

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

COURS et TD

Public concerné : DUT Informatique 2^{ème} année

Jacques LONCHAMP

Date : 2010/2011

**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://iut-charlemagne.univ-nancy2.fr>**

Table des matières

PARTIE 1 : COURS

1. Présentation d'UML	p. 5
2. Les cas d'utilisation	p. 9
3. Les diagrammes de classes	p. 15
4. Les diagrammes d'interactions	p. 25
5. Les diagrammes d'états et d'activités	p. 29
6. Traduction schéma de classes vers schéma relationnel	p. 35
7. Conclusion	p. 39

PARTIE 2 : TRAVAUX DIRIGES

1. TD cas d'utilisation	p. 49
2. TD diagrammes de classes	p. 53
3. TD diagrammes de séquences	p. 57
4. TD diagrammes de modélisation de la dynamique	p. 55
5. TD classes vers relationnel	p. 63

PARTIE 3 : ETUDE DE CAS UML	p. 65
------------------------------------	--------------

UML - Présentation



- Merise correspond à une première évolution dans les années 80 autour des idées :
 - **de système d'information (SI)**,
 - **de niveaux de modélisation** (conceptuel, organisationnel, physique)
 - **de séparation données/traitements**,
 - **de base de données**.
- 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.

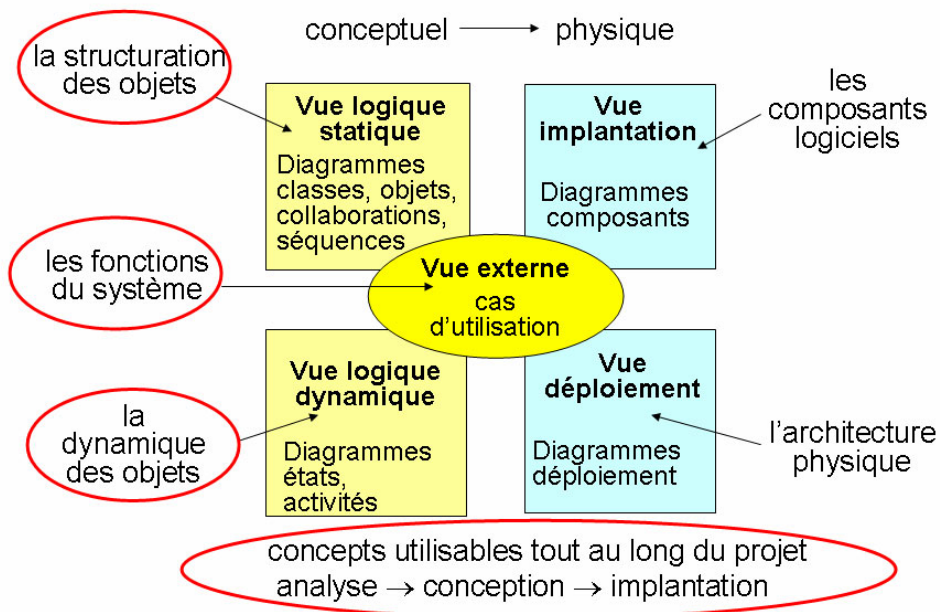
UML : un même langage tout au long de la démarche d'informatisation

- La représentation du monde réel (modèle des besoins) se fait avec les mêmes concepts que celle du logiciel (modèle d'implantation) : **objets, classes, opérations, attributs et associations**.
- La démarche ne consiste plus à réécrire un modèle d'un certain niveau avec les concepts du niveau suivant au moyen de règles de traduction comme en Merise.
- On passe d'un niveau à un autre par **enrichissement** des éléments existants et adjonction d'éléments nouveaux en conservant le même langage de haut niveau.

UML

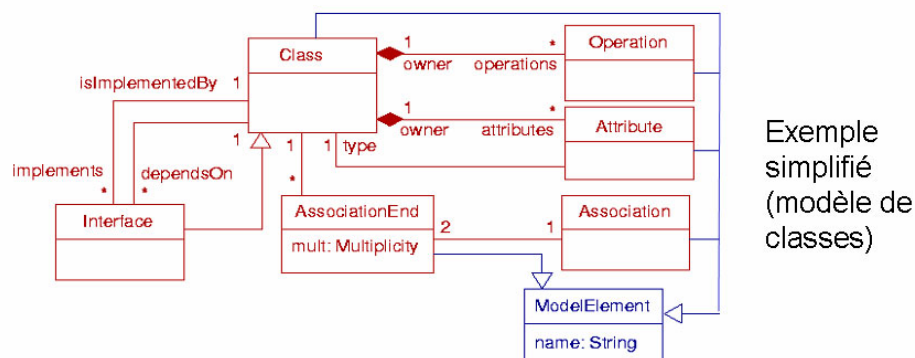
- Un langage pas une méthode : UML définit des modes de représentation (diagrammes et notations) mais n'impose pas de démarche standardisée.
- Un langage de modélisation objet permettant de documenter dans des modèles toutes les phases du développement (analyse, conception et implantation).
- Convient pour toutes les démarches et langages de programmation objet ('Unified Modeling Language').
- Dans le domaine public. C'est l'OMG ('Object Management Group') chargé de la normalisation des technologies objets qui pilote UML.
- Version actuelle : UML 2.1.2.

UML est une proposition complexe (13 types de diagrammes) et évolutive. Nous n'en étudions que les **bases** :



Le Méta-modèle UML

- Les concepts UML ont été formalisés en UML (définition récursive).
- Ce « méta-modèle »
 - décrit formellement les concepts avec leur syntaxe et sémantique,
 - fait la preuve de la puissance d'expression de la notation capable (entre autres) de se représenter elle-même,
 - sert de description de référence pour la construction d'outils.



Les cas d'utilisation

Définition des cas d'utilisation ('use cases')

- Expriment le fonctionnement d'un système selon le point de vue des utilisateurs. Permettent d'impliquer les utilisateurs dès les premiers stades du développement pour exprimer leurs attentes.
- Permettent de définir les **besoins des utilisateurs et les fonctionnalités du système** (le « quoi? » avant le « comment? ») :
 - **délimitation du système**,
 - **relations avec son environnement**,
 - **fonctions** attendues.
- Modélisent à la fois des **activités** (fonctionnalités) et des **communications** (interactions).

**Sont utilisables pour tout projet
indépendamment d'UML et de l'approche objet.**

Acteurs et cas

- **Acteur** : personne ou système qui interagit avec le système étudié en échangeant de l'information.

Ex: utilisateurs directs du système, responsables de son fonctionnement, autres systèmes qui interagissent avec lui ...

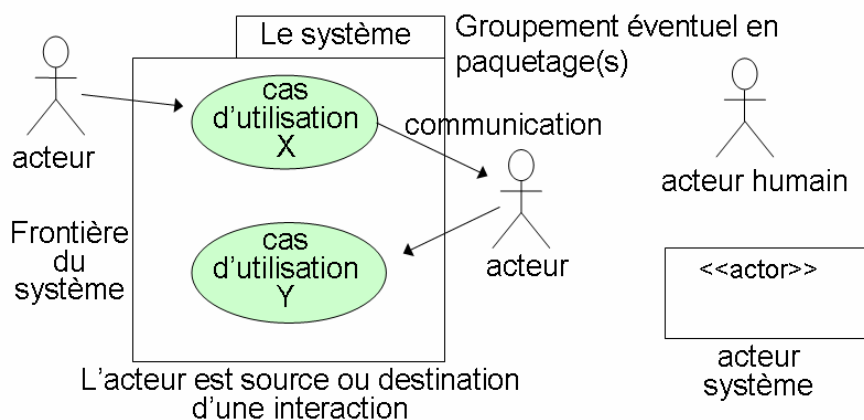
Un acteur représente un rôle. La même personne physique peut jouer le rôle de plusieurs acteurs et plusieurs personnes physiques peuvent jouer le même rôle et donc agir comme un même acteur.

- **Cas** : une interaction avec le système par un acteur dans une certaine intention; un service rendu par le système; une fonctionnalité.

Diagrammes de cas d'utilisation

Décrivent les interactions entre les acteurs et le système représenté comme un ensemble de cas.

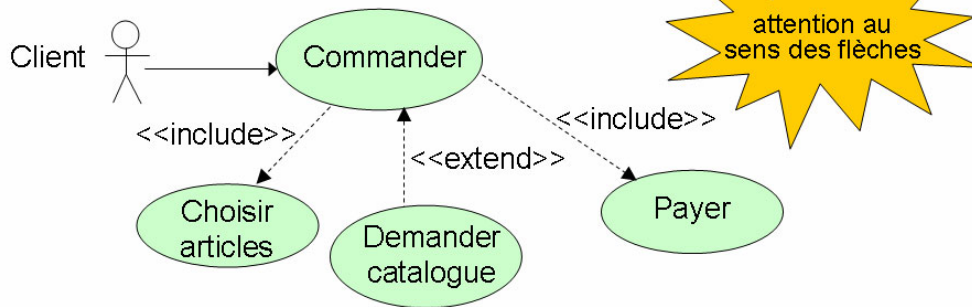
Les interactions sont orientées (avec une flèche) ou non. Les communications externes (entre acteurs) ne sont pas représentées).



