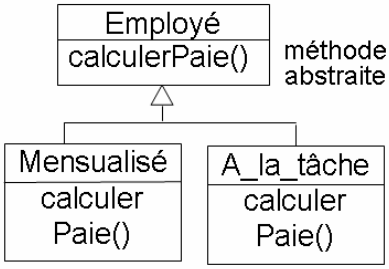
L'héritage multiple (plusieurs super classes) est autorisé dans la notation UML.



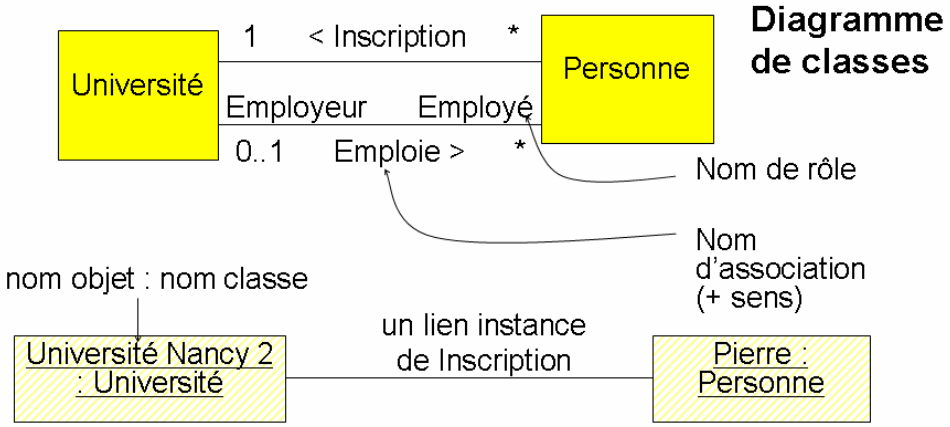
Les objets collaborent par 'envoi de messages' (appels d'opérations). Un même message peut être traité de manière différente selon la nature de l'objet receveur. On parle de **polymorphisme.** L'émetteur n'a pas besoin de connaître la classe du receveur.

Ex : on paye des employés de 2 types ('mensualisés' et 'à la tache'). Il suffit d'envoyer le message calculerPaie() à toutes les instances de la classe Employé. La bonne méthode est appliquée en fonction du **type effectif** de l'employé défini à la création de l'instance. Si un nouveau type d'employé apparaît le programme de paie n'a pas à être modifié:

pour tout e *dans* Employé *faire*
  e.calculerPaie();



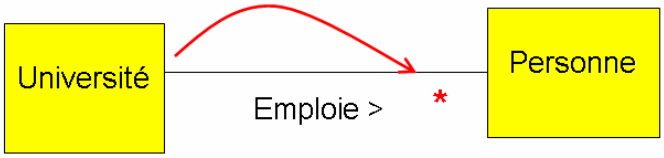
Une **association** exprime une connexion sémantique bidirectionnelle entre classes. Une association décrit un ensemble de **liens** (instances de l'association). Le rôle décrit une extrémité d'une association. Les cardinalités (ou multiplicités) indiquent le nombre d'instances d'une classe pour chaque instance de l'autre classe.



Remarque : ceci est un **diagramme d'objets.**

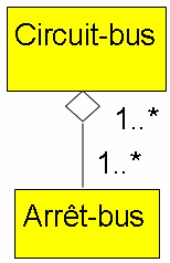
Multiplicités :

1	un et un seul	M..N	de M à N (entiers naturels)
*	plusieurs	1..*	de 1 à plusieurs
0..1	zéro ou un	0..*	de zéro à plusieurs

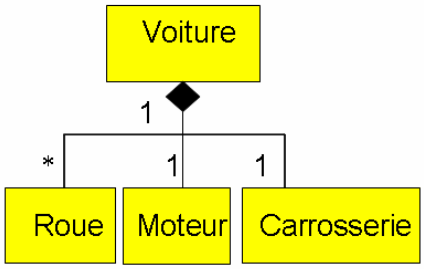


Contrairement aux cardinalités de Merise, les multiplicités UML sont placées du côté de la destination. Une Université emploie plusieurs Personnes : le symbole * est placé du côté Personne. A une Personne est associé zéro ou une Université (avec le rôle Employeur) : le symbole 0..1 est placé du côté Université.

L'agrégation est une association qui décrit une relation d'inclusion entre une partie et un tout (l'agrégat). Elle est réflexive, transitive, non symétrique. Si la relation d'agrégation est une **composition** (composant/composé avec des durées de vie liées pour les objets), elle est symbolisée par un losange plein ; sinon (pour des durées de vie indépendantes), elle est représentée par un losange vide du côté de l'agrégat.

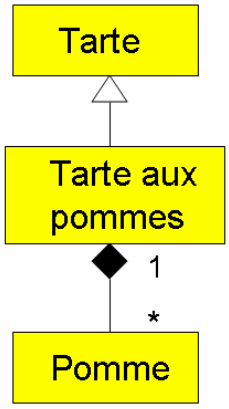


agrégation

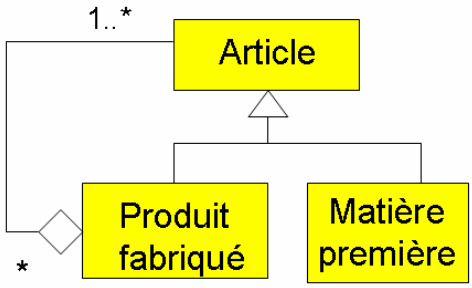


composition

Il ne faut pas confondre généralisation/spécialisation et agrégation. Quand une classe est une spécialisation d'une autre elle est **de même nature**, ce qui n'est pas le cas avec l'agrégation. Par contre ces relations sont souvent associées.



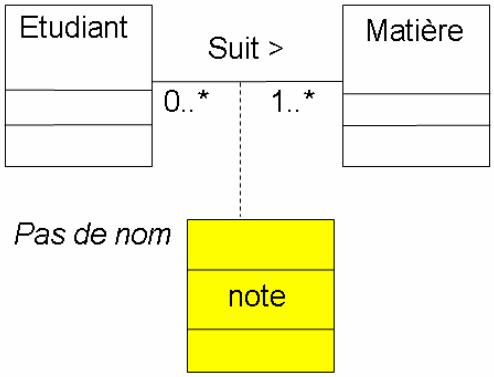
Une pomme n'est pas de même nature qu'une tarte !



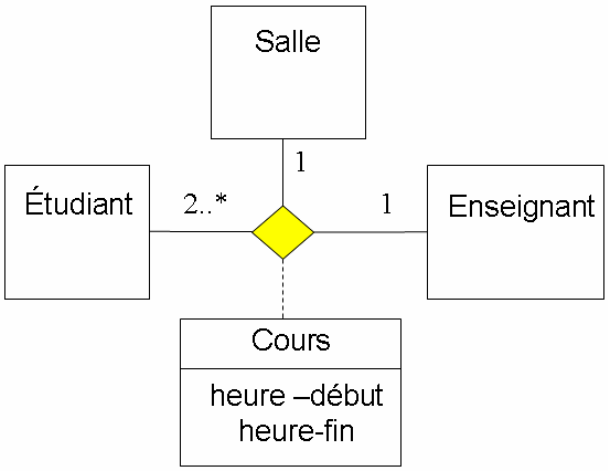
Un article est acheté (matière première) ou fabriqué à partir d'autres articles et/ou de matières premières).

Autres concepts

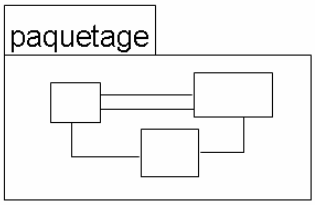
La **classe-association** : association porteuse d'attributs et/ou d'opérations, représentée comme une classe anonyme associée à l'association.



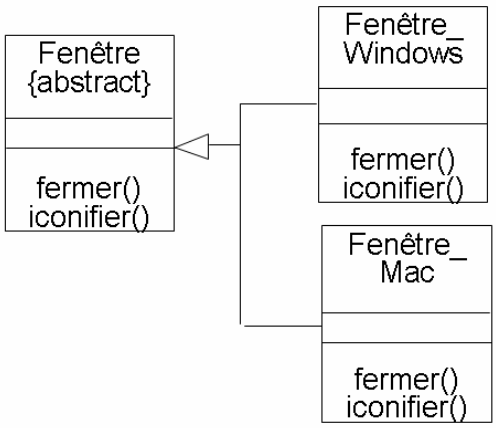
L'association d'arité n : représentée par un losange avec n 'pattes' auquel peut être associé une classe porteuse d'attributs et/ou d'opérations.



Le paquetage (package) : c'est une notion qui peut apparaître dans tous les diagrammes pour spécifier le regroupement d'éléments au sein d'un sous modèle (cas, classes, objets, composants, autres paquetages, ...).

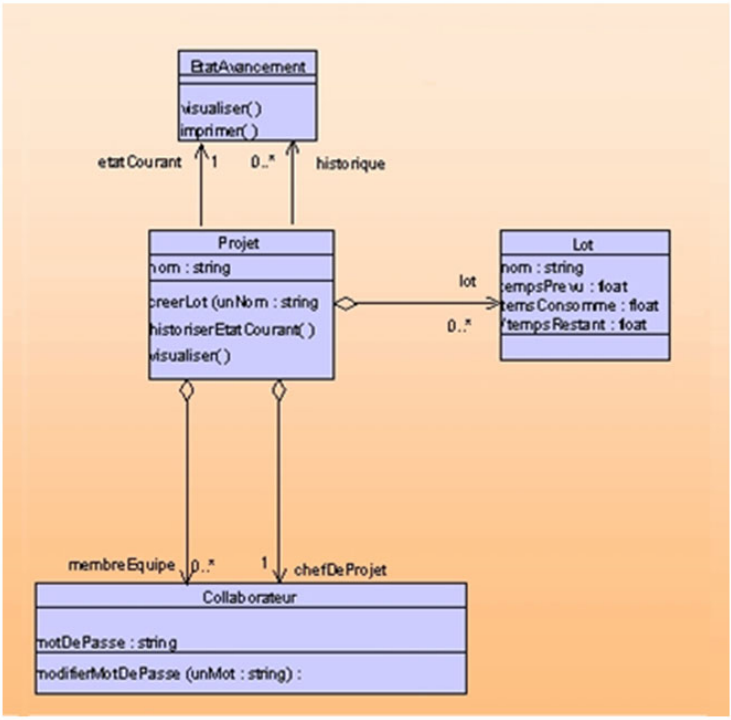


La classe abstraite : elle est non instanciable directement ; elle décrit des mécanismes généraux et laisse non décrit certains aspects (méthodes abstraites) ; elle est spécialisée par des classes concrètes (instanciables) qui précisent les méthodes abstraites.



L'interface : classe ne contenant que des opérations abstraites ; une interface précise les fonctionnalités que les classe qui les implantent doivent fournir.

Exemple de diagramme de classes



Les diagrammes d'interaction

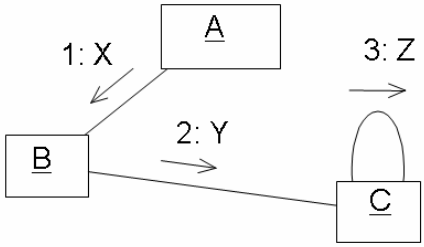
Une application est une société d'objets qui collaborent afin de réaliser les fonctions de l'application. Le comportement global d'une application repose donc sur la **communication** entre les objets qui la composent (échanges de **messages = appels de méthodes**). Ces collaborations sont spécifiées grâce à 2 types de diagrammes d'interaction : les **diagrammes de collaborations** et les **diagrammes de séquences**.

Ils sont utiles pour préciser la réalisation des cas d'utilisation au niveau de l'analyse ou pour spécifier en détail la dynamique d'un ensemble de classes ou d'objets au niveau de la conception ou de la réalisation.

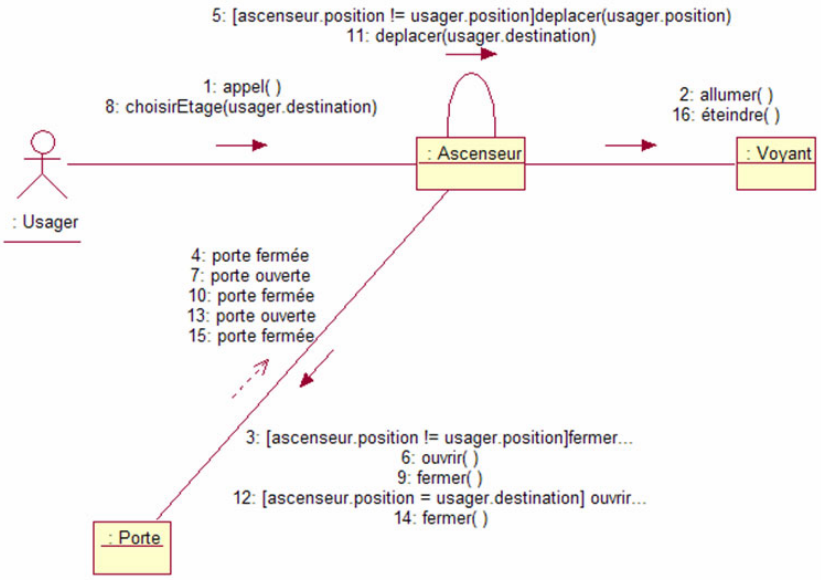
Les diagrammes de collaborations

Les diagrammes de collaborations mettent l'accent sur **«l'organisation spatiale»** des objets. Les **messages** peuvent être numérotés pour introduire une dimension temporelle. De nombreuses notations annexes permettent de caractériser les messages.

Exemple : Un objet A envoie un message X à un objet B, puis l'objet B envoie un message Y à un objet C, et enfin C s'envoie à lui même un message Z

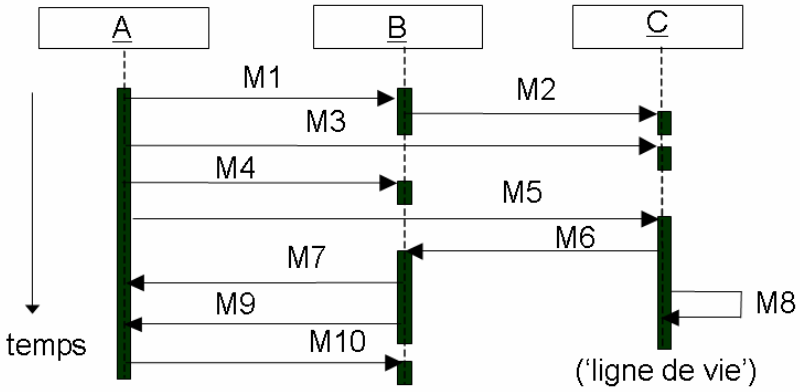


Exemple : ascenseur (appel externe)

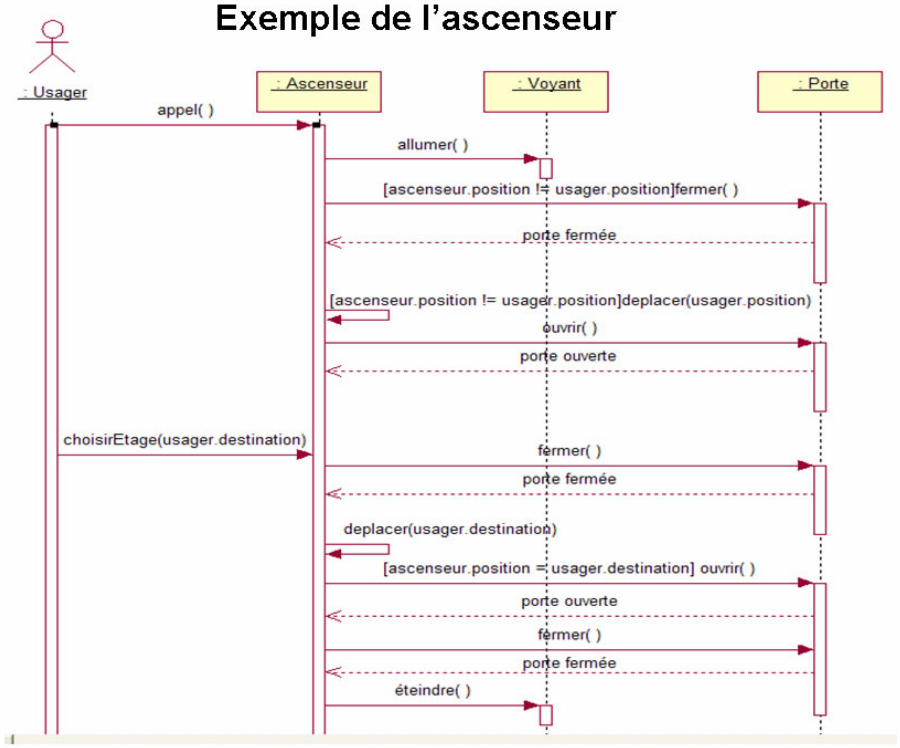


Les diagrammes de séquences

Ils mettent l'accent sur l'organisation **temporelle**. De nombreuses notations annexes permettent de préciser la nature des messages (appel de procédure, simple signal, message répétitif, conditionnel, réflexif, récursif, etc.) et les données véhiculées.



Exemple de l'ascenseur



Les diagrammes d'états et d'activités

Ils servent à préciser comment le système évolue au cours du temps, autrement dit sa **dynamique**.

Nous ne détaillons pas toutes les notations associées à ces diagrammes qui sont plutôt complexes.

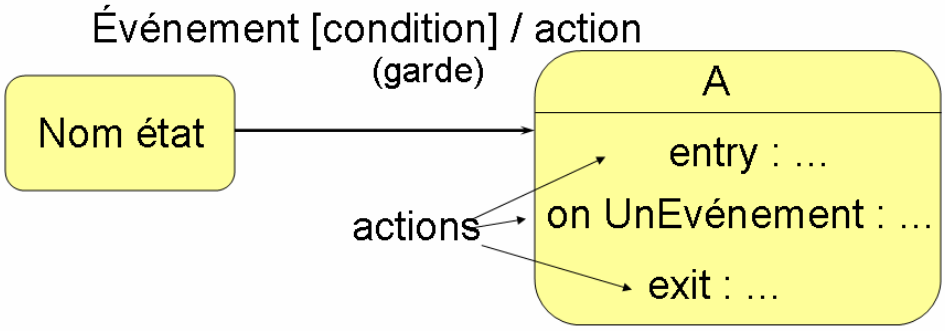
Les diagrammes d'états

Les diagrammes d'états servent à décrire le comportement des objets (classes) complexes.

Ces diagrammes décrivent **tous les états possibles de l'objet et les transitions possibles entre ces états (cycle de vie des objets)**.

On les appelle aussi parfois 'automates' ou 'diagrammes de transitions'.

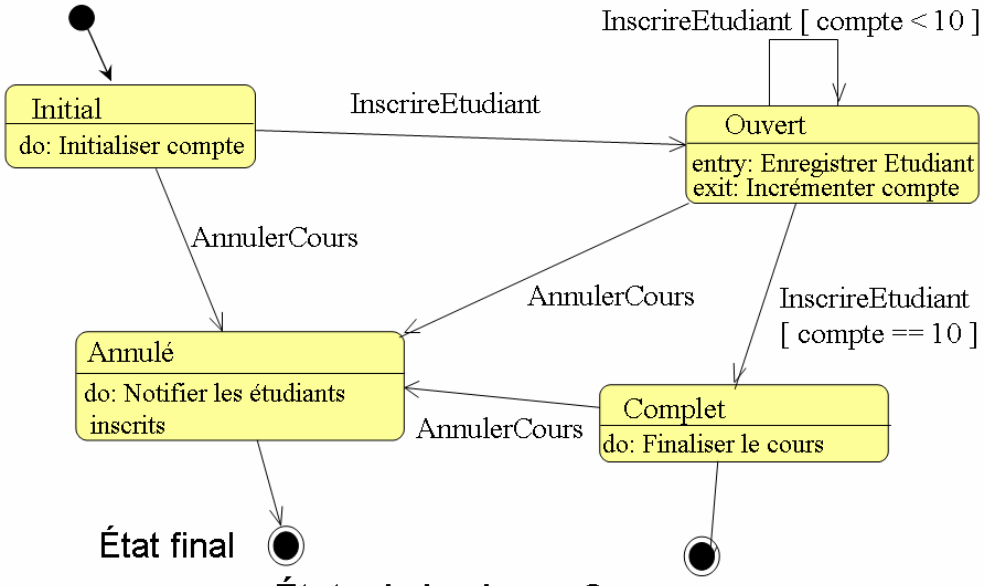
En UML on peut faire figurer des **conditions** (gardes) sur les transitions et des **actions** sur les transitions et sur les états (au début de l'état, pendant l'état, à la fin fin de l'état).



État de départ Transition État d'arrivée

- État initial
- État final

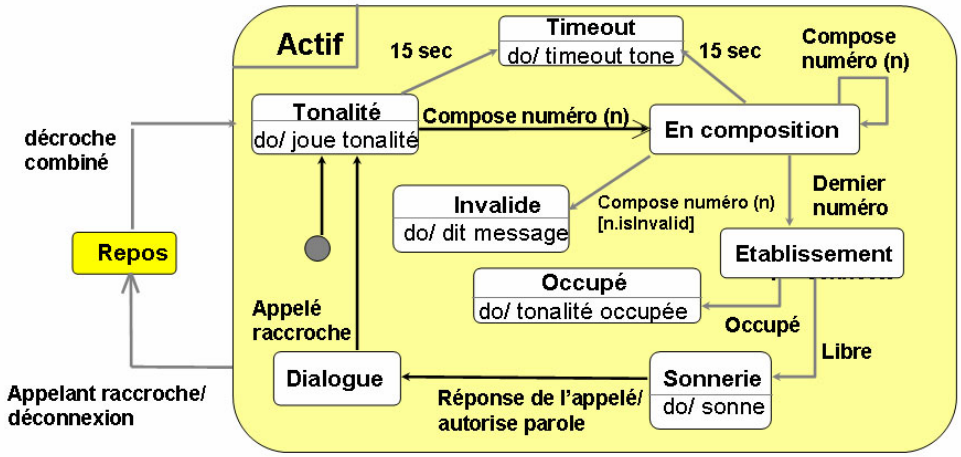
Exemple 1 : diagramme simple



États de la classe Cours

Les transitions correspondent aux appels de méthodes

Exemple 2 : téléphone (diagramme avec décomposition d'état)

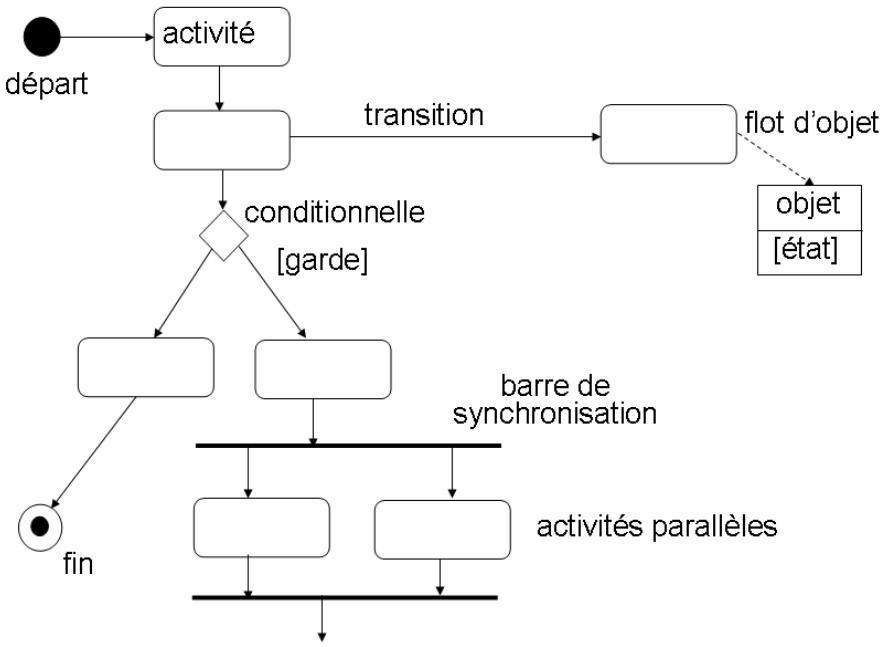


Les diagrammes d'activité

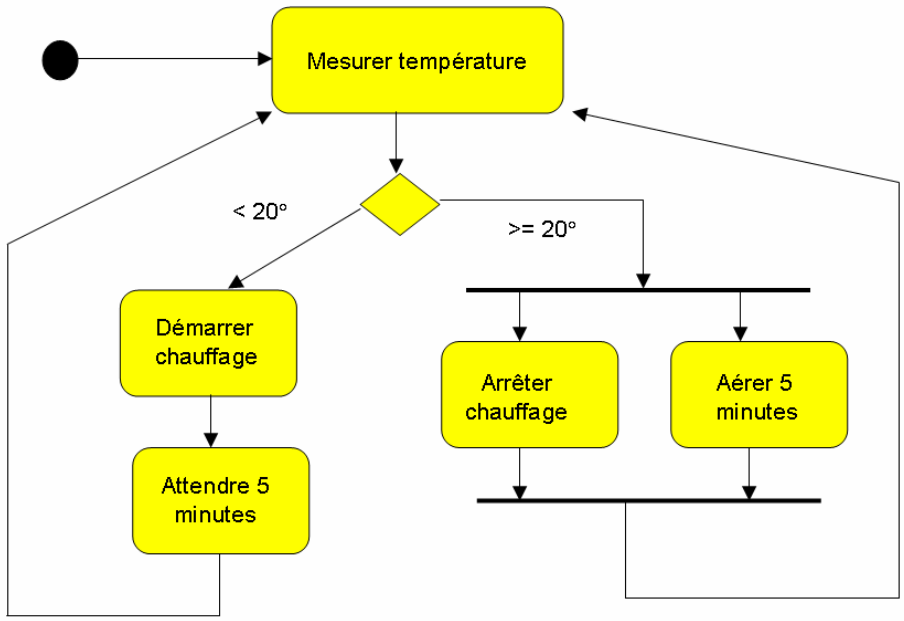
Ils permettent de décrire le flot de contrôle entre opérations (séquence, choix, itération, parallélisme). Il s'agit en gros d'organigrammes incluant éventuellement du parallélisme.

A un niveau macroscopique, les diagrammes d'activité permettent de décrire des **enchaînements de fonctionnalités**. Ils complètent donc bien les cas d'utilisation au niveau de l'analyse des besoins.

A un niveau microscopique, les diagrammes d'activité permettent, par exemple, de décrire **l'algorithme d'une action** d'un diagramme d'états.

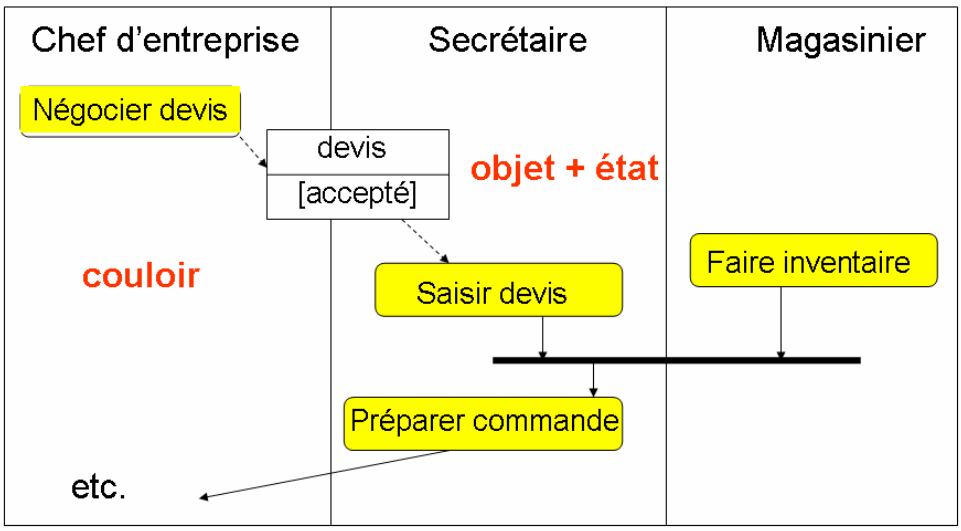


Exemple



Comparaison avec Merise

Des extensions aux diagrammes d'activité ont été proposées pour représenter aussi l'organisation (style MOT avec des 'couloirs' ou 'swimlane').



Traduction schéma de classes vers schéma relationnel

1. Traduction des classes

- Ajouter une clé artificielle (ex: N°…) si une clé naturelle n'existe pas.
- Inclure les attributs dans la relation.
- Les méthodes sont exprimées dans un langage procédural; on peut utiliser les procédures stockées; on peut avoir des attributs calculés.

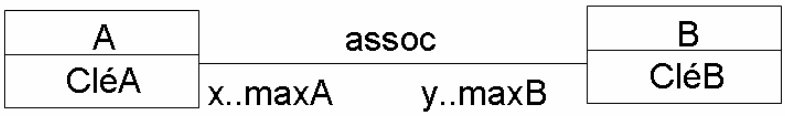
Ex:

Rectangle
largeur hauteur
agrandir(..) donnerAire()

Rectangle(N°rect, largeur, hauteur [,aire])

2. Traduction des associations

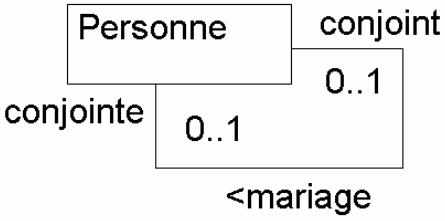
La solution dépend des multiplicités :



	maxA = 1	maxA > 1
maxB = 1	une des autres solutions	CléB dans A comme clé étrangère
maxB > 1	CléA dans B comme clé étrangère	créer relation assoc avec comme clé CléA et CléB

Exemples:

Département (N°Dep, …)
Ville(NomVille, N°Dep, …)
Il faut bien entendu que Ville.N°Dep ⊆ Département.N°Dep



Personne(N°Pers, N°PersConjoint, …) avec valeurs nulles
ou Personne(N°Pers, …)
et Mariage(N°PersConjoint, N°PersConjointe)

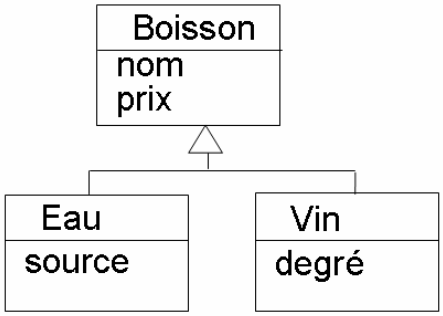
Auteur(N°Auteur, …)
Livre(N°ISBN, …)
Écrit(N°Auteur,N°ISBN, …)
Il faut bien entendu que Ecrit.N°Auteur⊆Auteur.N°Auteur
et que Ecrit.N°ISBN⊆Livre.N°ISBN
(contrainte d'intégrité référentielle)

Agrégation et composition se traduisent comme des associations (avec des delete en cascade dans le cas de la composition !).

3. Traduction de la généralisation

Solution 1 : tout dans la même table avec un attribut définissant le type et les attributs non utilisés à null

Ex :



Boisson(nom, type, source, degré, …)
avec type∈{eau, vin}
On peut créer une vue pour retrouver la bonne description :

```
CRETE VIEW Eau AS
  SELECT nom, prix,
     source
  WHERE type = 'eau'
```

Solution 2 : une relation par classe; le type est défini par le présence dans une relation donnée.

Boisson(nom, prix, …)
Eau(nom, source, …)
Vin(nom, degré, …)

On peut créer une vue pour retrouver la bonne description :

```
CREATE VIEW Eau1 AS
  SELECT e.nom, b.prix, e.source
  FROM Boisson b, Eau e
  WHERE b.nom = e.nom
```

Le processus de développement objet

UML est un langage de modélisation objet. Il n'impose pas de processus particulier et reste compatible avec différents styles (méthodes) de développement.

Cependant, une certaine démarche intellectuelle est sous jacente à la notation.

1) Les **cas d'utilisation** et **scénarios** expriment un point de vue fonctionnel sur le système. Ils constituent clairement la base de la spécification initiale du système.

2) Puis, les **diagrammes de collaborations et de séquences** que l'on peut associer aux cas d'utilisation font apparaître des classes d'objets candidates et donc passer d'une vue fonctionnelle à une vue objets. On «raconte les cas» en faisant communiquer des classes d'objets, **introduites au fur et à mesure**.

3) Les **schémas de classe (et d'objets)** sont d'abord esquissés à partir de ces «classes candidates» issues des cas d'utilisation. Il s'agit de classes caractéristiques du problème (classes 'de base', ou classes 'du domaine' ou classes 'métiers'). Elles peuvent être regroupées en plusieurs **paquetages.** On parle d'analyse du domaine.

Il existe d'autres techniques pour trouver des classes candidates comme :

- l'analyse des termes qui apparaissent souvent dans les documents décrivant les besoins,
- les abstractions importantes du domaine signalées par les experts du domaine, les clients, les utilisateurs,
- les patrons d'analyse ('design patterns'), etc.

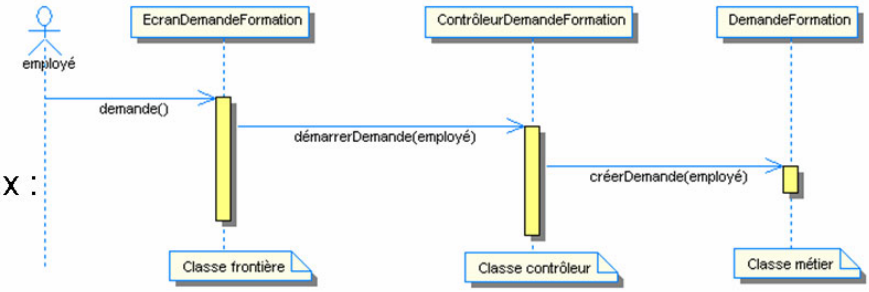
D'autres éléments apparaissent en exploitant les diagrammes de comportement (activités, états).

4) **Les schémas de classe** sont enrichis au fur et à mesure où des éléments nouveaux apparaissent.

Ces éléments sont obtenus d'abord par un **raffinement de l'analyse**. On passe également de l'analyse du domaine à l'analyse de l'application (fonctionnalités retenues, dialogue avec le système, logique de l'application, …).