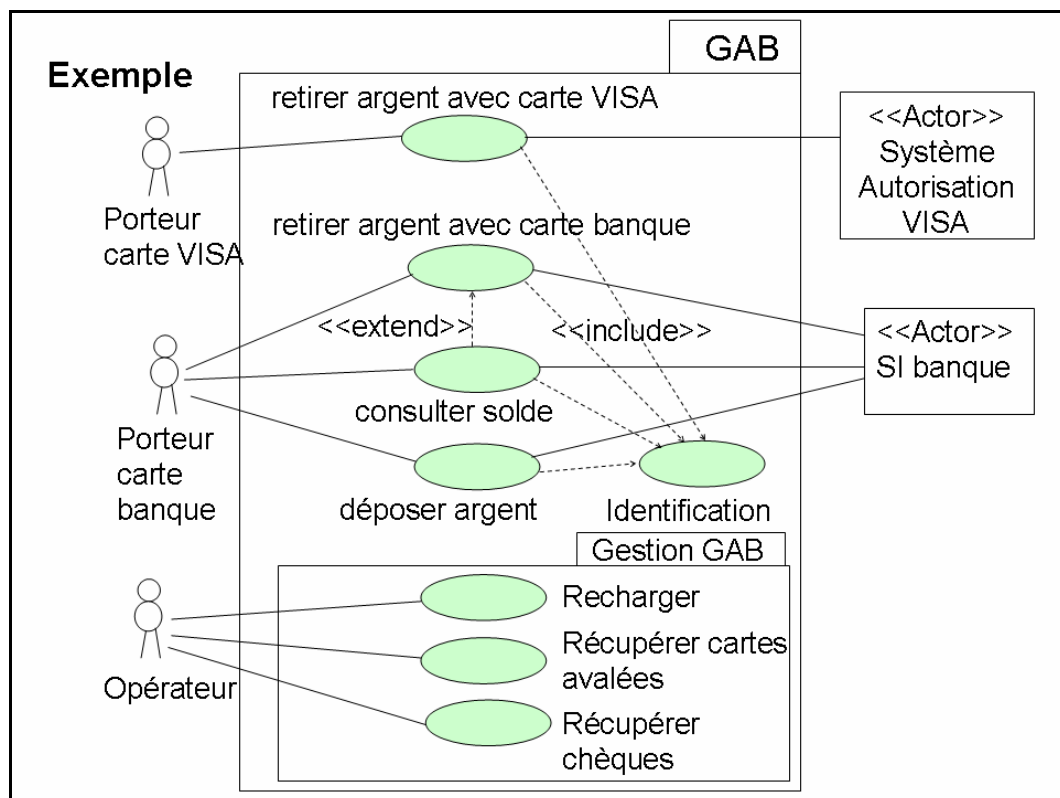
Relations entre cas

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

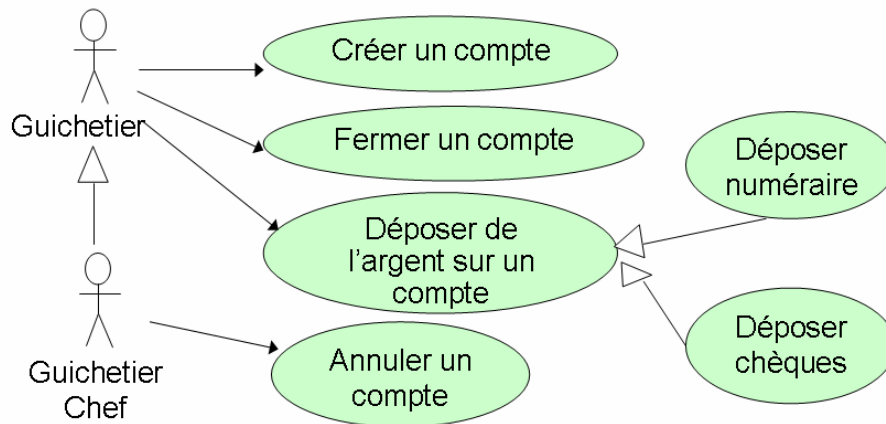
A **<<extend>>** B (le cas A est une extension optionnelle du cas B à un certain point de son exécution).



<<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.



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



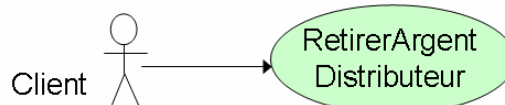
Un 'Guichetier Chef' est un 'Guichetier' spécialisé qui peut faire tout ce que peut faire un Guichetier et, en plus, il peut annuler un compte. 'Déposer chèques' et 'Déposer numéraire' sont 2 spécialisations de 'Déposer de l'argent sur un compte' (2 manières de faire).

Spécification des cas

Chaque cas d'utilisation doit être précisé par une description **textuelle structurée** :

- 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,
- les informations échangées entre acteur et système,
- les contraintes non fonctionnelles (performance, sécurité...).

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) Sécurité ...

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 (préparation des tests).

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.

...

Utilisation

- Elaborer avec les utilisateurs une description de très haut niveau du système.

Ex:

- Le client parcourt le catalogue.
 - Il ajoute les objets qui l'intéressent dans son caddy.
 - Pour payer il donne ses informations de cartes bancaires et son adresse
 - Il confirme l'achat.
 - Le système contrôle la validité du paiement et confirme la commande.
- En déduire une liste d'acteurs et de cas.
 - Construire un premier diagramme. Le structurer pour le rendre plus clair (relations <<include>>, <<extend>>, généralisation, spécialisation).
 - Détailler chaque cas (description textuelle structurée).
 - Eventuellement définir les scénarios les plus importants qui pourront servir au moment des tests.

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 les descriptions textuelles et les scénarios, l'analyste dispose de moyens simples pour exprimer de manière semi-formelle les **besoins fonctionnels et non fonctionnels du système étudié** (son cahier des charges).

Les diagrammes de classes

Objectif

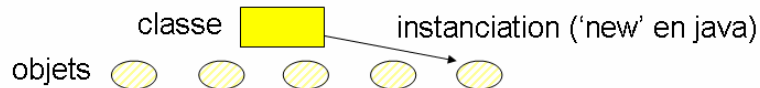
- Décrire la structure **statique** du système.
- Sous forme de **classes** et de **relations** entre classes.
- Lors de l'analyse
 - **classes du domaine** (correspondant aux 'objets métiers').
- Lors de la conception :
 - ajout des **classes « techniques »** liées aux choix de conception (interfaces utilisateurs, persistance, patrons de conception...).
- Lors de l'implantation :
 - ajout des **classes liées à l'implantation** dans un langage de programmation donné (structures de données ...).

Le concept de classe

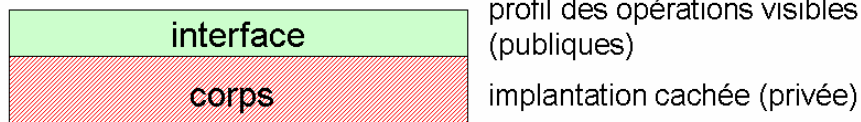
- Décrit un ensemble d'objets (instances de la classe). Décrit leurs **éléments communs** (les différences sont ignorées).

- **Classe = type + module**

Type : 'fabrique' d'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)



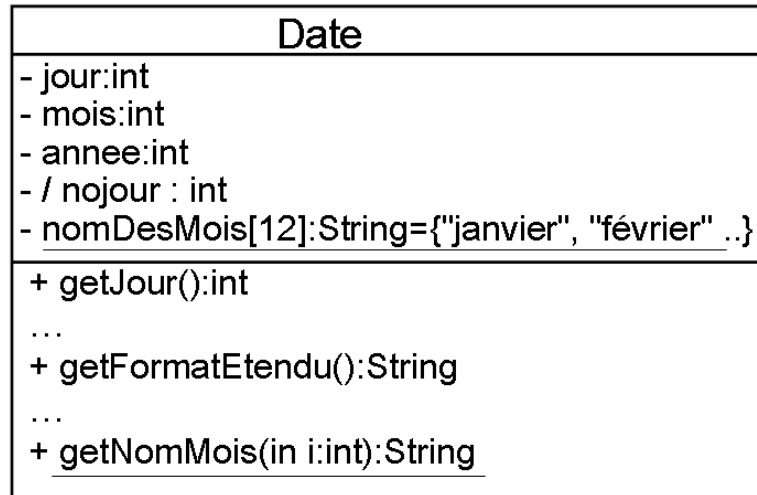
- Notation de base (suffisante au niveau analyse)

Nom de classe	Compte
Attributs	libellé
Opérations()	solde
	créditer()
	débiter()

- Nombreuses notations supplémentaires (aux niveaux conception et implantation) :
 - 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)
 - Types des attributs et profils des méthodes