On complète souvent les classes métiers (modèles) par des classes frontières (vues, écrans) et des classes contrôleurs pour gérer les interactions (schéma MVC).

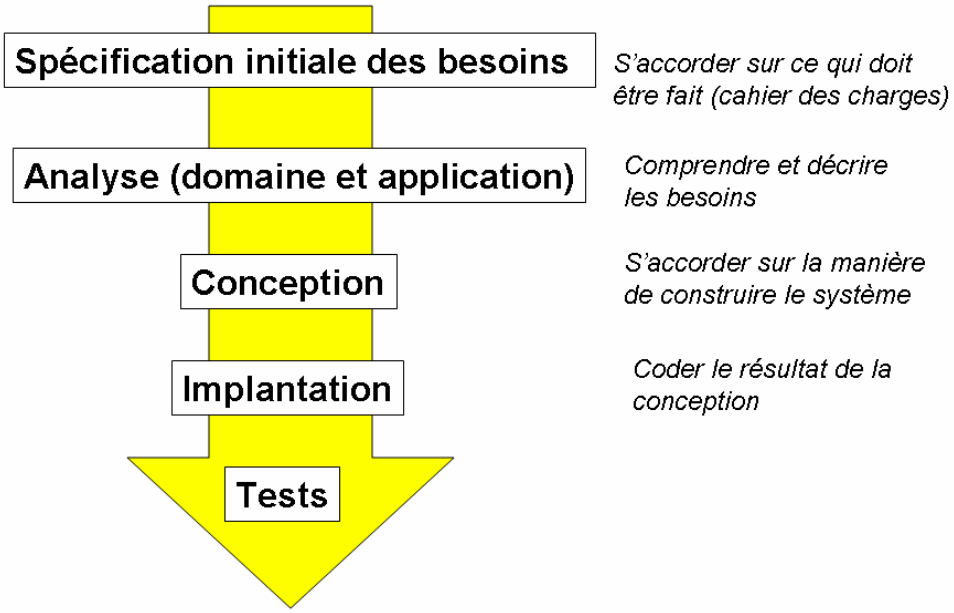
Ex :



5) Ultérieurement, au niveau de la phase de conception sont introduites une **architecture** en packages (ex : architecture en couches) et **des classes techniques** liées :

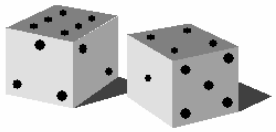
- à la persistance des données (fichiers, sérialisation des objets, interfaçage de bases de données relationnelles ou objets, …),
- à la concurrence des traitements et à la distribution des classes sur le réseau (middleware RMI, CORBA, SOAP, …), etc.

Les modèles sont encore raffinés (algorithmes) et optimisés (chemins d'accès aux classes, …).



Développement linéaire, incrémental, par prototypage, etc.

Un ‘exemple jouet’

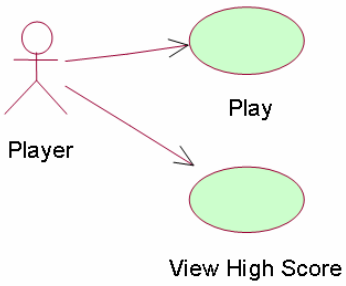


- Il s’agit d’un jeu de dés.
- Le joueur lance 10 fois 2 dés.
- Si le total fait 7, il marque 10 points à son score.
- En fin de partie, son score est inscrit dans le tableau des scores (‘high score’).

La spécification des besoins

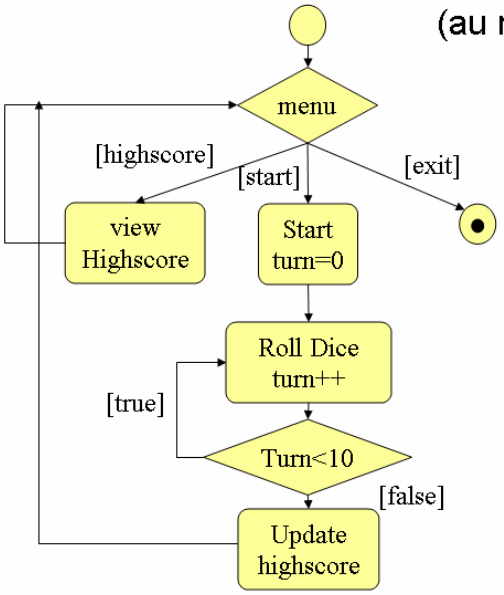
- Décrire les besoins des utilisateurs du système.
- Centrée sur les cas d’utilisation.

Premier cas d'utilisation



- Cas 'Play' :
  Description : le joueur lance 10 x les dés; à chaque fois que le total fait 7, +10 pts.
- Cas 'View High Score'
  Description : Le joueur consulte en lecture seulement les high scores.

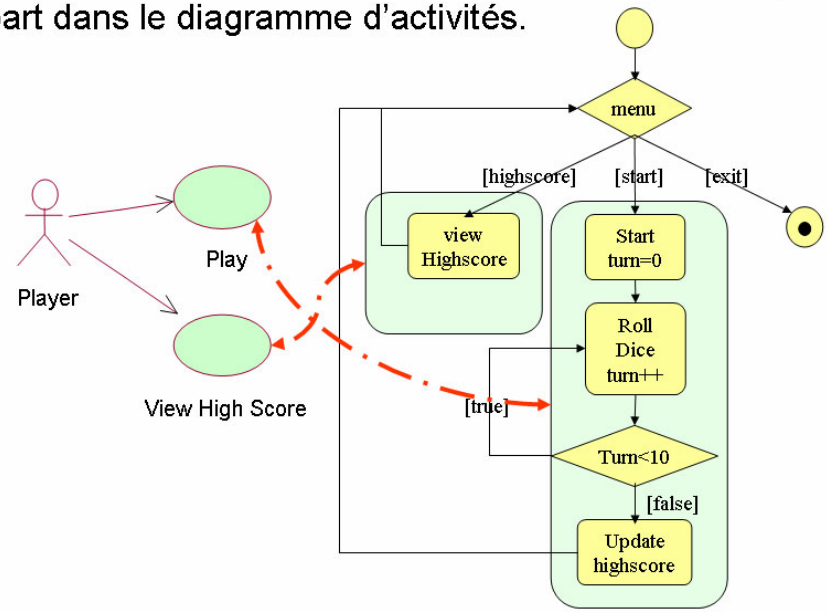
Diagramme d'activités



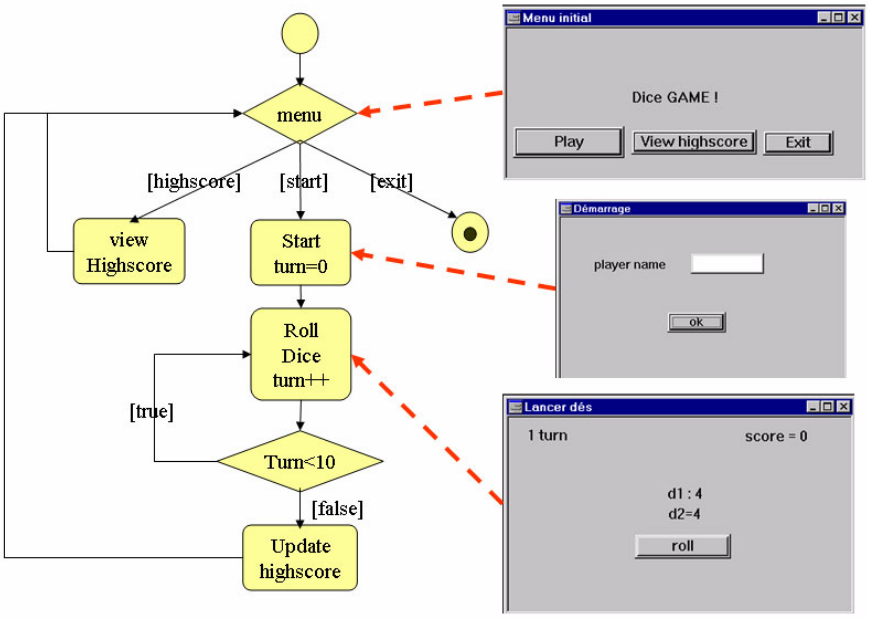
(au niveau **macroscopique** : enchaînement des fonctionnalités)

Sa construction permet de décrire l'organisation générale des traitements et facilite le dialogue avec les utilisateurs.

Il doit exister une certaine cohérence entre les diagrammes : tous les cas doivent se retrouver quelque part dans le diagramme d'activités.



On peut compléter ce diagramme par des maquettes d'écran associées aux différentes opérations.

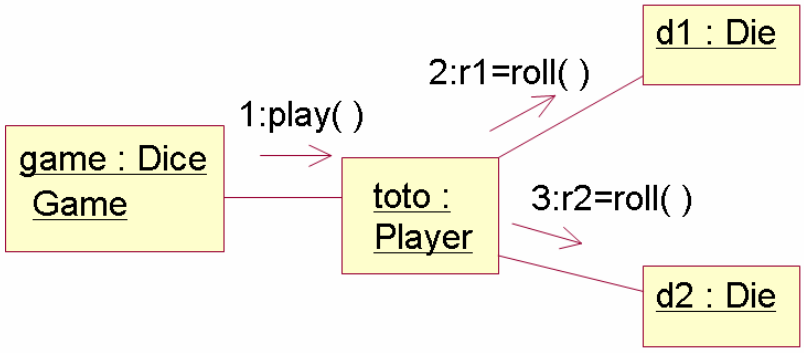


L'analyse du système

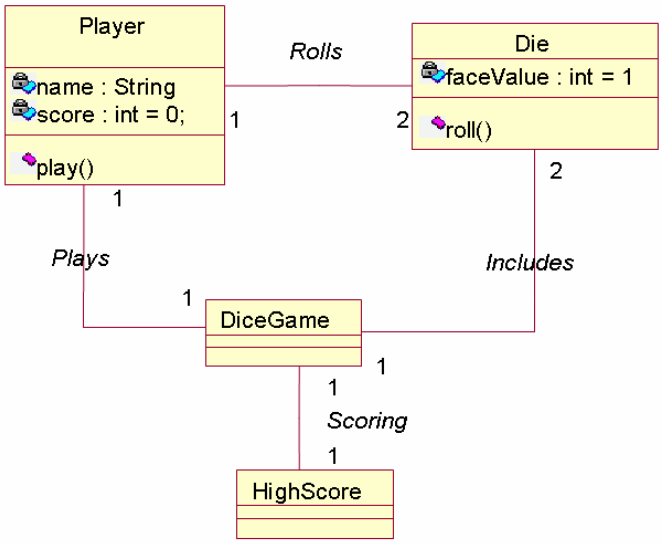
Définition des classes du domaine et de leur dynamique.

Premier diagramme de collaboration

«Il s'agit d'un jeu de dés. Le joueur lance 10 fois 2 dés» => objets (jeu, joueur, dés), relations (joue, lance) et ordonnancement des messages



Premier diagramme de classes



Cohérence des diagrammes :

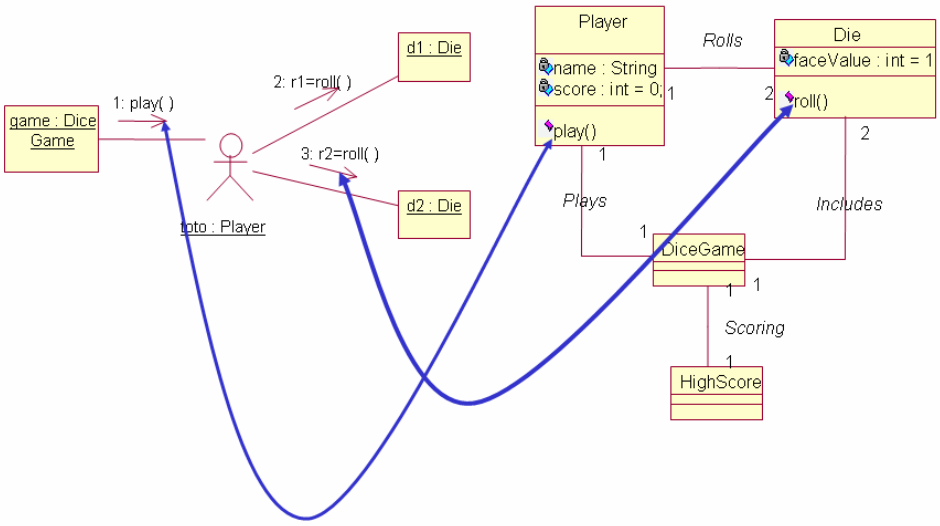
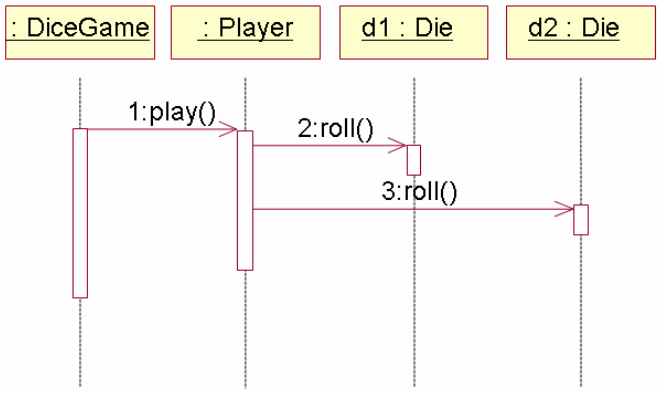


Diagramme de séquence



Enchaînement des messages.

Il faut aussi modéliser la création des objets en début de partie. Le joueur (:Player) n'est créé qu'au démarrage de la partie. On aurait aussi pu le faire à la création du jeu. L'avantage est qu'on peut changer de nom à chaque partie!

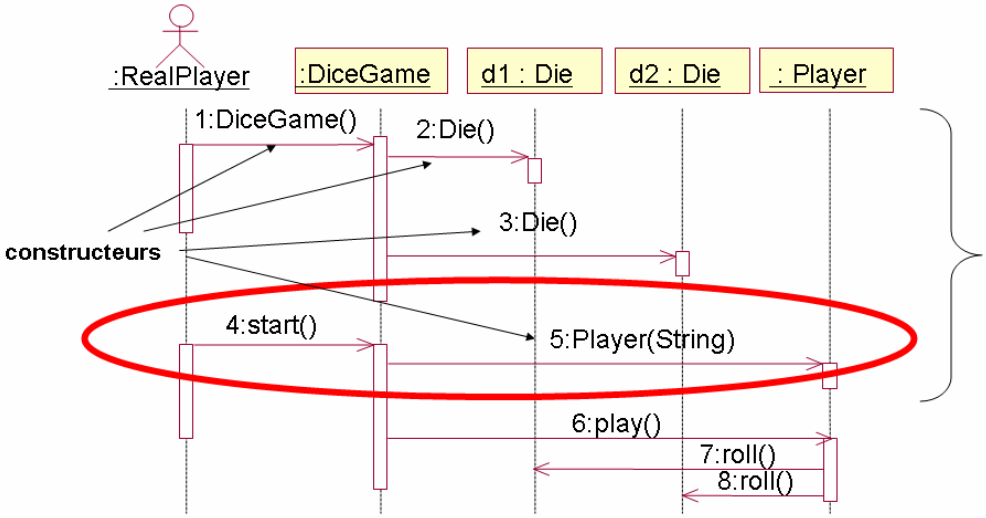
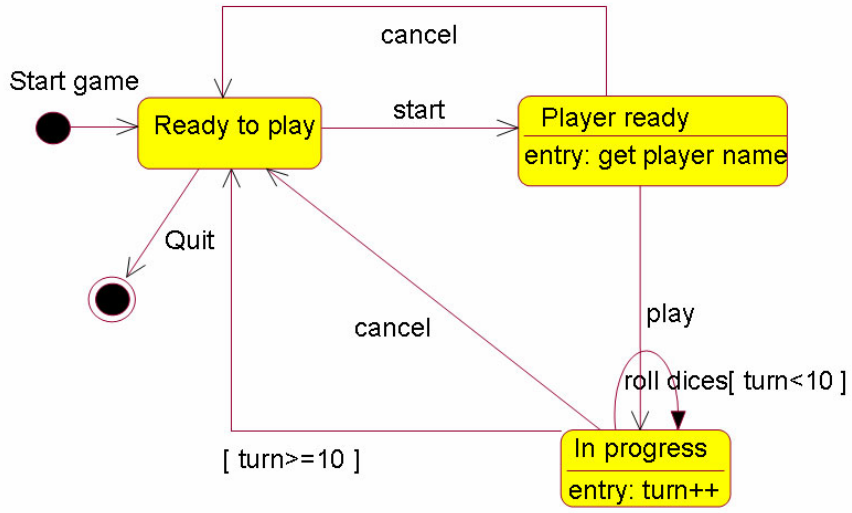
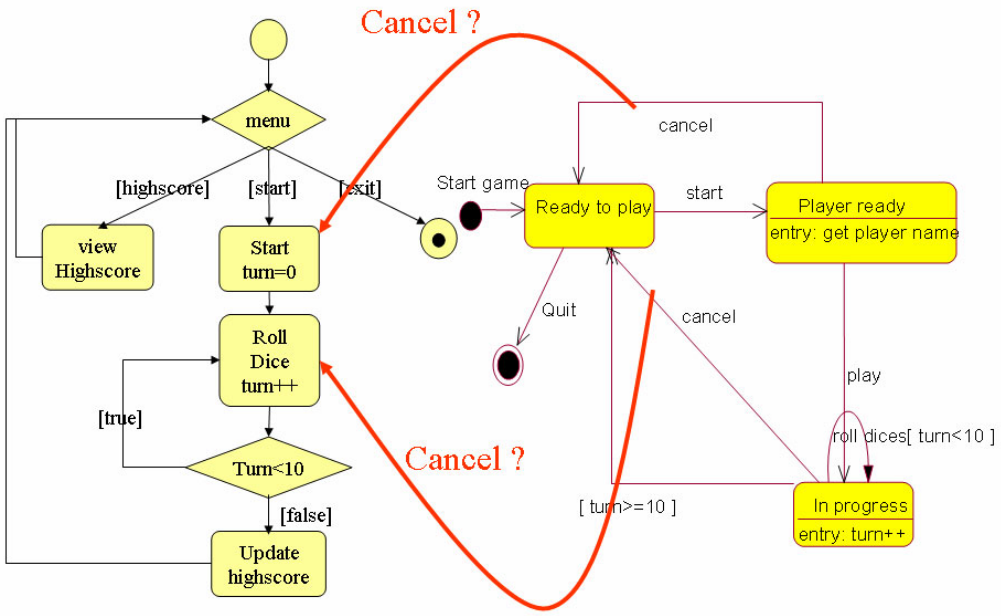