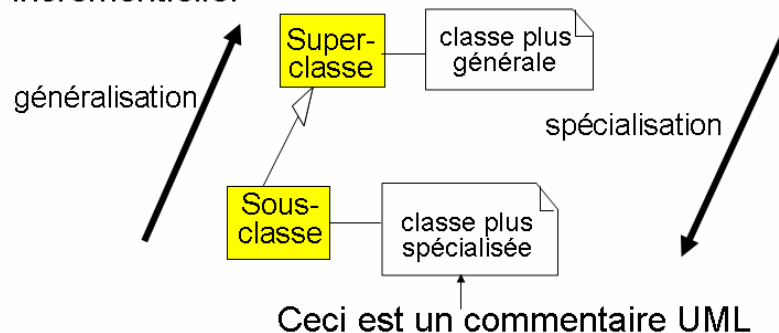
- opérations et méthodes de classe : soulignées
- méthodes abstraites : en italiques
- attributs calculés : notés / attribut : type

Ex :



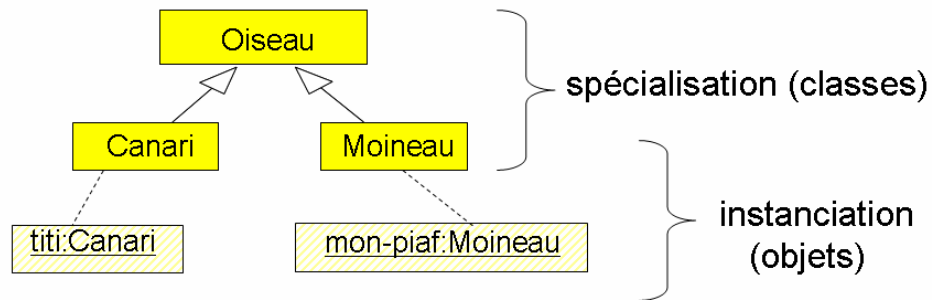
La hiérarchisation des classes

- La hiérarchisation des classes permet de gérer la complexité.
- **Généralisation** : factorisation des éléments communs de classes (attributs, opérations); favorise la réduction de la complexité.
- **Spécialisation** : adapter une classe générale à un cas particulier; favorise la réutilisation et la modification incrémentielle.



Remarques

- Ne pas confondre spécialisation et instanciation !

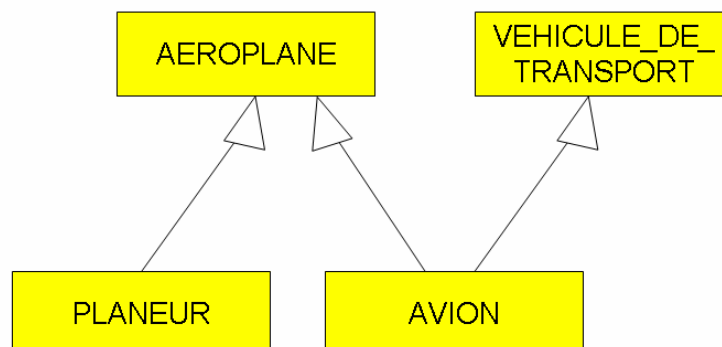


- Notation UML des objets : identificateur:classe ou :classe (objet anonyme)
- Les objets de la classe spécialisée héritent de la description des attributs (variables) et des opérations (méthodes) de la super-classe.
- Elles peuvent en ajouter d'autres et/ou en redéfinir certaines.

Héritage multiple (plusieurs super classes)

Autorisé dans la notation UML.

Ex :



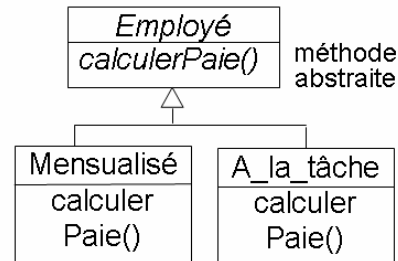
La collaboration des objets

Par « envoi de messages » (appels d'opérations/méthodes).

Un même message peut être traité de manière différente selon la nature de l'objet receveur (**polymorphisme**). L'émetteur n'a pas à connaître la classe du receveur.

Ex : paye d'employés de 2 types ('à la tâche' et 'mensualisés'). Par envoi du message `calculerPaie()` à toutes les instances de `Employé`. La bonne méthode est appliquée selon le **type effectif** de l'employé défini à la création de l'instance (**liaison dynamique**). Si un nouveau type d'employé est ajouté le programme, très simple, n'a pas à être modifié :

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

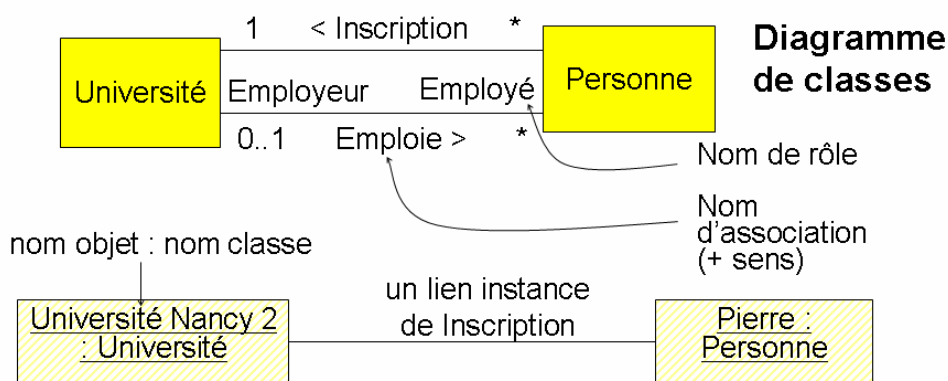


Le concept d'association

Exprime une connexion sémantique entre classes. Décrit un ensemble de **liens** (instances de l'association).

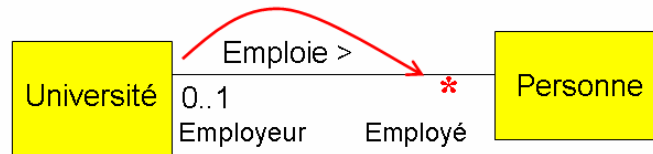
Le concept de rôle caractérise les extrémités.

Les **multiplicités** (cardinalités) caractérisent le nombre d'instances des classes impliquées dans l'association.



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

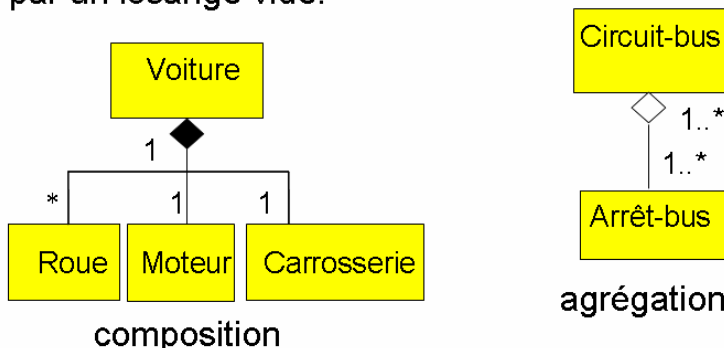


Contrairement aux cardinalités de Merise, les multiplicités UML sont placées du côté de la destination. Une Université emploie plusieurs Personnes : le symbole * est placé du côté Personne. A une Personne est associé zéro ou une Université (avec le rôle Employeur) : le symbole 0..1 est placé du côté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

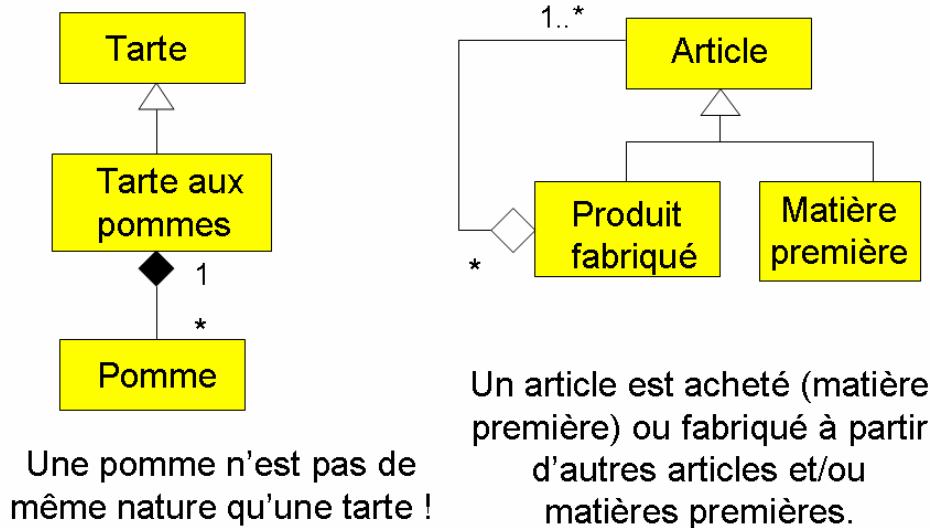
Associations spécialisées

L'agrégation est une association qui décrit une relation d'inclusion entre un tout (l'agrégat) et ses parties.