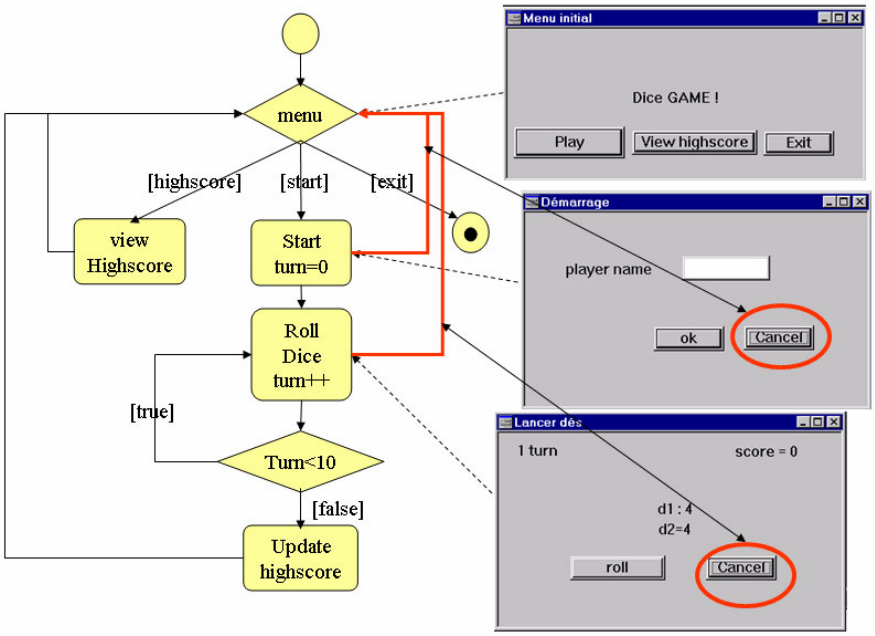
Diagramme d'états d'un objet «partie» (:DiceGame)



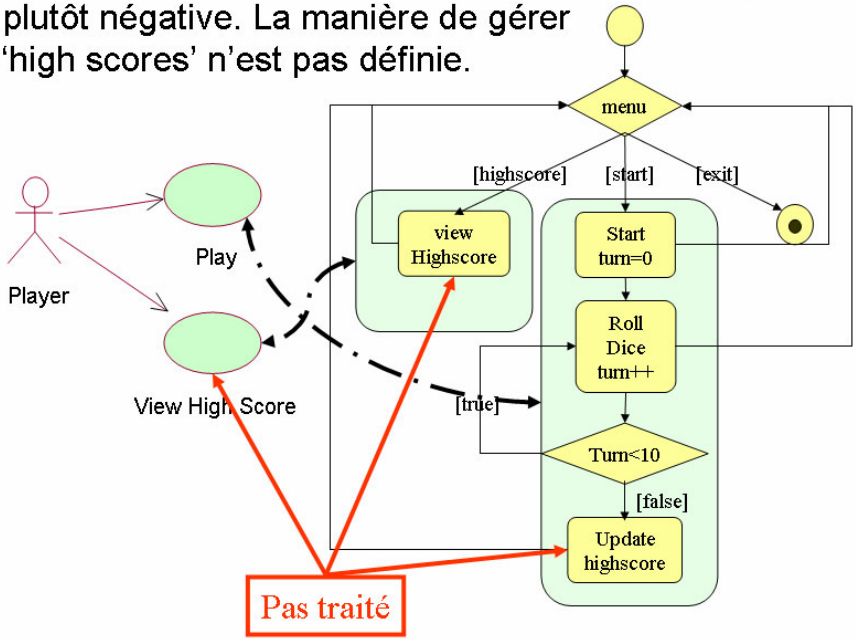
On détecte des incohérences entre diagrammes :



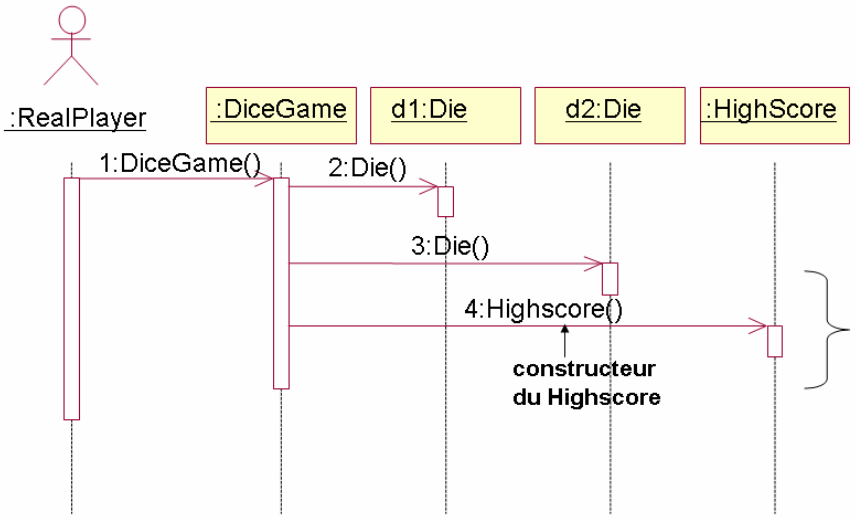
On modifie l'analyse pour les prendre en compte :



L'analyse est-elle complète ? Il faut se demander si les cas d'utilisation sont correctement couverts. La réponse est plutôt négative. La manière de gérer les 'high scores' n'est pas définie.



Il faut reprendre les schémas pour gérer les 'high scores' (constructeur et contenu).



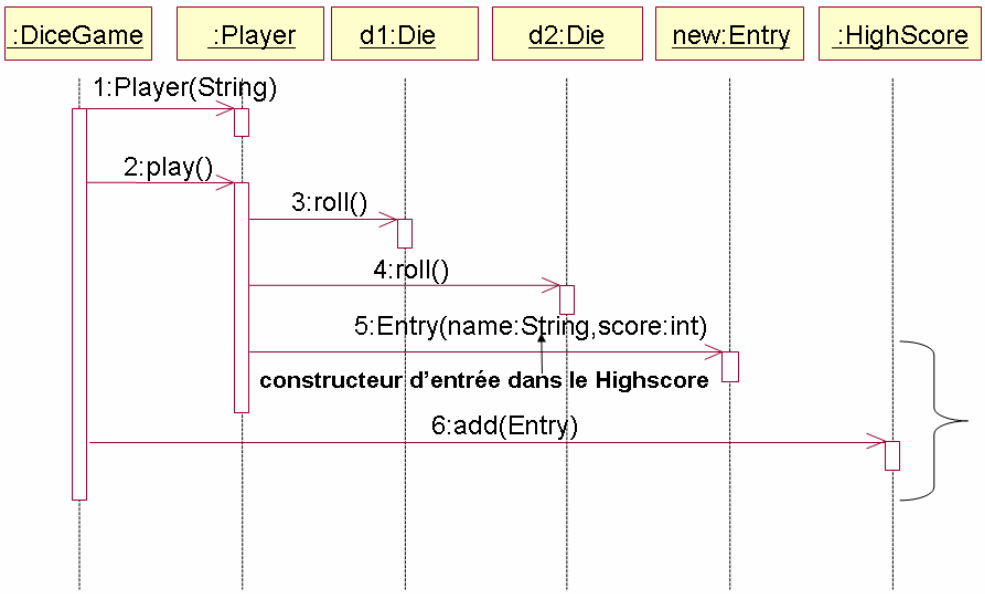
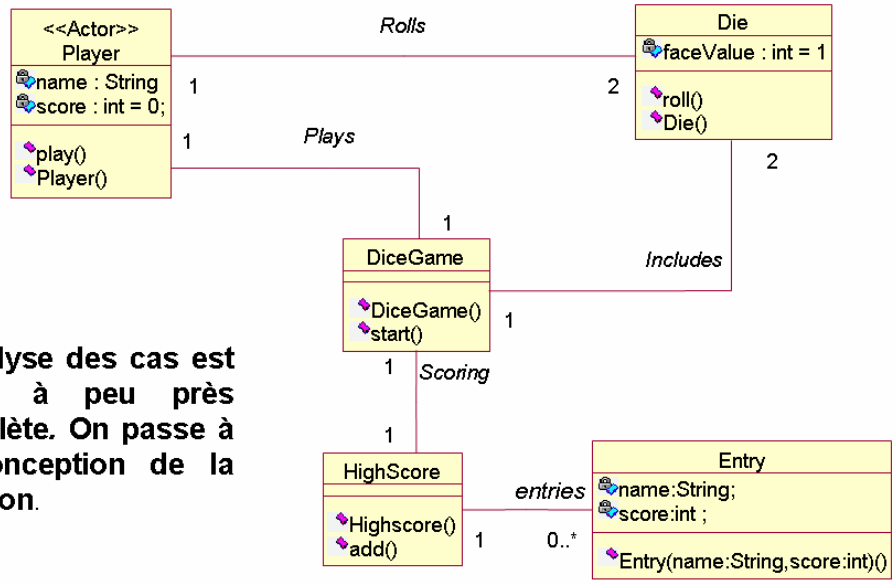


Diagramme de classes corrigé



L'analyse des cas est jugée à peu près complète. On passe à la conception de la solution.

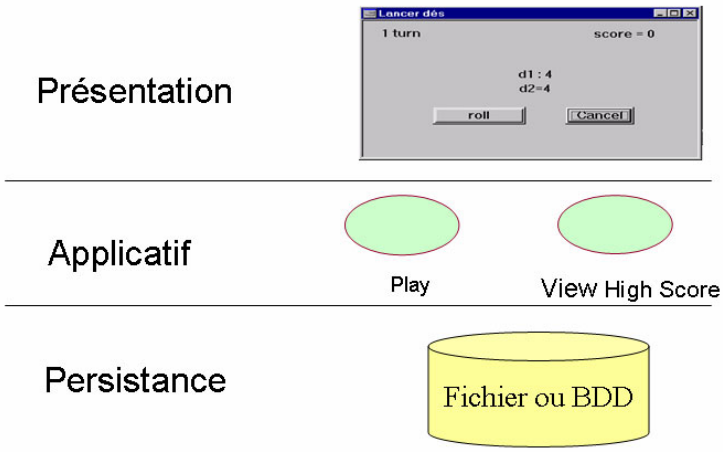
La conception

Concevoir l'implantation : architecture logique, architecture physique, interfaces utilisateurs, persistance des 'high scores', etc.

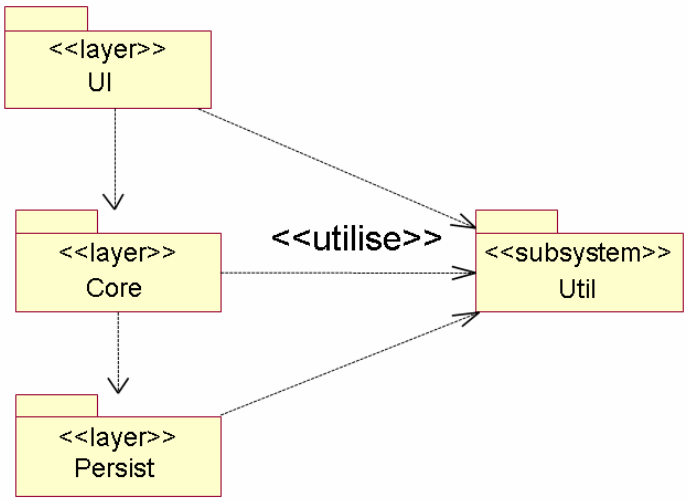
Ajouter les classes 'techniques' permettant d'implanter cette solution.

Conception architecturale (logique)

Organisation classique en 3 couches :

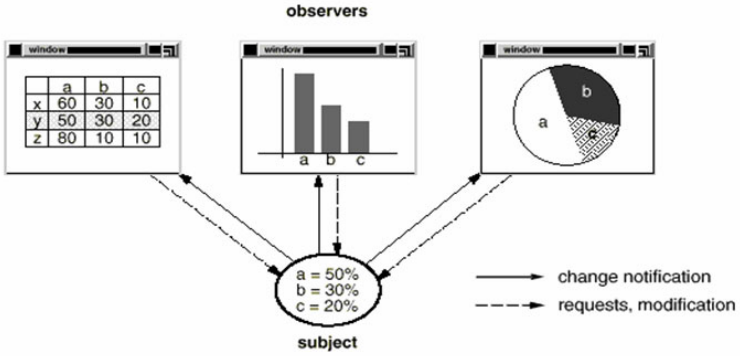


On fait correspondre à cette architecture des packages et des dépendances. Le package Util regroupe tout ce qui ne se rattache pas naturellement aux autres. On indique qui utilise qui.



Chaque couche est ensuite analysée du point de vue technique par exemple en reprenant des schémas de classes 'en kit' pour répondre à des problèmes récurrents de conception ('design patterns' ou patrons de conception).

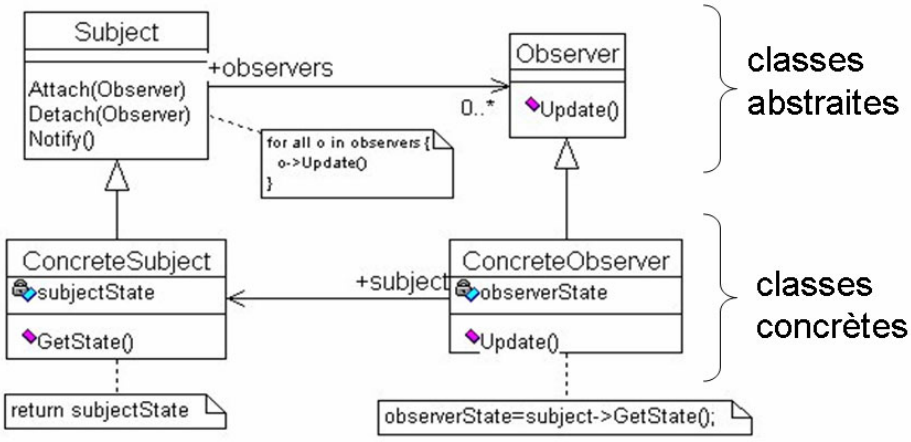
Exemple : le pattern 'Observer' qui répond au besoin d'un lien 1 vers n entre objets. Tout changement du sujet doit être automatiquement notifié à tous les observateurs (dont le nombre est inconnu au départ).



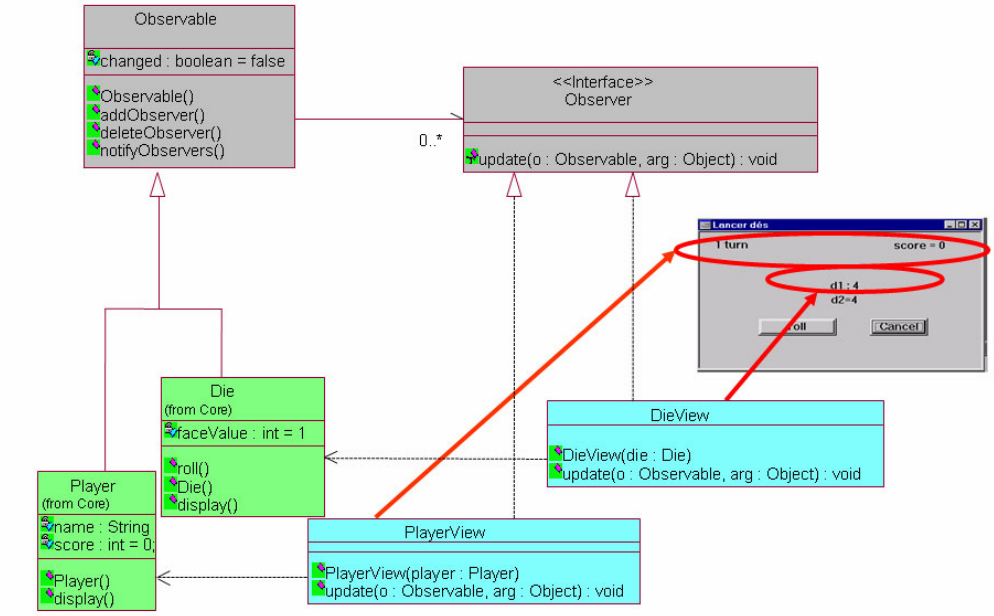
Le pattern 'Observer'

• Lorsqu'un sujet est modifié, il doit avertir (méthode Notify) les observateurs qui se sont enregistrés (méthode Attach).