Si les durées de vie des objets sont liées on parle de **composition** (composant/composé), symbolisée par un losange plein du côté du composé; sinon (pour des durées de vie indépendantes) l'agrégation est symbolisée par un losange vide.

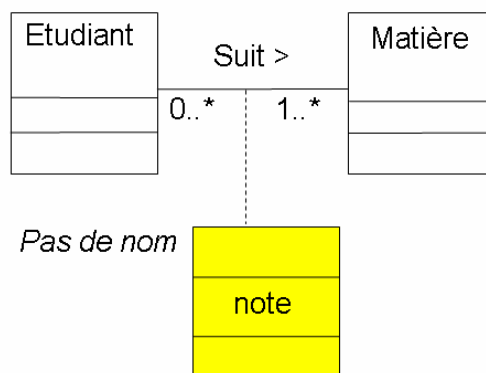


Ne pas confondre généralisation/spécialisation et agrégation ! Quand une classe est une spécialisation d'une autre elle est de même nature ce qui n'est pas le cas avec l'agrégation. Ces relations peuvent être associées.



Autres concepts

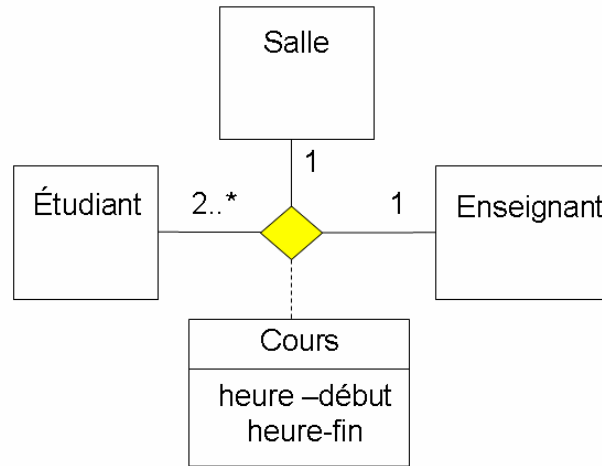
La **classe-association** : association porteuse d'attributs et/ou d'opérations, représentée comme une classe anonyme associée à l'association.



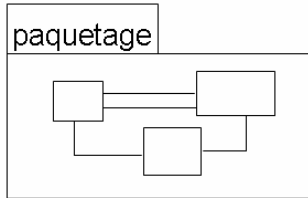
L'association d'arité n : représentée par un losange avec n 'pattes' auquel peut être associé une classe porteuse d'attributs et/ou d'opérations.

```
graph TD; Salle[Salle] ---|1| Assoc{ }; Etudiant[Étudiant] ---|2..*| Assoc; Enseignant[Enseignant] ---|1| Assoc; Assoc -.- Cours[Cours];
```

Le diagramme illustre une association d'arité n entre trois classes : Salle, Étudiant, et Enseignant. Une association, représentée par un losange jaune, relie ces classes. Les multiplicités sont 1 pour Salle, 2..* pour Étudiant, et 1 pour Enseignant. Une classe Cours, qui sert de classe porteuse d'attributs (heure-début, heure-fin), est associée à l'association par une relation de généralisation (ligne pointillée).

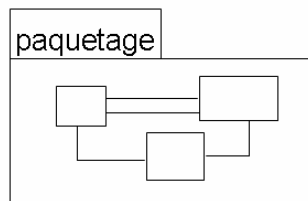


Le paquetage (package) : c'est une notion qui peut apparaître dans tous les diagrammes pour spécifier le regroupement d'éléments au sein d'un sous modèle (cas, classes, objets, composants, autres paquetages, ...).

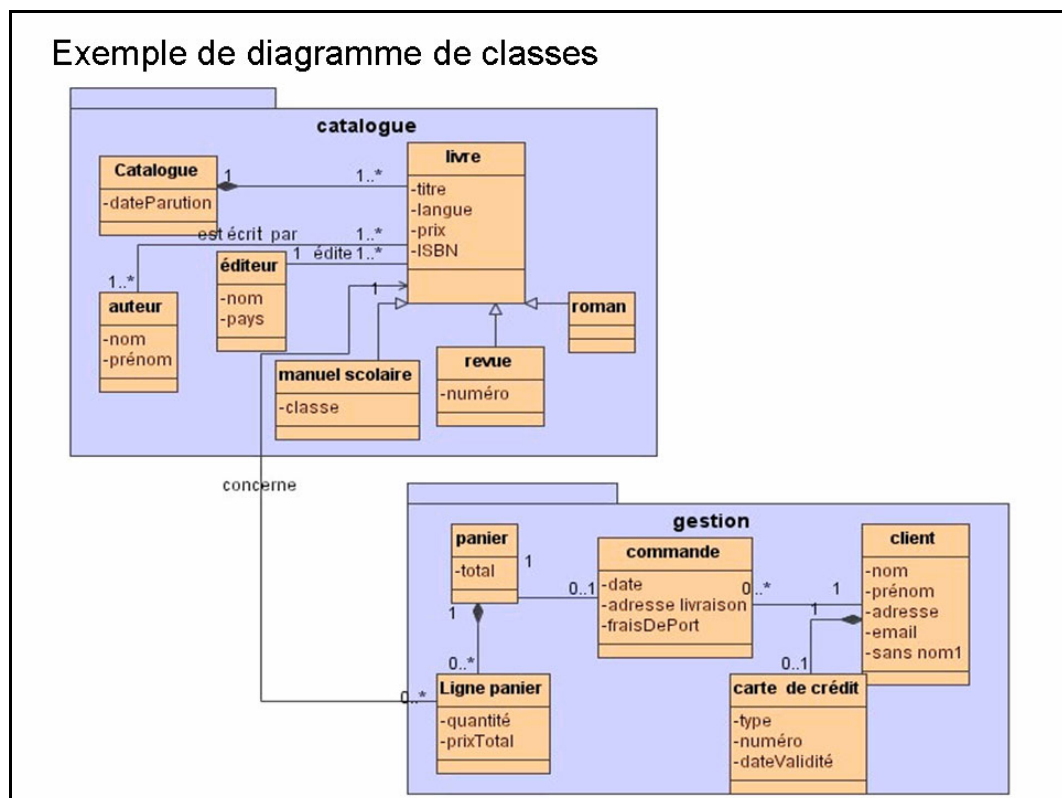
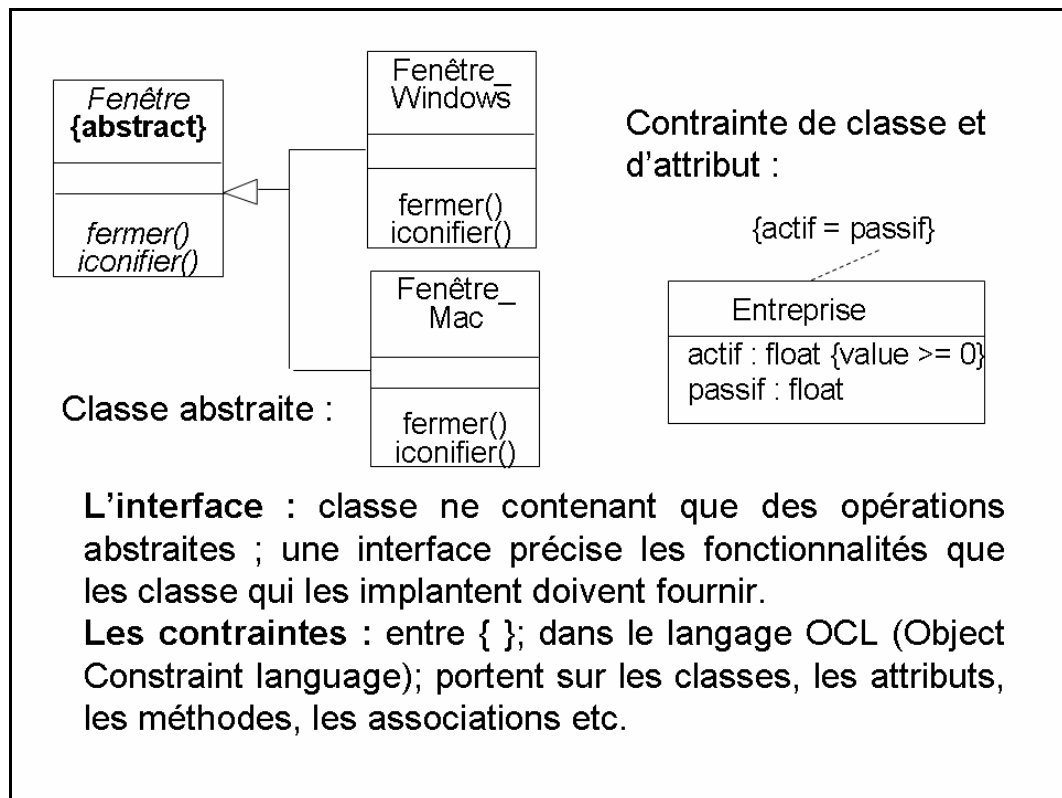


The diagram shows a large rectangular container labeled 'paquetage' at the top left. Inside this container, there are three smaller rectangular boxes. Two boxes are positioned at the top, connected by a horizontal line. A third box is positioned below them, connected to both of the top boxes by lines that form a 'V' shape, indicating a hierarchical or structural relationship within the package.

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.



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.



Les diagrammes d'interaction

- Application = {objets} qui interagissent pour réaliser les fonctions de l'application.
- Le comportement de l'application repose sur les **communications** entre objets (échanges de **messages** = appels de méthodes).
- Spécifiés par 2 types de diagrammes d'interaction : **diagrammes de collaborations** et **diagrammes de séquences**.

Utiles pour préciser

- la réalisation des cas d'utilisation au niveau de l'analyse (cahier charges)
- la dynamique d'un ensemble de classes ou d'objets au niveau de la conception ou de la programmation.

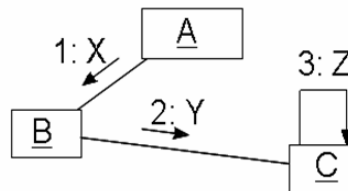
Les diagrammes de collaborations

Mettent l'accent sur l'organisation «spatiale» des objets (qui communique avec qui ?). Les messages peuvent être numérotés pour introduire une dimension temporelle.

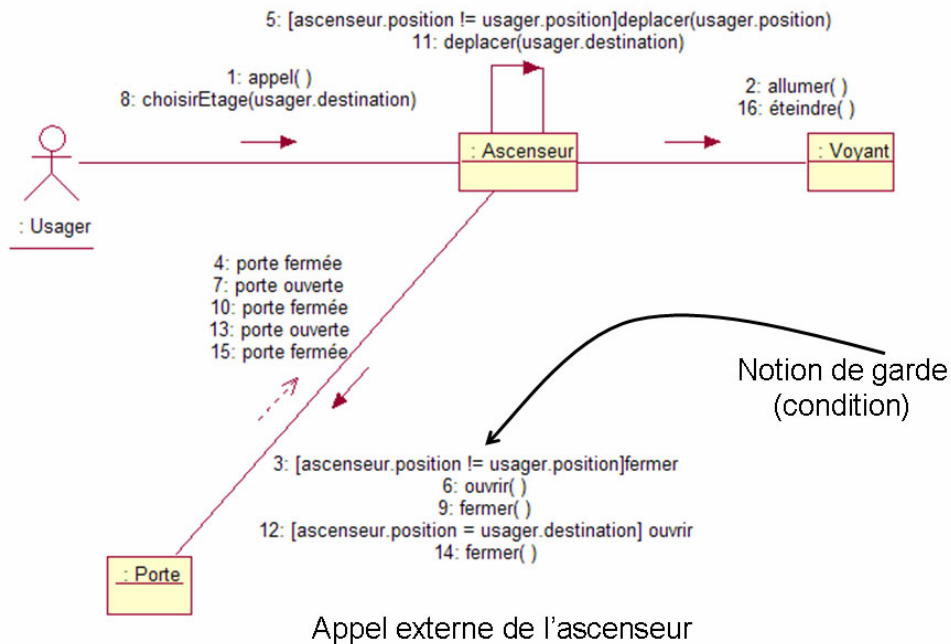
La notation UML permet de caractériser + ou - les messages :

- appel synchrone (avec attente de réponse)
- ← retour explicite ('return')
- appel asynchrone (sans attente réponse)
- ↪ appel réflexif (à soi même), etc.

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

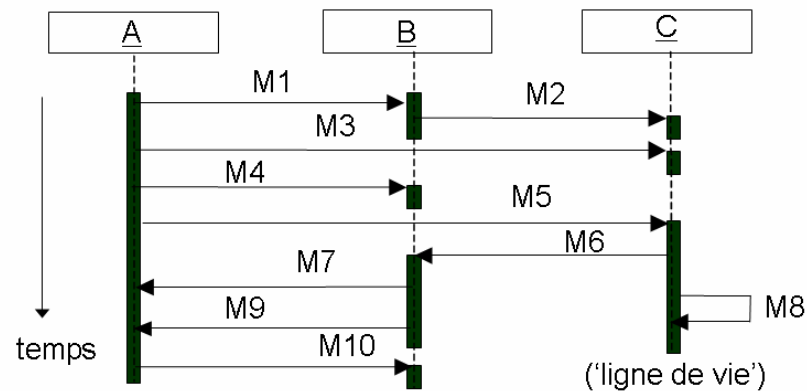


Exemple : ascenseur (cahier des charges)

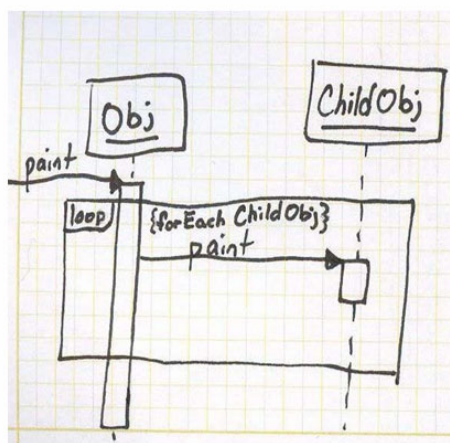


Les diagrammes de séquences

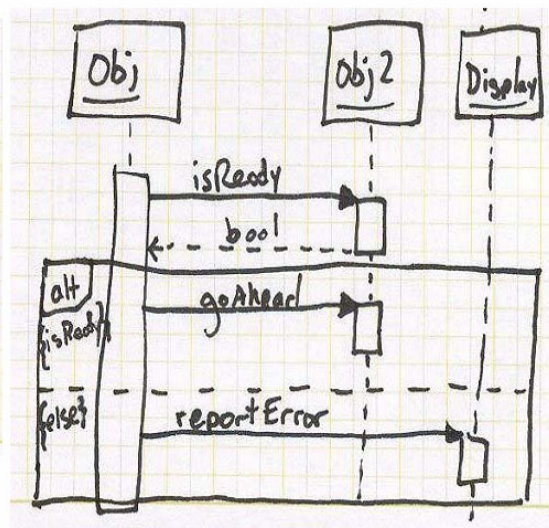
Ils mettent l'accent sur l'organisation temporelle. De nombreuses notations annexes permettent de préciser la nature des messages : message répétitif, conditionnel, réflexif, récursif, etc., et les données véhiculées.