• Lorsqu'un observateur est averti de la modification d'un sujet, il se met à jour (méthode Update).



Application du pattern au jeu de dés



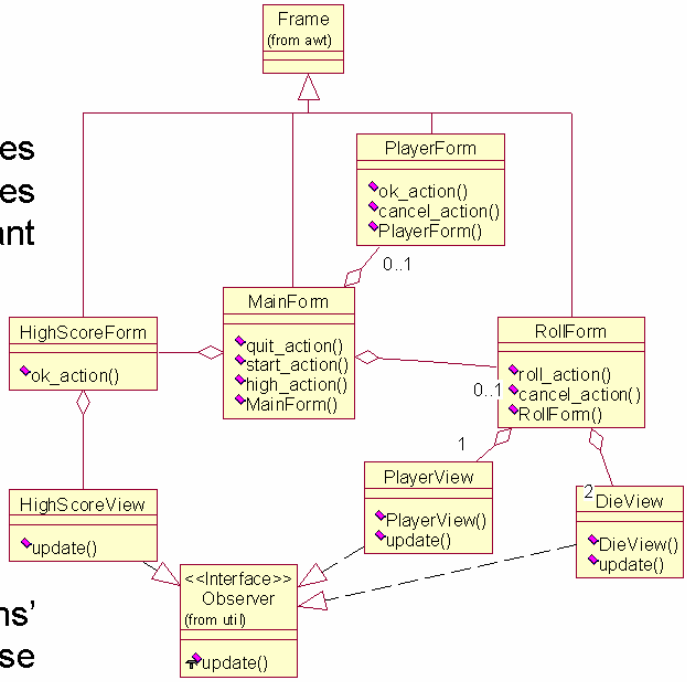
classes de base (subject) panneaux affichage (observers)

Nous ne détaillons pas les autres aspects techniques de la conception pour la couche Core.

Pour la couche Persist il faut choisir comment sauvegarder les scores : fichier (sérialisation de la table des entrées) ou base de données via JDBC. Cela donne de nouvelles classes et diagrammes de séquences.

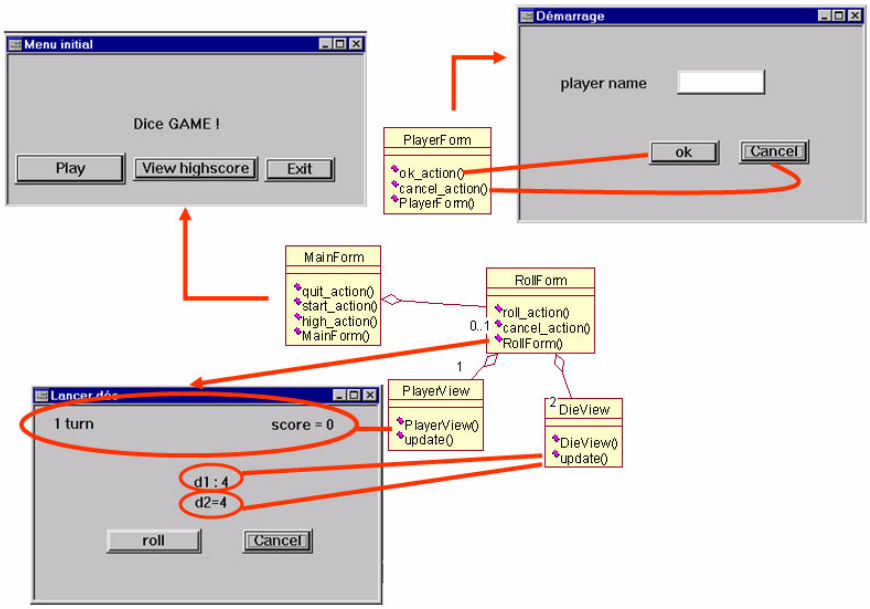
Enfin, la couche UI (interface utilisateur) comprend un ensemble de classes correspondant aux fenêtres (Form) et un ensemble de classes correspondants aux panneaux dans ces fenêtres susceptibles d'évoluer au long du jeu.

Il faut définir les fenêtres graphiques ('forms') contenant des panneaux ('views').

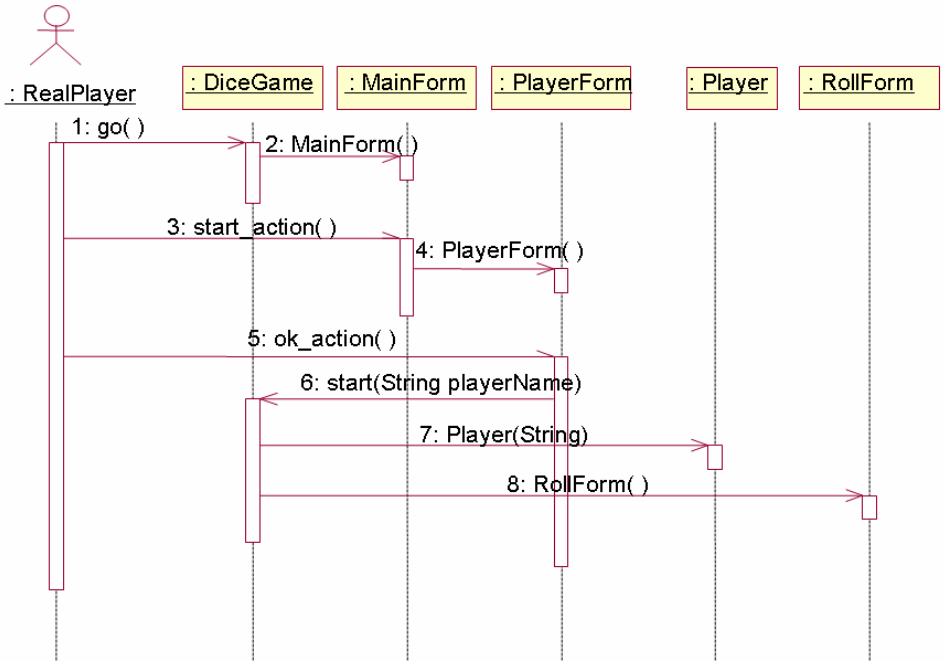


Toutes les 'forms' héritent de la classe java.awt. Frame.

Liens entre 'forms' (fenêtres) et 'views' (panneaux) :

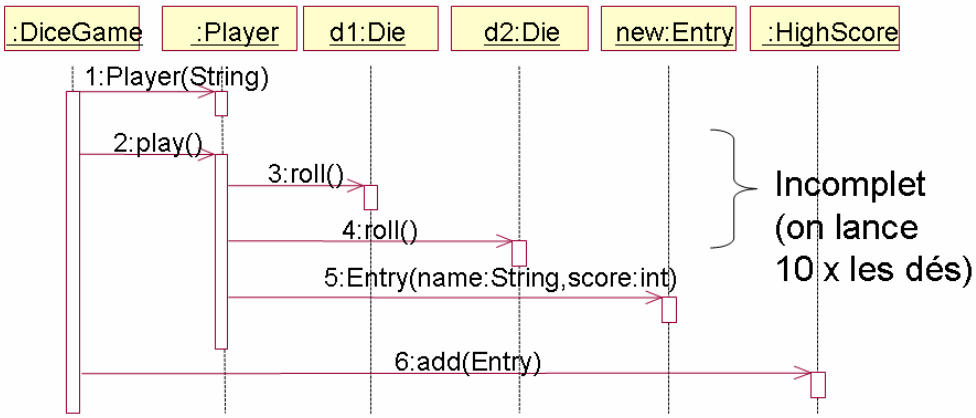


La description détaillée de ces composants est décrite avec les diagrammes de séquence.

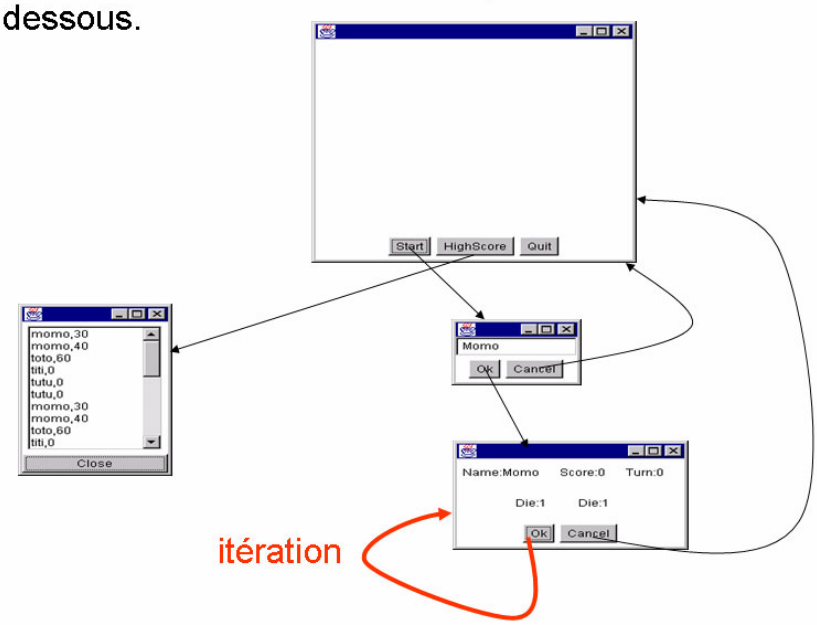


La séparation entre analyse et conception est un peu factice. Par exemple, on peut s'apercevoir à ce stade que certains aspects du problème n'ont pas été assez spécifiés lors de l'analyse. Le **développement** est en réalité **itératif.**

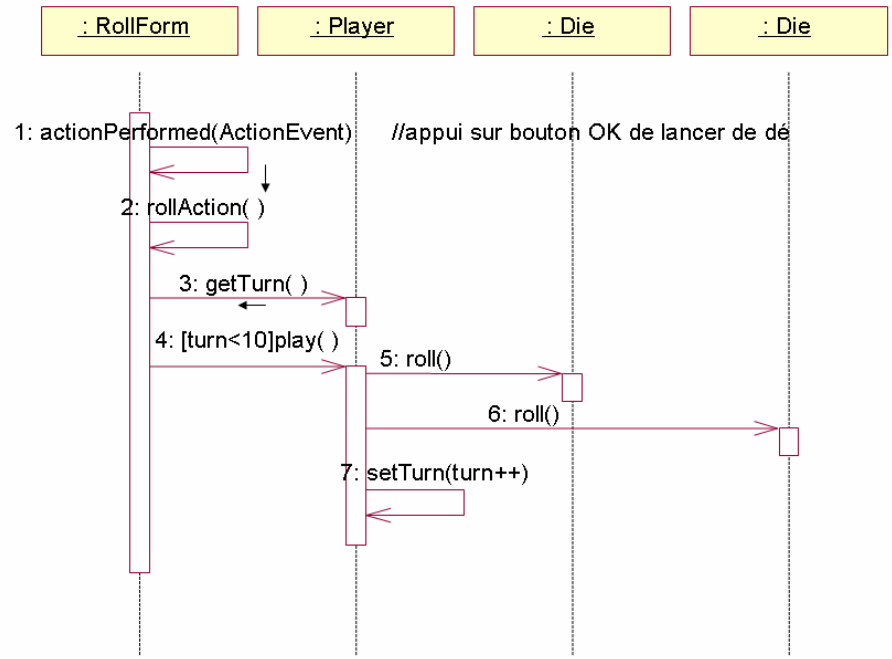
Exemple :



Ce schéma ne gère pas les 10 tours de jeu et la fin du jeu. La version remaniée doit correspondre au fonctionnement ci dessous.



Soit le diagramme de séquence complété suivant :



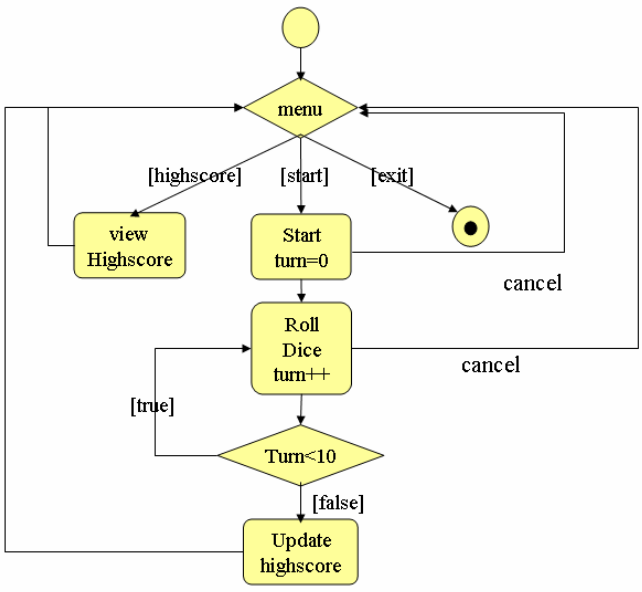
Les tests

Il faut d'abord tester toutes les méthodes, classe par classe (tests unitaires).

Puis il faut tester les composants (tests d'intégration), et enfin, le système tout entier (test d'acceptation au regard des use case et du diagramme d'activités définissant les besoins initiaux).

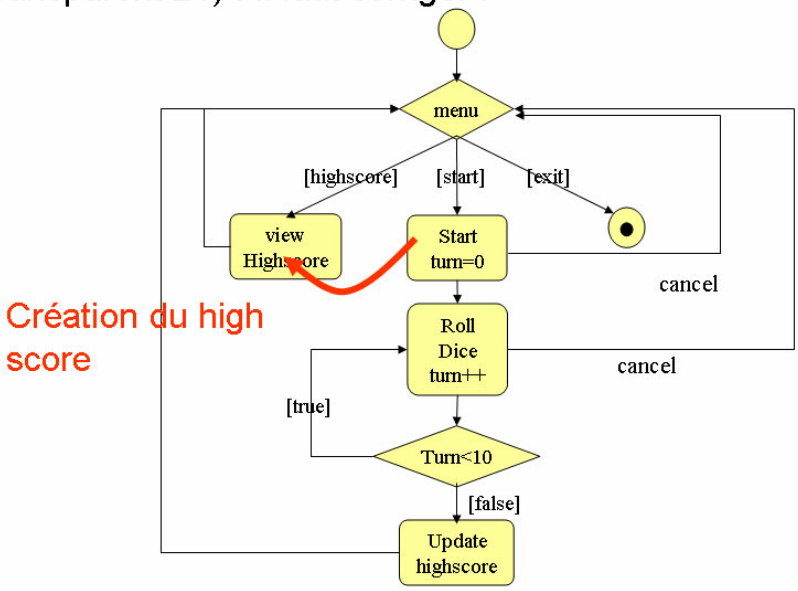
Vis à vis des cas d'utilisation et de leur description tout semble correct.

Vis à vis du diagramme d'activités, il faut tester tous les chemins possibles. Regardons ce point.



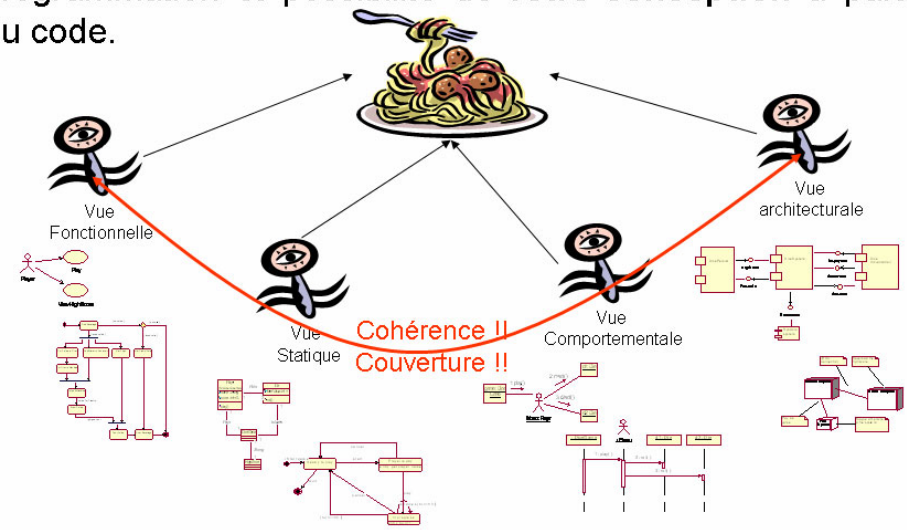
Le chemin 'normal' start, roll*, highscore, exit marche correctement. Par contre le chemin highscore, exit plante !

Il s'agit d'un problème de (mauvaise) conception. En effet, c'est DiceGame qui crée Highscore lors du start (cf. transparent 21) . Il faut corriger !



Création du high score

Conclusion UML permet d'avoir **plusieurs points de vue** à chaque étape et de réfléchir en termes de **couverture des besoins et de cohérence**. Il y a **continuité dans les moyens d'expression** tout au long du projet jusqu'à la programmation et possibilité de **rétro-conception** à partir du code.