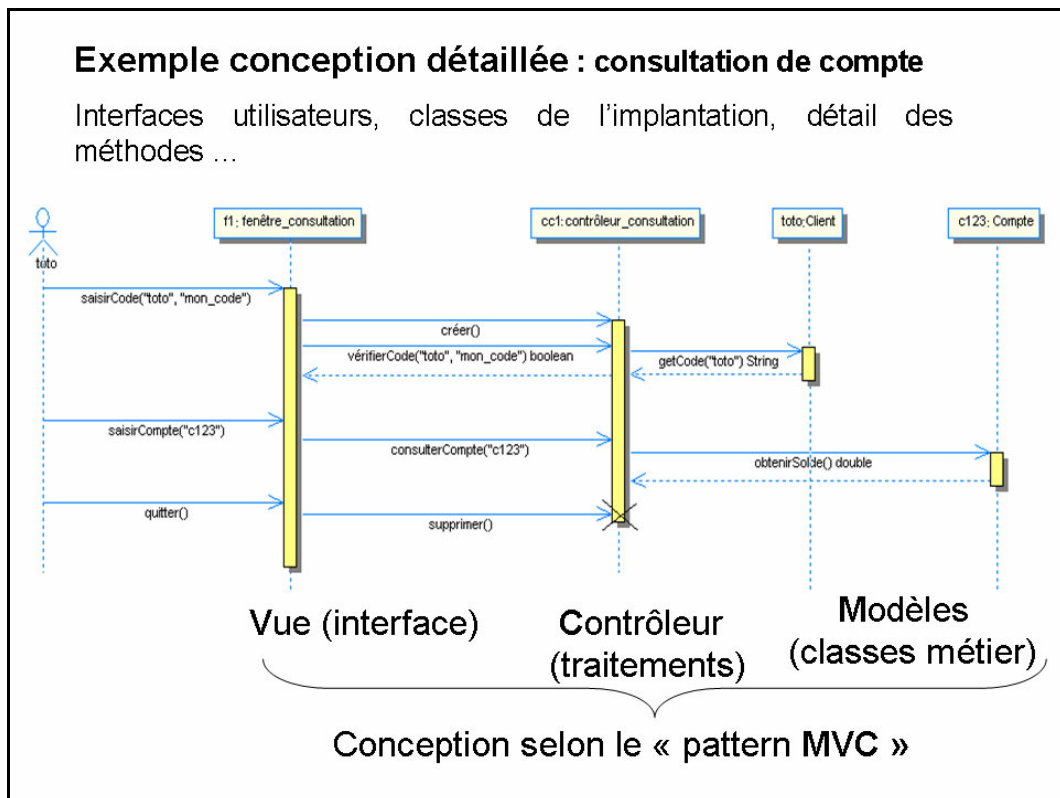
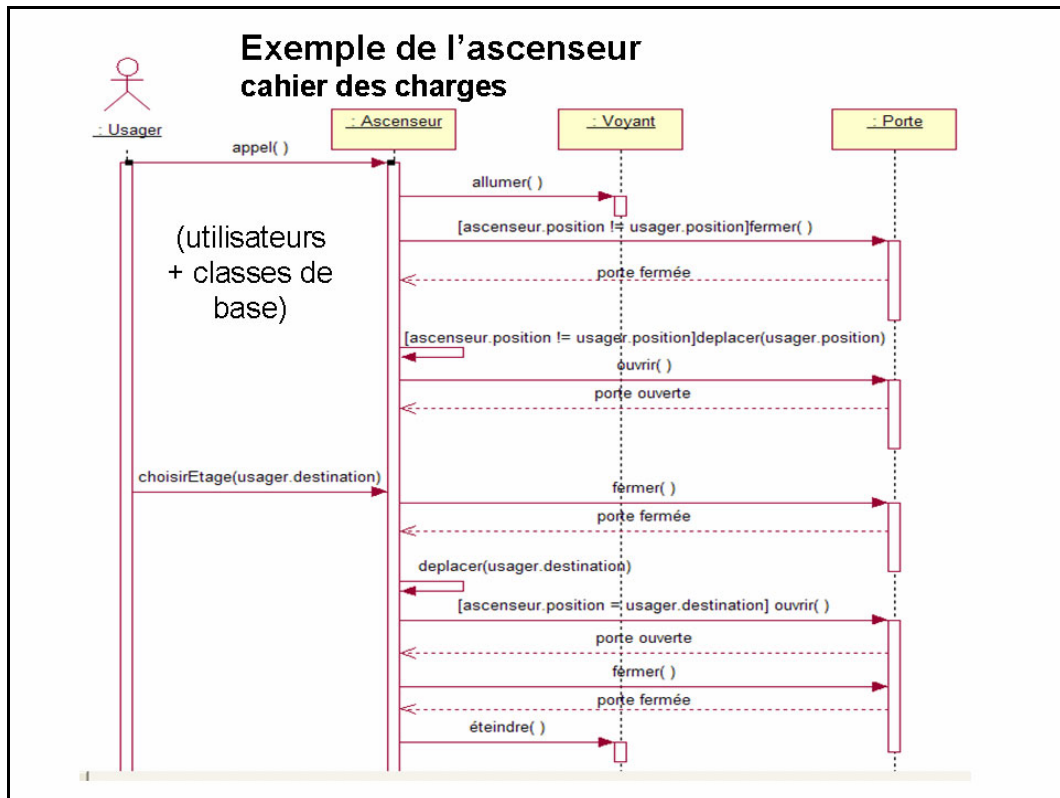
Exemples :



Message répétitif



Message conditionnel



Les diagrammes d'états et d'activités

Les diagrammes d'états et les diagrammes d'activités servent à préciser comment un système se comporte 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

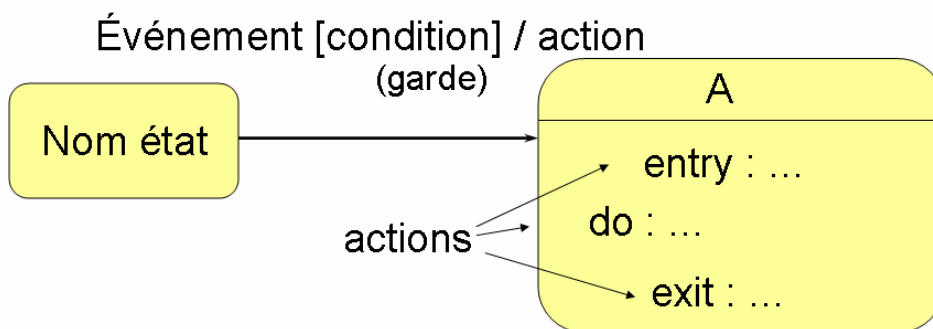
Les diagrammes d'états servent à décrire le cycle de vie des classes (objets) complexes.

Ces diagrammes décrivent tous les états possibles de l'objet et les transitions possibles entre ces états.



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 de l'état).