Conclusion

Merise et UML

- Merise est une méthode de conception de SI plus orientée vers la compréhension et la formalisation des besoins et la description de la solution (acteurs/ flux, MCT/MOT, MCD/ MLD) que vers la production de logiciel .

  Merise reste très utilisée pour la **modélisation des données en vue de la construction d'une BD relationnelle** (MCD, MLD) (ex: applications de gestion classiques ou en intranet)
- Au contraire, UML est adapté pour **concevoir et développer des systèmes logiciels développés dans des langages objets** (ex: applications java/J2EE ou VB/C#/.net).

- Certes UML permet aussi de modéliser des données (modèle de classes sans méthodes) et le fonctionnement métier (cas d'utilisation et diagrammes d'activités) mais n'apporte rien de vraiment neuf dans cette optique.

Merise et UML apparaissent donc **plus complémentaires que concurrents**.

TD ACSI : Cas d'utilisation UML

1. Gestion de la formation

Une entreprise souhaite modéliser avec UML le processus de formation de ses employés afin d'informatiser certaines tâches.
Le processus de formation est initialisé quand l'employé dépose une demande de formation. Cet employé peut éventuellement consulter le catalogue des formations offertes par les organismes sélectionnés par le responsable formation. Cette demande est examinée par le responsable. Pour prendre sa décision (accord ou refus), le responsable examine le catalogue des formations agréées qu'il tient à jour. Il informe l'employé du contenu de la formation choisie et lui soumet la liste des prochaines sessions prévues. Lorsque l'employé à fait son choix il inscrit l'employé à la session retenue auprès de l'organisme de formation concerné.
En cas d'empêchement l'employé doit le signaler au plus vite au responsable formation, pour que celui-ci demande l'annulation de l'inscription à l'organisme concerné.
A la fin de la formation l'employé transmet une appréciation sur le stage suivi.
Le responsable formation valide la formation au vu de la facture envoyée par l'organisme de formation.

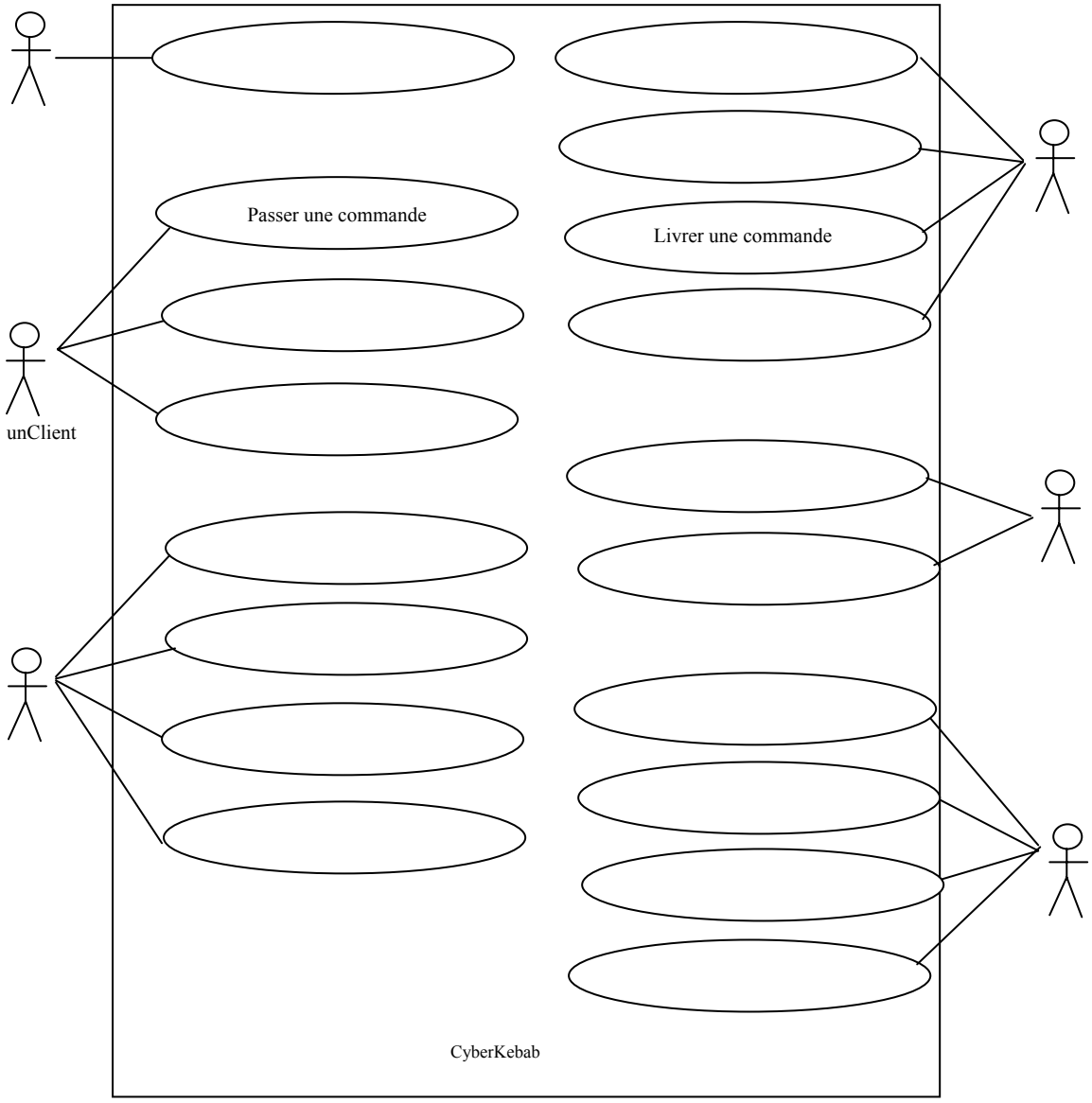
Travail à faire
Identifier les acteurs et les cas. Dessiner le diagramme des cas d'utilisation en structurant éventuellement les cas.

2. Cyber-Kebab

L'entreprise MegaKebab regroupe dans une même ville de nombreux restaurants appelés "Points Kebab". Elle est spécialisée dans la livraison à domicile de Kebabs et autres spécialités. Actuellement, les commandes se font par téléphone directement auprès de chaque restaurant. Un nombre limité de commandes peut être traité et chaque client doit connaître la carte des plats offerts par le Point Kebab contacté (ils varient d'un restaurant à l'autre). La direction de MegaKebab souhaite informatiser le processus de commande/fabrication/livraison via un logiciel baptisé CyberKebab.
Grâce à ce logiciel, MegaKebab souhaite gérer à distance et de manière centralisée toutes les commandes, les Points Kebab et les employés appelés "Collaborateurs". Cette centralisation doit permettre de rendre accessible sur Internet tous les plats disponibles. Chaque plat est décrit par un nom, une photo et un prix (identique partout). Dans le cadre de la politique marketing, une durée est également associée à chaque plat chaud : si le temps écoulé entre la fin de préparation et la livraison est supérieur à cette durée, le client peut se faire rembourser sa commande. Cependant, pour ne pas inciter les clients à utiliser cette possibilité, cette opération n'est pas disponible sur Internet : le client doit remplir une demande écrite sur papier libre et l'envoyer au gérant de MegaKebab. A tout moment il est possible de passer une commande par Internet. Le client doit disposer d'une carte de crédit qui l'identifie de manière unique. Lors d'une première commande il lui est également demandé de saisir son nom et de situer son lieu de résidence sur une carte de la ville. Une même commande peut comporter plusieurs plats. Pour chaque plat sélectionné le client doit indiquer la quantité désirée. Après avoir passé sa commande, le client peut à tout moment consulter l'état de sa commande. Tant que la commande n'est pas partie du PointKebab, il peut l'annuler.
Les Points Kebab sont ouverts 24h/24. Pour assurer un service 24h/24 dans toute la ville, MegaKebab fait appel à un grand nombre de collaborateurs, souvent étudiants, qui ont des horaires très flexibles. Lors de leur embauche, un téléphone portable leur est remis. Il suffit d'appuyer sur un bouton pour faire part de leur disponibilité auprès de MegaKebab. Un autre bouton permet d'indiquer qu'ils ne le sont plus. A tout moment le gérant peut consulter via Internet l'état du système global. Il peut affecter un collaborateur soit à un Point Kebab soit à la livraison. Un collaborateur peut ainsi changer de lieu de travail ou de rôle plusieurs fois dans une journée : le rôle du gérant est d'optimiser l'attribution de chacun en fonction des commandes. Lorsqu'un client passe une commande, il n'indique pas de PointKebab particulier; c'est le gérant qui affecte la commande à un PointKebab et à un livreur. Le gérant cherche en général à optimiser la distance parcourue ainsi que les activités des PointKebabs et des collaborateurs.
Chaque livreur utilise son propre moyen de transport (bus, vélo, roller, voiture …). Par contre, un appareil appelé "Pilote" lui est remis lors de son affectation à la livraison. Chaque pilote intègre un GPS permettant de localiser le livreur de manière précise via une liaison satellite. Un écran permet au

livreur de consulter les commandes qui lui ont été affectées. Il peut à tout moment consulter la carte et se situer par rapport aux points Kebab et aux clients à livrer. Le livreur utilise également le pilote pour indiquer quand il récupère une commande auprès du Point Kebab et quand il livre la commande au client.
Dans chaque Point Kebab un collaborateur joue le rôle de "coordinateur". C'est le seul du restaurant à agir directement avec CyberKebab : les autres collaborateurs préparent les plats. Le coordinateur consulte les commandes à réaliser et indique pour chaque commande quand sa préparation débute, quand elle se termine et quand elle est remise au livreur.

Travail à faire
Compléter le diagramme des cas d'utilisation du système CyberKebab donné ci-dessous. Seuls les acteurs humains sont pris en compte (ni le Pilote, ni le téléphone ne sont représentés).



3. Gestion d'une bibliothèque

La bibliothèque prête des livres et magazines à des emprunteurs, qui sont enregistrés dans le système de même que les livres et les magazines Les titres les plus demandés peuvent exister en plusieurs exemplaires. Les vieux exemplaires sont retirés quand ils sont dépassés ou abîmés.

Le «bibliothécaire» est l'employé qui interagit avec les emprunteurs et dont le travail est assisté par le système. Les documents ne sont pas en accès libre. Les clients peuvent consulter des listes de titres et demandent les titres désirés.
Un emprunteur peut réserver un titre non disponible. Quand le livre ou magazine est retourné ou acheté, la personne est avertie. La réservation est annulée quand l'emprunt est fait ou explicitement par une opération d'annulation. Le système doit permettre facilement de créer, mettre à jour, supprimer des titres, des emprunteurs, des emprunts, des réservations.

Travail à faire
Dessiner le diagramme des cas d'utilisation du système.

4. Application commerciale

1.1. Besoin
Des commerçants souhaitent participer à l'animation du centre ville par la création d'un groupement de commerçants dont les principaux objectifs sont :
- attirer la clientèle en centre ville par la création d'une carte de fidélité,
- animer des opérations de marketing (semaine de promotion, ...),
- constituer un carnet d'adresse des clients pour des opérations de mailing ciblés.

Sur chaque achat donnant lieu à un acte de fidélisation, le commerçant accorde une remise de 5 % sur le montant de l'achat. Sur cette remise, est prélevée une participation au frais destinée au prestataire informatique.
1.2. Architecture
Tous les commerçants sont dotés d'un TPE (Terminal de Paiement Electronique) sur lequel, à coté du logiciel de paiement bancaire, peut être installé un logiciel de fidélité.
Le TPE dialogue avec un serveur hébergé chez le prestataire qui mémorise les transactions et effectue les calculs de fidélité. Le serveur doit disposer d'une application permettant d'effectuer la comptabilité de fin de mois en liaison avec les banques et de calculer les statistiques.
Un serveur WEB doit permettre au groupement de saisir les informations nécessaires au fonctionnement de la fidélité.
1.3. Principes
Le principe de la fidélité est le suivant.
1) Chaque client est possesseur d'une carte de fidélité magnétique valide dans tous les commerçants du groupement.
2) Chaque achat donne lieu à une provision de remise (Montant d'achat * 5% - frais). Le commerçant est débité chaque mois des provisions accordées plus la cotisation de fonctionnement au profit d'un compte détenu par le groupement.
3) A bout du dixième achat, la somme des provisions (moins les frais) devient une remise accordée au client qui est déduite par le commerçant du montant de ce dixième achat. Si la remise accordée est supérieure au montant de l'achat, le commerçant débourse de sa caisse la différence. Le commerçant est crédité du montant de la remise accordée, en fin de mois, à partir du compte du groupement.

1.4 'Acte de fidélité'
Lorsqu'un nouveau client arrive, le commerçant donne au client une carte magnétique permettant d'effectuer les 'actes de fidélité'. Le commerçant demande au client de remplir une fiche sur laquelle le client donne son adresse. Le commerçant complète la fiche avec le numéro de la carte de fidélité. La fiche est envoyée à la secrétaire du groupement qui effectue la saisie des adresses.
L'enregistrement d'un acte d'achat donnant lieu à une remise au titre de la fidélité est le suivant :
1) Le commerçant saisit le montant de l'achat.
2) Le commerçant introduit la carte de fidélité.
3) Le TPE appelle un serveur chez le prestataire informatique par le réseau téléphonique commuté.
4) Le serveur reçoit la demande de fidélité avec le numéro du commerçant, le numéro de la carte et le montant. Il vérifie l'existence de la carte, du commerçant et calcule le montant de la fidélité. En cas d'erreur un message de rejet s'affiche sur le TPE.
5) 1er cas :
   - si le client n'a pas droit à la remise, le serveur renvoie la provision de remise accordée,

- le TPE édite un ticket en double exemplaire (l'un pour le commerçant, l'autre pour le client) avec le nom du commerçant, la date d'achat, le montant de l'achat, la provision de remise, le cumul des provisions et le rang de l'achat (de 1 à 9),
- le commerçant encaisse la totalité du montant de l'achat.

6) 2ème cas :
   - Le client a droit à une remise, le serveur renvoie les informations permettant le remboursement de la remise (somme des provisions - somme des frais)
   - Le TPE édite un ticket comprenant, le nom du commerçant, la date d'achat, la provision de remise sur cet achat, le montant des provisions de remise, et le restant dû par le client (achats moins remise). Il est possible que la somme des remises soit supérieure au montant de l'achat.
   - Le commerçant encaisse le montant de l'achat moins la remise ou décaisse l'écart.

1.5. Site Web

Le site Web doit permettre au groupement de paramétrer l'application sur le serveur, de saisir les adresses et d'effectuer des interrogations sur la vie des cartes. Peu de commerçants disposent d'internet. La secrétaire du groupement est la principale utilisatrice du site Web.

1.6. Comptabilité

La comptabilité est décrite plus haut. Elle est effectuée en fin de mois pour les transactions du premier au dernier jour du mois. Le groupement souhaite que les prélèvements et les virements soient effectués automatiquement vers la banque qui détient le compte du groupement.

Travail à faire

Dessiner le diagramme des principaux cas d'utilisation (ils peuvent regrouper logiquement plusieurs fonctionnalités que l'on détaillera).

TD ACSI : Diagramme de classes UML

1. Gestion cirque

Le propriétaire d'un cirque souhaite informatiser une partie de la gestion de ses spectacles. Proposer un diagramme de classes UML qui réponde aux spécifications, fournies ci-dessous.
Les membres du personnel du cirque sont caractérisés par un numéro (en général leur numéro INSEE), leur nom, leur prénom, leur date de naissance et leur salaire. On souhaite de surcroît stocker les pseudonymes des artistes et le numéro du permis de conduire des chauffeurs de poids lourds.
Les artistes sont susceptibles d'assurer plusieurs numéros, chaque numéro étant caractérisé par un code, son nom, le nombre d'artistes présents sur scène et sa durée. De plus, on souhaite savoir l'instrument utilisé pour les numéros musicaux, l'animal concerné par les numéros de dressage et le type des acrobaties (contorsionnisme, équilibrisme, trapèze volant…).
Par ailleurs, chaque numéro peut nécessiter un certain nombre d'accessoires caractérisés par un numéro de série, une désignation, une couleur et un volume.
On souhaite également savoir, individuellement, quels artistes utilisent quels accessoires.
Enfin, les accessoires sont rangés après chaque spectacle dans des camions caractérisés par leur numéro d'immatriculation, leur marque, leur modèle et leur capacité (en volume). Selon la taille du camion, une équipe plus ou moins nombreuses de chauffeurs lui est assigné (de un à trois chauffeurs).

Travail à faire

Dessiner le diagramme de classes.

2. Gestion de la formation

On reprend l'énoncé de la gestion de la formation pour lequel on a déjà construit les cas d'utilisation.

Travail à faire

Dessiner le diagramme de classes de cette application, incluant toutes les classes que l'on peut déduire de l'énoncé, ainsi que les associations entre classes avec leurs cardinalités.

3. Cyber-kebab

On reprend l'énoncé du cyber-kebab pour lequel on a déjà construit les cas d'utilisation.

Travail à faire

Compléter le diagramme de classes en annexe sans ajouter ni classes, ni associations mais en complétant les zones en pointillés. Les zones de petite taille correspondent à des cardinalités.

4. Carte géographique

Une carte géographique est caractérisée par une échelle, la longitude et la latitude de son coin inférieur gauche, la hauteur et la largeur de la zone couverte par la carte. Une carte comporte un ensemble de données géographiques de natures diverses. Les villes et les montagnes sont repérées par un point unique. Chaque point a 2 coordonnées x et y calculées par rapport au coin inférieur gauche de la carte. Un nom est associé à chaque donnée géographique repérée par un point. Les routes et les rivières sont repérées par des lignes brisées, c'est à dire par un ensemble de points correspondant aux extrémités de ses segments de droite. Les routes et les rivières ont des noms et des épaisseurs de trait. Les lacs, mers et forêts sont représentées par des régions caractérisées par un nom et une couleur de remplissage. Une région est une ligne brisée refermée sur elle même.

Travail à faire

Donnez un schéma de classes UML permettant de représenter une carte en utilisant les relations de spécialisation (héritage) et de décomposition (aggrégation).

5. Les démons

a. Pour chaque paragraphe numéroté, dessinez un diagramme UML permettant de représenter les notions que ce paragraphe décrit.

(1) Les démons sont de deux sortes : les fermions et les bosons.

(2) Un être vivant possède une ou plusieurs loges dans lesquelles viennent se placer des démons. Un démon est ubiquiste, cela signifie qu'il peut être présent dans plusieurs loges.

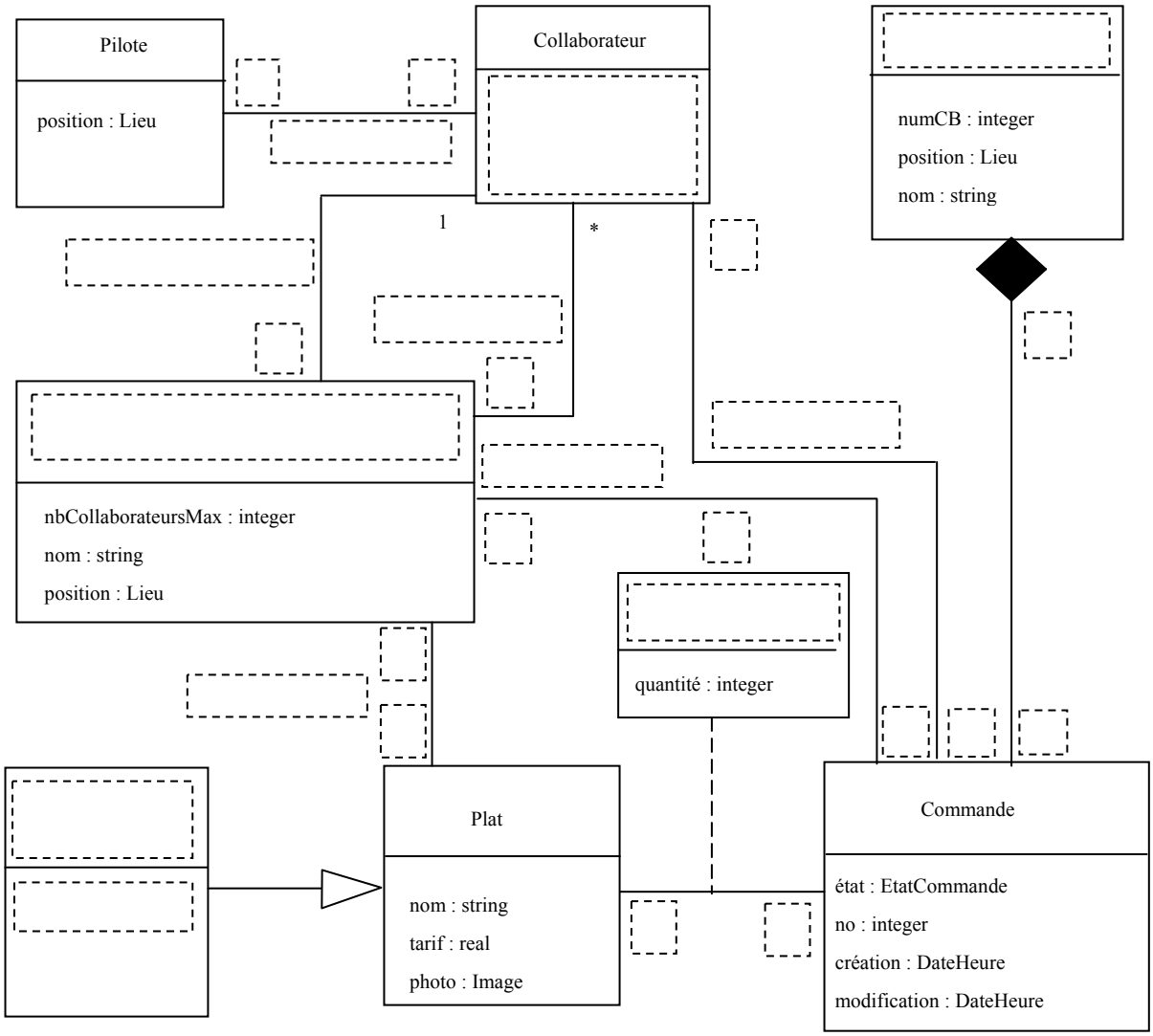
(3) Les bosons sont toujours à plusieurs dans une loge. Dans ce cas la loge est dite bosonique. Un fermion, par contre, est toujours seul dans une loge. Dans ce cas la loge est dite fermionique.

(4) Les êtres humains normaux possèdent deux loges bosoniques (remplies de bosons). 5% sont hors norme : ils possèdent une loge avec des bosons et une loge fermionique (avec un fermion). 0,000001% sont rarissimes : ils possèdent deux loges avec un fermion.
(5) Il existe plusieurs sortes de bosons : les hypnotiques, les processionnaires et les caracoles.
b. Synthétisez les diagrammes précédents en un seul.
c. Un démon possède une puissance, représentée par un nombre. Pour un boson, ce nombre est entier, il s'appelle le charme. Pour un fermion, ce nombre est réel, il s'appelle la résistance. Les hypnotiques ont un charme variable, les caracoles ont un charme constant de 1, les processionnaires ont un charme constant de 2.
Placez dans les classes les attributs 'puissance', 'charme', 'résistance'.
Idem avec les méthodes 'void occuperUneLoge(Loge)', 'void ecrireCharme(entier)', 'entier lireCharme()', réel lireResistance()', 'void afficherBosons()', 'void afficherFermion()'.

6. Les Vols

Une compagnie aérienne gère des vols, c'est-à-dire des parcours aériens entre une ville de départ et une ville d'arrivée, avec un numéro de vol et une fréquence. Un vol peut se décomposer en un ou plusieurs tronçons (s'il existe des escales dans des villes intermédiaires), caractérisés chacun par une ville et une heure de départ, une ville et une heure d'arrivée, une distance. Certains vols se partagent les mêmes tronçons mais pas nécessairement aux mêmes heures.
Lorsqu'un vol est programmé pour une date il constitue un départ, caractérisé par un numéro de départ. Un vol n'est programmé qu'une seule fois dans une journée à l'heure de départ.
Des passagers sont enregistrés pour un départ, caractérisés par un nom, une adresse et un numéro de téléphone.
Un avion est affecté à chaque départ, caractérisé par son immatriculation, son type et sa capacité. Il utilise une certaine quantité de kérosène pour le trajet qui dépend des conditions climatiques et donc de la date du départ.
Des personnels sont affectés à chaque départ. On distingue les non-navigants et les navigants. Ils sont caractérisés par leur nom, adresse et numéro de téléphone. Pour les navigants on garde le cumul des heures volées dans l'année.
Travail à Faire
Donnez un schéma de classes UML utilisant au maximum la relation de spécialisation/ généralisation entre classes (héritage).
Rappel : des attributs peuvent être attachés à une association grâce à une classe anonyme qui lui est liée.

Annexe

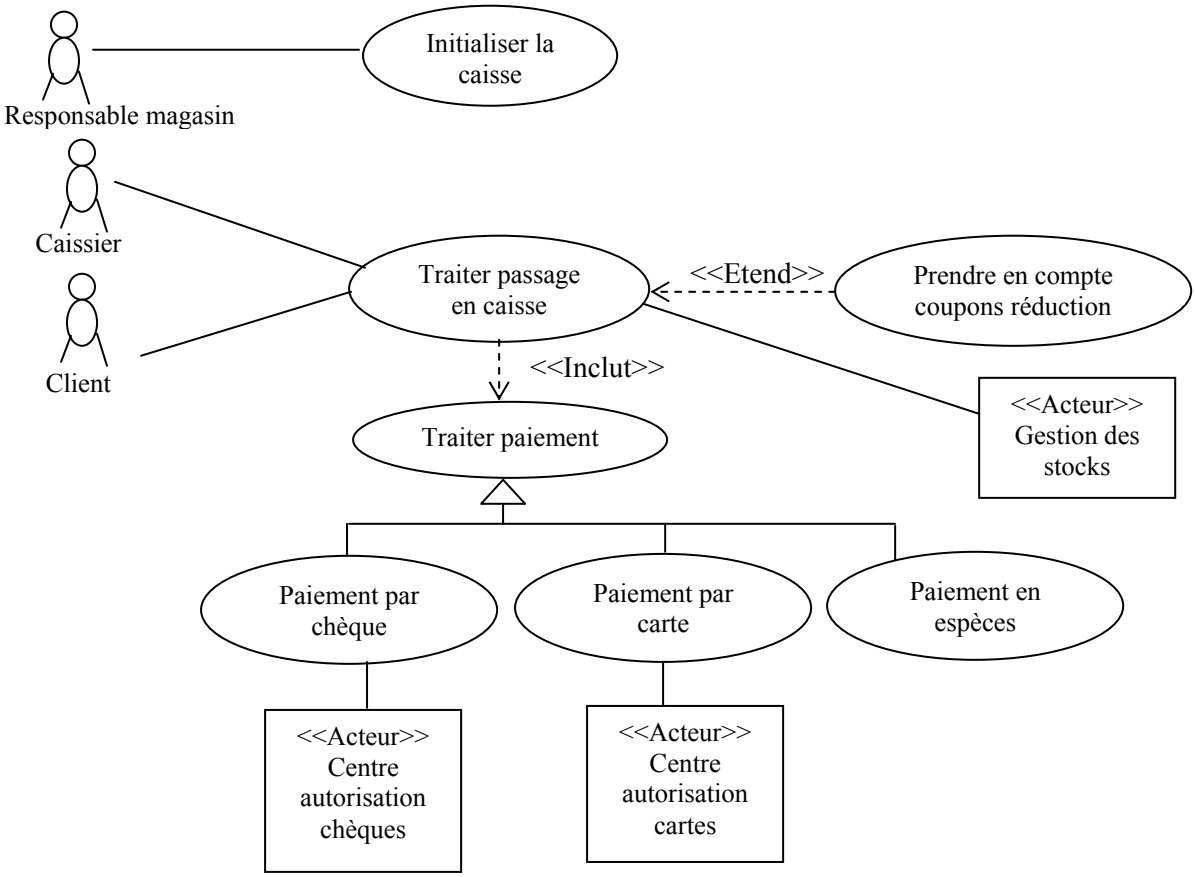


Lieu est un type permettant de situer dans l'espace.
DateHeure est un type permettant de situer dans le temps.
EtatCommande est un type énuméré prenant les valeurs suivantes :

TD ACSI : Diagramme de séquences UML

1. Passage en caisse - diagramme de séquence au niveau de l'analyse des besoins

On considère le cas d'utilisation 'Traiter le passage en caisse' au sein de la gestion des caisses enregistreuses d'un supermarché.



Le scénario nominal d'un passage en caisse avec paiement en espèces est le suivant :
- un client arrive à la caisse avec des articles à payer,
- le caissier enregistre le numéro d'identification de chaque article et la quantité si elle excède un,
- la caisse affiche le libellé et le prix de chaque article,
- lorsque tous les achats sont enregistrés le caissier signale la fin de l'enregistrement,
- la caisse affiche le total des achats,
- le client choisit de payer en espèces et donne une somme et éventuellement des coupons de réduction,
- la caissier enregistre la somme reçue et éventuellement les coupons de réduction,
- la caisse affiche la somme à rendre,
- le caissier encaisse la somme et sort la monnaie à rendre,
- le caissier rend la monnaie,
- la caisse enregistre la vente et imprime le ticket,
- le caissier donne le ticket de caisse au client.

Travail à faire

Représenter ce scénario comme un diagramme de séquence entre caisse, caissier et client. On pourra faire apparaître les messages et les flux matériels (en pointillés).

2. Application commerciale - diagramme de séquence au niveau de l'analyse des besoins

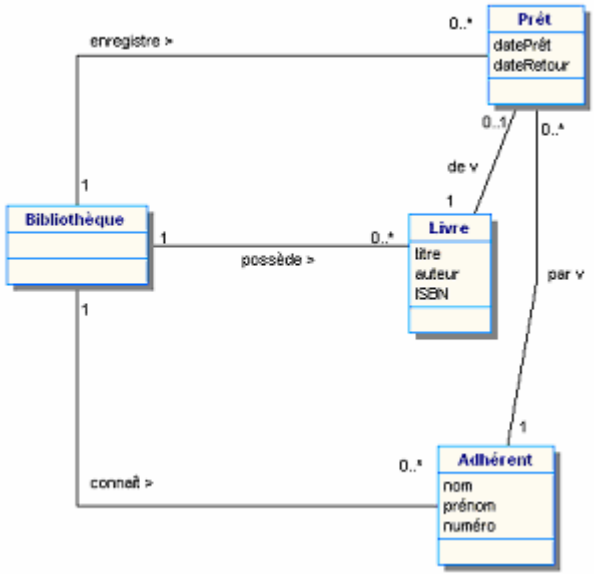
On reprend l'énoncé de l'application commerciale pour lequel on a déjà construit les cas d'utilisation.

Travail à faire

Dessiner le diagramme de séquence de l'acte de fidélité.

3. Gestion d'une bibliothèque - diagramme de séquence entre classes d'une application au niveau de l'analyse du système ('classes métiers')

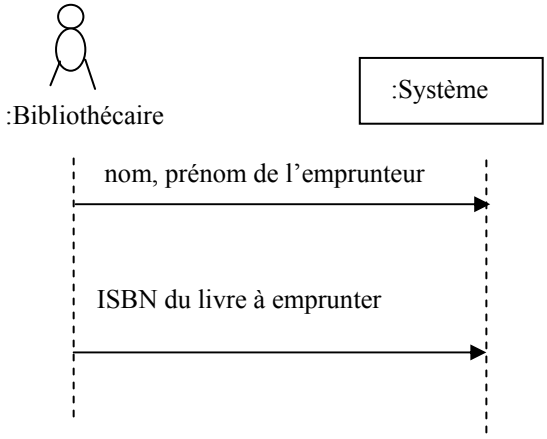
Au cours de l'analyse de la gestion d'une bibliothèque on a retenu les classes métier suivantes.



Rappel : une association s'implante par un attribut contenant un objet (si cardinalité 1) ou par une collection (table) d'objets (si cardinalité *). Donc l'implantation de Bibliothèque aura 3 attributs collection (tables) pour les 3 associations et celle de Prêt aura 2 attributs pour les associations 'de' et 'par'.

Travail à faire

Raffiner le diagramme de séquence suivant (associé au cas Emprunt des livres) en faisant intervenir les classes concernées et les messages qu'elles s'échangent.



Les tables de la classe Bibliothèque (table de tous les objets livre, table de tous les objets adhérents et table de tous les prêts pour une durée 15 jours) ont des méthodes :

- trouverLivre(ISBN), trouverAdhérent(nom, prénom) et trouverPrêt(n° prêt) qui retournent les objets cherchés,
- ajouterLivre(objet livre), ajouterAdhérent(objet adhérent) et ajouterPrêt(objet prêt) qui ajoutent les objets aux tables.

Rappel : pour créer un objet on appelle la méthode créer(valeurs initiales des attributs) qui retourne cet objet; pour modifier un attribut d'un objet on appelle la méthode setAttribut(valeur); pour lire la valeur d'un attribut d'un objet on appelle getAttribut() qui retourne la valeur.

TD ACSI : diagrammes de modélisation de la dynamique

1. Guichet automatique de banque - diagramme d'activités et d'états

Modéliser le retrait d'argent dans un guichet automatique de banque (GAB). La carte peut être invalide (ex : date d'expiration dépassée) et elle est refusée. Si elle est valide, le client doit taper son code. La carte est avalée après trois essais infructueux. Le système d'autorisation VISA autorise un certain montant ou refuse tout retrait. Une carte non récupérée après quelques secondes est avalée. Les billets non récupérés par le client sont repris. Un ticket est toujours imprimé pendant que les billets sont proposés.

Travail à faire

a) Modéliser avec un diagramme d'activités la dynamique de ce système.
b) Modéliser avec un diagramme d'états l'évolution de la carte de crédit dans le GAB.

2. Gestion de la formation - diagramme d'activités et d'états

On reprend l'énoncé de la gestion de la formation pour lequel on a déjà construit les cas d'utilisation.

Travail à faire

a) Modéliser avec un diagramme d'activités la dynamique de ce système.
b) Modéliser avec un diagramme d'états l'évolution d'une demande de formation.

3. Application commerciale - diagramme d'activités

On reprend l'énoncé de l'application commerciale pour lequel on a déjà construit les cas d'utilisation.
On dispose des informations complémentaires suivantes.