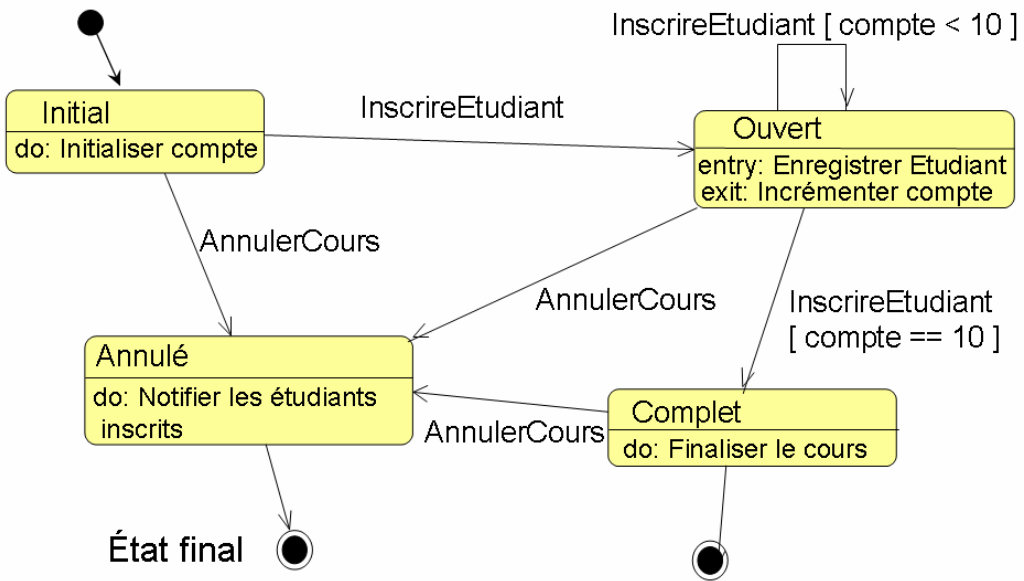


État de départ Transition État d'arrivée

● État initial

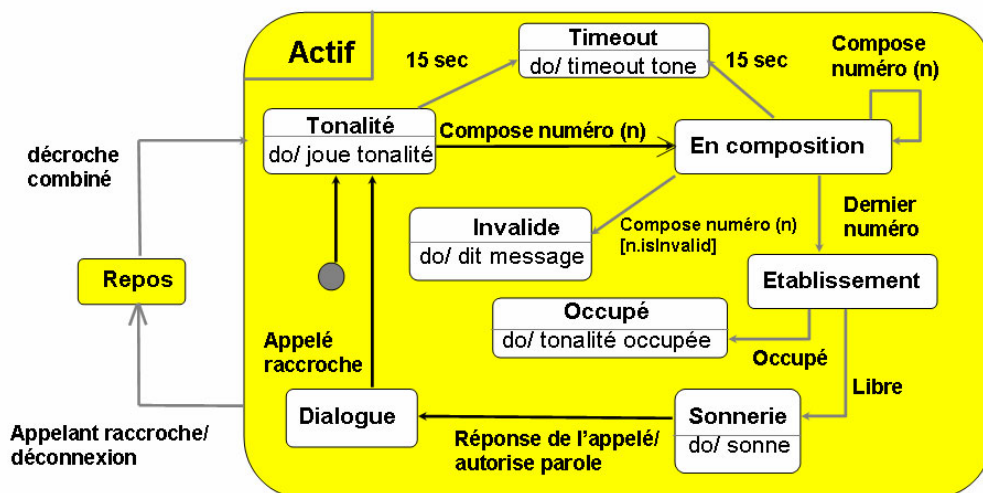
⦿ État final

Exemple 1 : États de la classe Cours

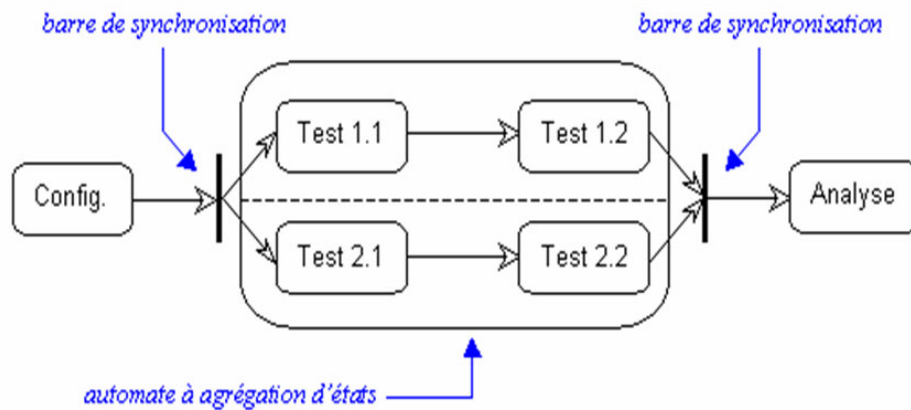


Les transitions correspondent aux appels de méthodes

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



Exemple 3 : Diagramme avec activités parallèles

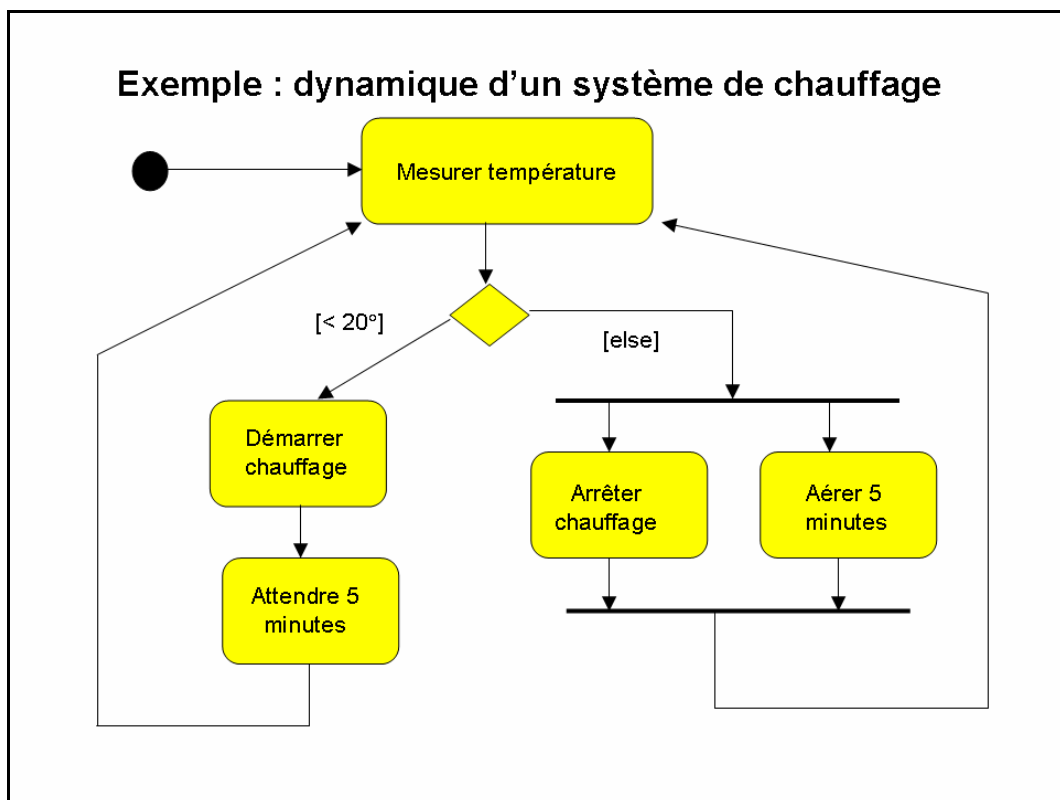
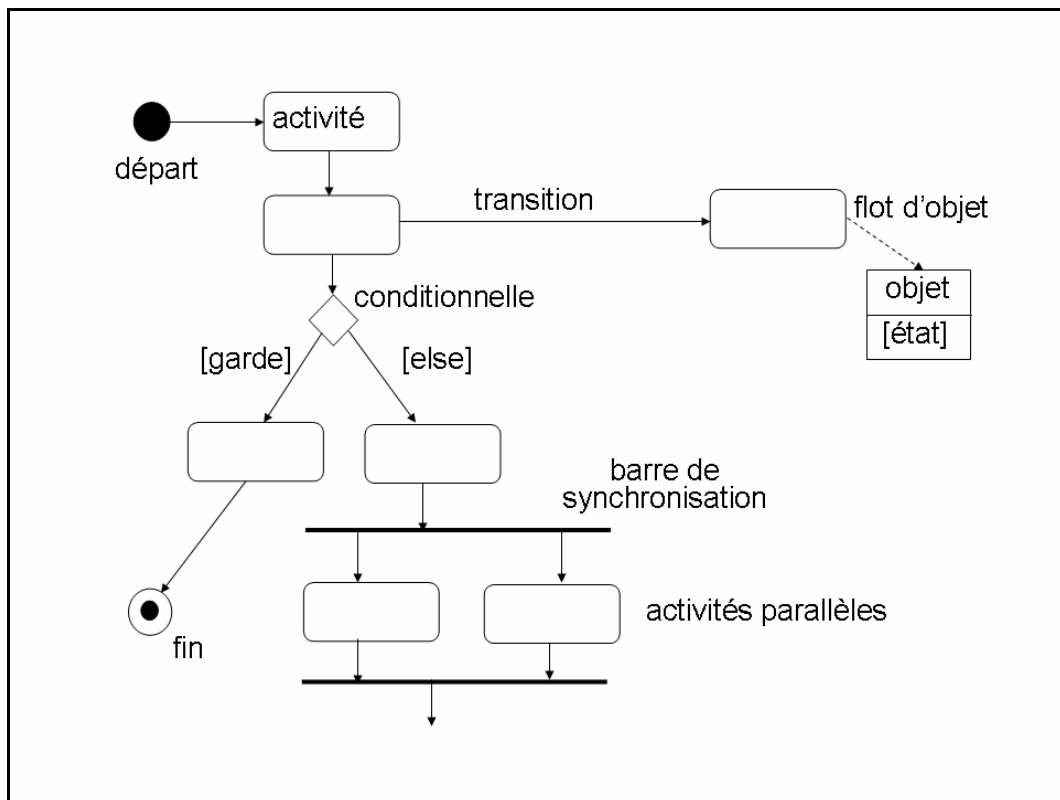


Les diagrammes d'activités

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.

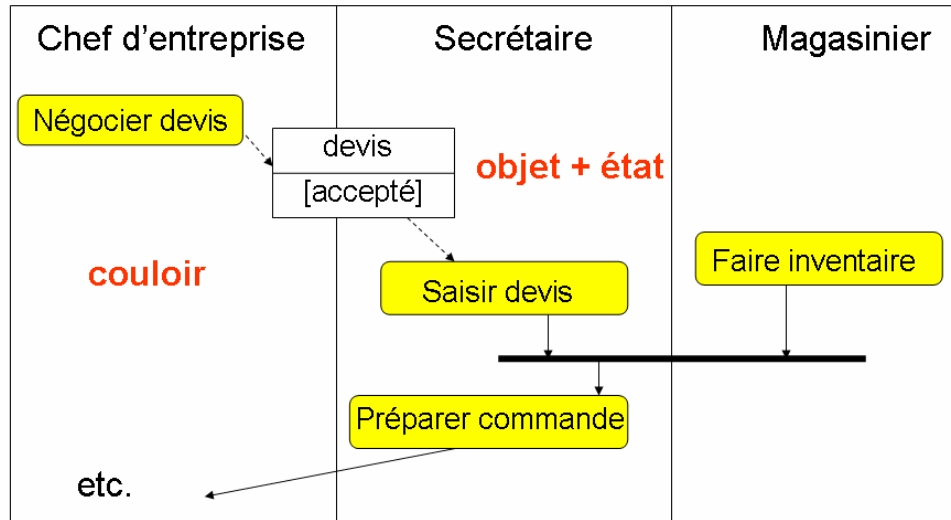
A un niveau **macroscopique**, les diagrammes d'activité permettent de décrire des **enchaînements de fonctionnalités** (« processus métiers »). Ils complètent donc bien les cas d'utilisation au niveau de l'analyse des besoins.

A un niveau **microscopique**, les diagrammes d'activité permettent, par exemple, de décrire l'**algorithme d'une action complexe (méthode)**.



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.

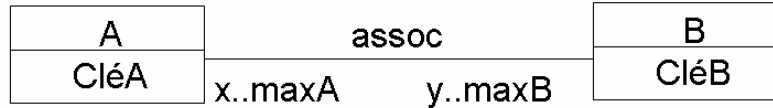
Ex:

Rectangle
largeur
hauteur
agrandir(..)
donnerAire()

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

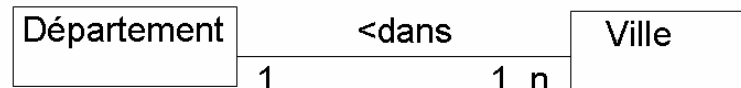
2. Traduction des associations

La solution dépend des multiplicités :



	maxA <= 1	maxA > 1
maxB <= 1	une des trois solutions du tableau	CléB dans A comme clé étrangère
maxB > 1	CléA dans B comme clé étrangère	créer table assoc avec comme clé CléA et CléB

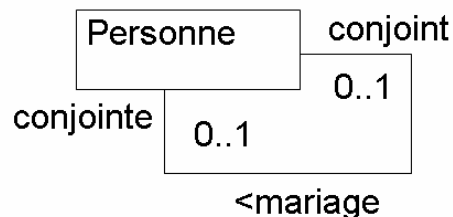
Exemples:



Département (N°Dep, ...)