– En cas d'erreur sur la transaction (généralement erreur sur la saisie du montant) : abandon de la transaction.
– En cas de carte illisible : abandon de la transaction.
– En cas de non-établissement de la ligne après 3 tentatives : abandon de la transaction.
– La validation des comptes se fait après encaissement par le commerçant grâce à un accusé de paiement envoyé au serveur.

Travail à faire

Modéliser avec un diagramme d'activités la dynamique de l'acte d'achat.

4. La vie d'un thread. Diagramme d'états

Dessinez un diagramme d'états correspondant à la dynamique d'un « thread » (processus léger) définie de la manière suivante. Le thread est :

- « non démarré » au début,
- « en cours » lorsqu'il possède toutes ses ressources applicatives plus le processeur,
- « en attente » lorsqu'il lui manque une ressource applicative,
- « prêt » lorsqu'il a toutes ses ressources applicatives et pas le processeur,
- « terminé » lorsqu'il a terminé son exécution.

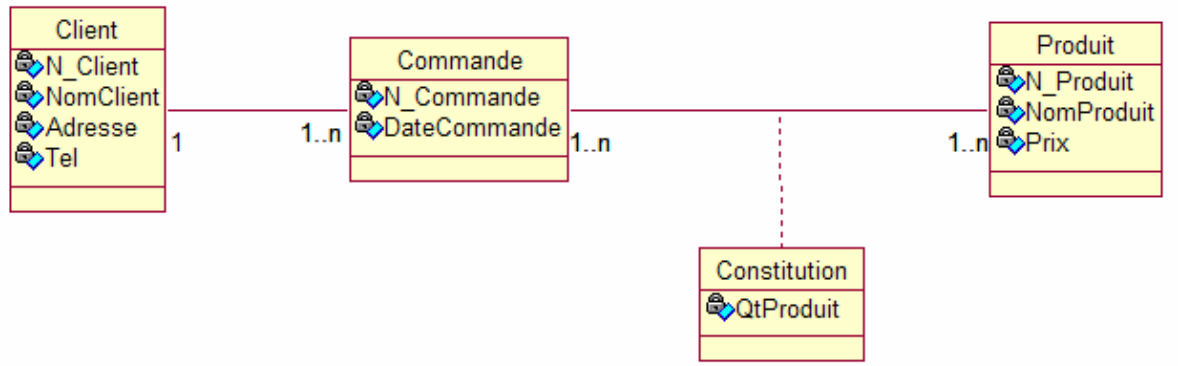
On supposera qu'un thread n'envoie pas d'événement. Il ne fait que les recevoir. On supposera que les événements reçus par le thread sont : « début », « ressource attendue », « ressource OK », « processeur OK », « fin » :

- « début » correspond au démarrage du thread (start en java, execlv en Unix, …). Avant la réception de « début », le thread est « non démarré ».
- « ressource attendue » correspond à l'indication qu'une ressource applicative attendue n'est pas disponible.
- « ressource OK » correspond à la libération d'une ressource applicative par un autre thread et donc à la réservation effective de la ressource par le thread qui l'attendait.
- « processeur OK » correspond à la libération du processeur par un autre thread et à l'utilisation effective du processeur par le thread qui l'attendait.
- « fin » correspond soit à l'exécution de la dernière instruction du programme exécuté par le thread soit à l'envoi d'un événement pour tuer définitivement le thread. Sur réception de « fin », le thread devient « terminé ».

TD ACSI : Classes vers relationnel

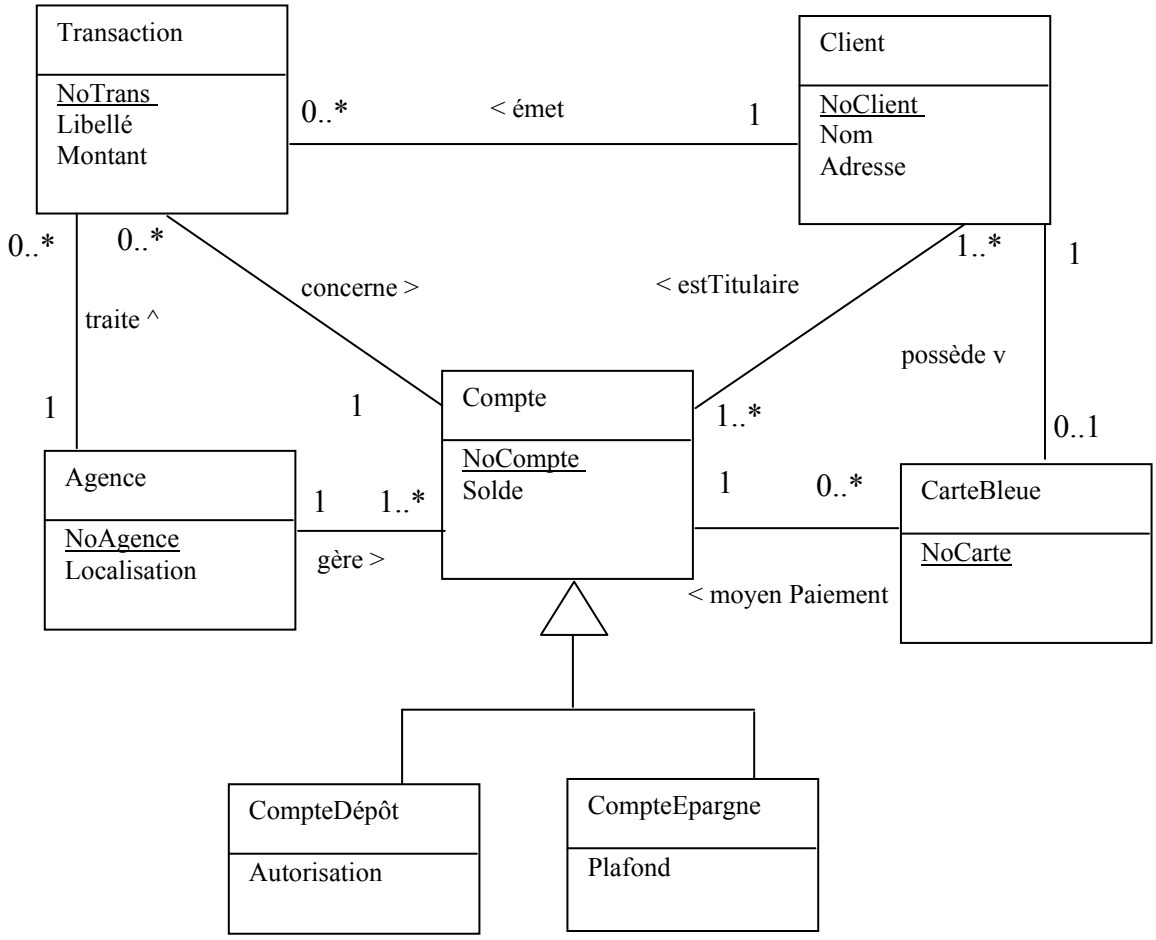
Exercice 1

Traduire le diagramme de classes UML suivant en relationnel.



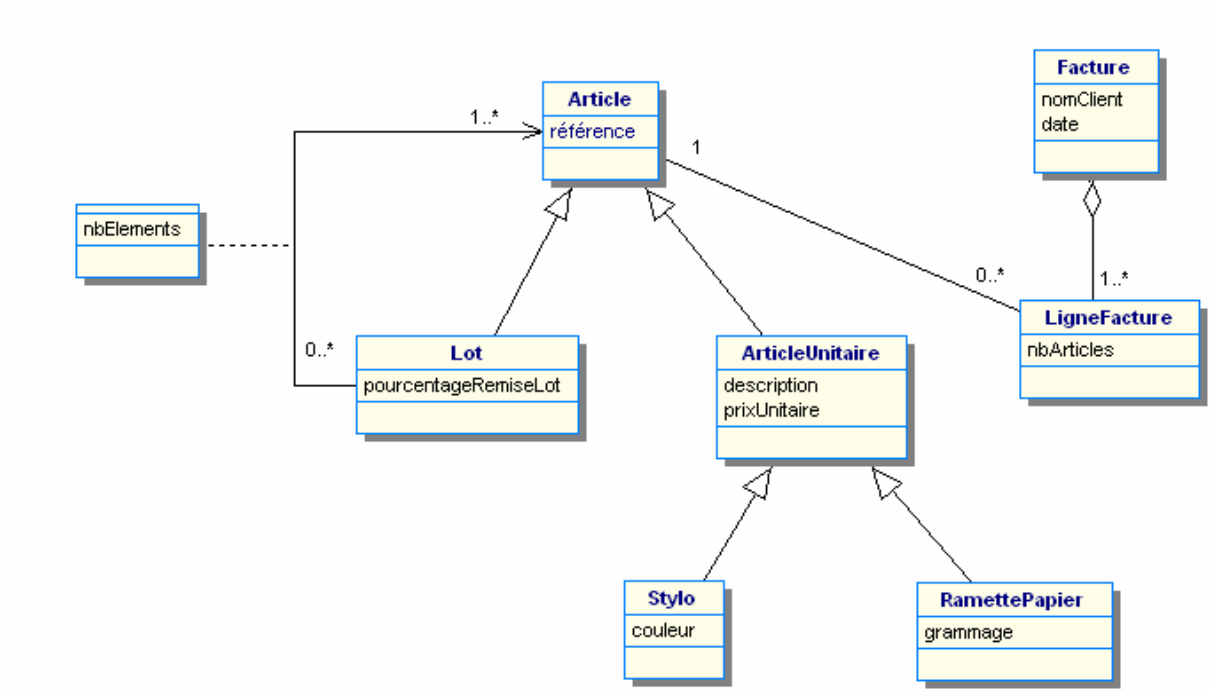
Exercice 2

Traduire le diagramme de classes UML suivant en modèle logique relationnel.

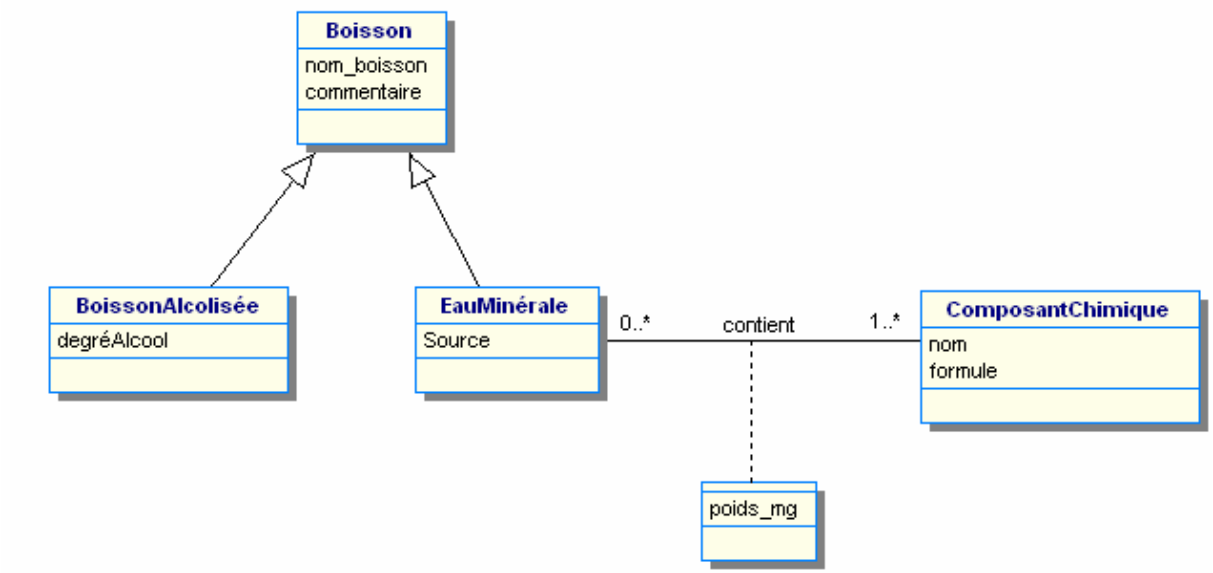


Exercice 3

Soit le schéma de classes ci-dessous.
a) D'après ce schéma, un lot peut-il contenir un lot ?
b) Traduisez ce schéma en relationnel avec la stratégie qui consiste à associer une table par classe de l'arbre d'héritage,
c) Même question avec la stratégie qui consiste à associer une seule table à tout l'arbre d'héritage.



Exercice 4
Soit le schéma de classes ci-dessous.
a) Traduisez ce schéma en relationnel avec la stratégie qui consiste à associer une table par classe de l'arbre d'héritage,
b) Même question, avec la stratégie qui consiste à associer une seule table à tout l'arbre d'héritage.

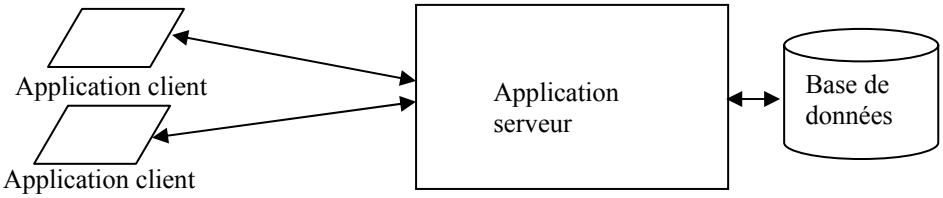


Etude de cas UML

Analyse d'un serveur de réunions virtuelles sur Internet

1. Présentation

Il s'agit d'adapter le concept de messagerie instantanée à un contexte de réunions de travail au sein d'une entreprise géographiquement dispersée. L'authentification des utilisateurs (login/mot de passe) est obligatoire pour utiliser l'application. Il vous faut analyser la partie serveur de cette application client-serveur permettant de faire des réunions virtuelles sur Internet.



L'objectif de cette application est de permettre d'imiter le plus possible le déroulement de réunions de travail classiques. Cependant, dans la première version de ce projet, les interventions des utilisateurs se feront en mode textuel seulement.

Le serveur devra permettre de planifier et de gérer le déroulement de plusieurs réunions simultanées. Une fois connectée (à l'aide d'un nom de login et d'un mot de passe mémorisé par le serveur), un utilisateur a la possibilité de :

- planifier des réunions virtuelles (choix d'un nom, définition du sujet, date de début et durée prévue, ordre du jour) dont il devient l'organisateur,
- consulter les détails d'organisation d'une réunion (tous les utilisateurs),
- modifier ces détails d'organisation (seulement l'organisateur),
- ouvrir et clôturer une réunion (seulement l'animateur – cf. ci-dessous),
- entrer (virtuellement) dans une réunion précédemment ouverte (seulement les participants autorisés),
- en sortir.

En cours de réunion, un participant peut demander à prendre la parole. Quand elle lui est accordée, il peut entrer le texte d'un message qui sera transmis en 'temps-réel' par le serveur à tous les participants de la réunion.

Une personne peut participer simultanément à plusieurs réunions.

Les messages sont stockés avec un n° d'ordre de réception, la date et heure de réception et le nom de l'auteur du message. Cela permet à un retardataire de recevoir l'ensemble des messages déjà échangés dans la réunion.

Plusieurs sortes de réunions doivent pouvoir être organisées :

- réunions 'standards', avec un organisateur qui se charge de la planification de la réunion et désigne un animateur chargé de choisir les intervenants successifs parmi ceux qui demandent la parole ; tout utilisateur peut participer (réunions publiques) ;

- réunions ‘privées’, qui sont des réunions standards dont l'entrée est réservée à un groupe de participants autorisé par l'organisateur ;
- réunions ‘démocratiques’, qui sont planifiées comme des réunions standards et, comme elles, sont publiques. Les intervenants successifs sont choisis automatiquement par le serveur sur la base d'une politique premier demandeur-premier servi (FIFO). La réunion est ouverte et fermée par l’organisateur qui joue le rôle d’animateur.

L’administrateur du système peut ajouter/supprimer des utilisateurs avec leur login et leur mot de passe.

2. Travail à faire

a) Etablir le diagramme des cas d’utilisation du système.
b) A partir de l’énoncé, proposer un diagramme de classes initial avec les utilisateurs, les réunions, les principales associations et relations d’héritage entre ces concepts et les attributs essentiels. Les méthodes seront ajoutées ultérieurement.
c) Expliciter quelques cas par des diagrammes de séquence : connexion au serveur, planification d’une réunion virtuelle, ouverture d’une réunion virtuelle. Ces diagrammes doivent montrer toutes les classes qui participent (c'est-à-dire qui hébergent des traitements, qui sont créées, qui sont interrogées, …) avec les messages qui circulent vers et depuis ces classes. Le point d’entrée est un acteur qui envoie un message depuis l’application client.
d) Donner les diagrammes d’états des deux classes principales du diagramme de classes initial.
e) A partir des résultats précédents et de l’énoncé, enrichir le diagramme de classes avec les classes, les associations, les attributs et méthodes jusqu’à ce qu’il apparaisse ‘raisonnablement complet’ pour cette phase initiale d’analyse.
   Essayer d’utiliser un maximum de possibilités de la notation objet UML (héritage, agrégation, …).

Un dossier d’analyse de l’existant devra être rendu qui réponde à ces questions.
Les schémas seront réalisés à l’aide de Win’Design.
Les schémas devront être accompagnés d’explications à chaque fois que des choix non évidents auront été effectués.