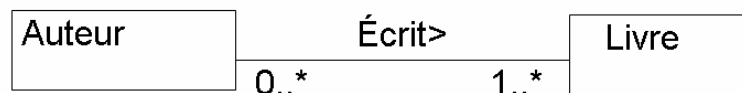
Ville(NomVille, N°Dep, ...)

Il faut bien entendu que $Ville.N^{\circ}Dep \subseteq Département.N^{\circ}Dep$



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 $\text{Ecrit.N°Auteur} \subseteq \text{Auteur.N°Auteur}$

et que $\text{Ecrit.N°ISBN} \subseteq \text{Livres.N°ISBN}$

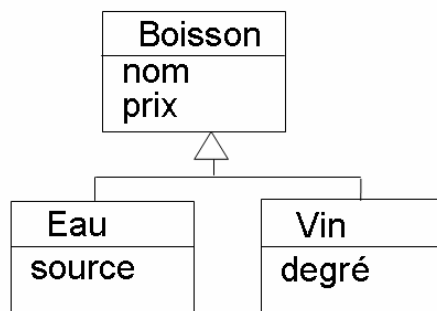
(contrainte d'intégrité référentielle)

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 $\text{type} \in \{\text{eau}, \text{vin}\}$

On peut créer une vue pour retrouver les sous classes complètes :

```

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

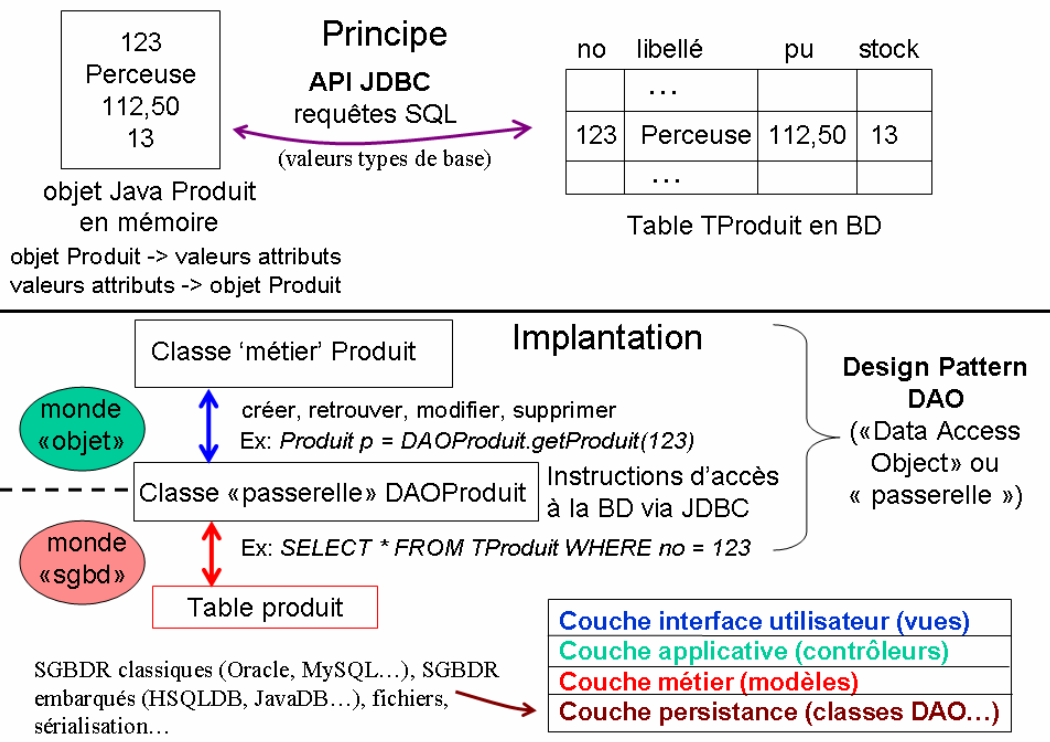
Solution 2 : une relation par classe; le type est défini par le présence dans une relation donnée.

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

On peut créer une vue pour retrouver la description complète d'une sous classe :

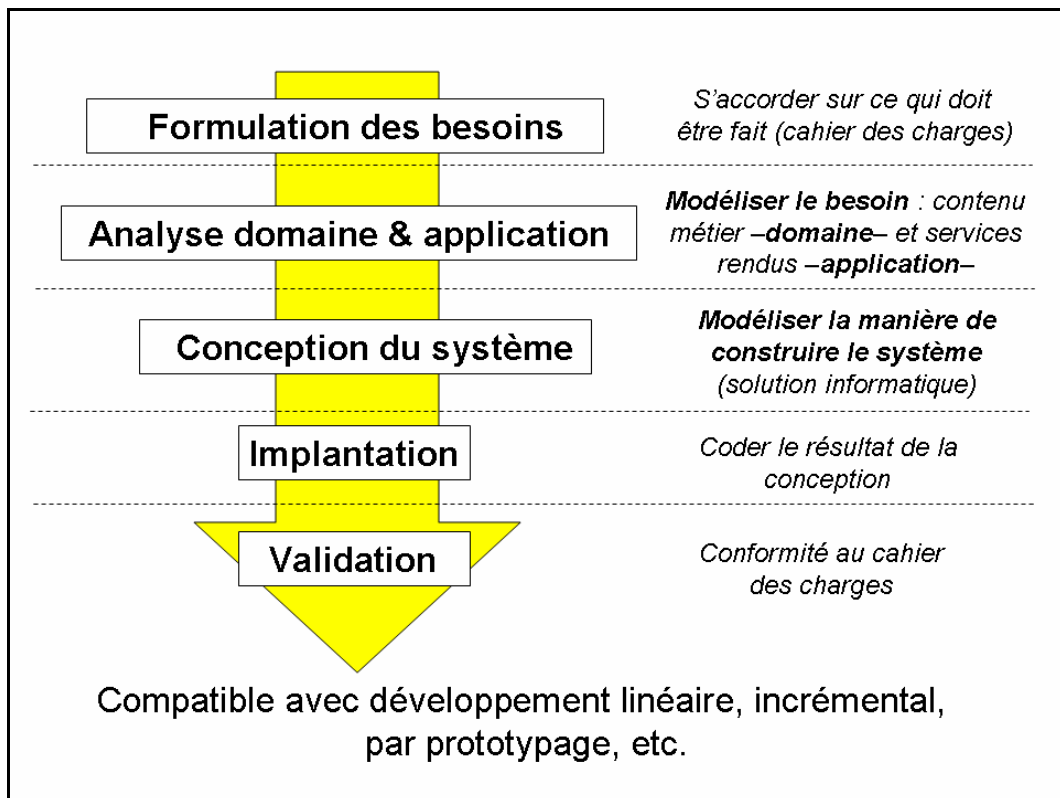
```
CREATE VIEW EauComplet AS
SELECT e.nom, b.prix, e.source
FROM Boisson b, Eau e
WHERE b.nom = e.nom
```

Mise en œuvre : Java vers SGBDR



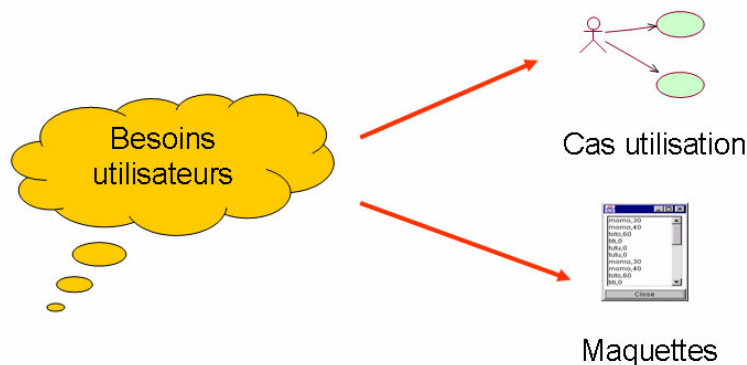
Le processus de développement objet

- UML est un langage de modélisation objet compatible avec tous les styles de développement de logiciel :
 - linéaire : analyse → conception → codage → test,
 - incrémental :
 - implantation d'un sous-ensemble des fonctionnalités,
 - release courte,
 - intégration progressive (cœur → incréments successifs)
 - « refactoring »,
 - tests de non-régression automatisés.
 - par prototypage : réalisation d'un prototype exécutable qui prouve la faisabilité quand les besoins ne sont pas très précis au départ → construction du système à partir du prototype ou non.
- Cependant, une certaine démarche intellectuelle est sous jacente à la notation.



Formulation des besoins

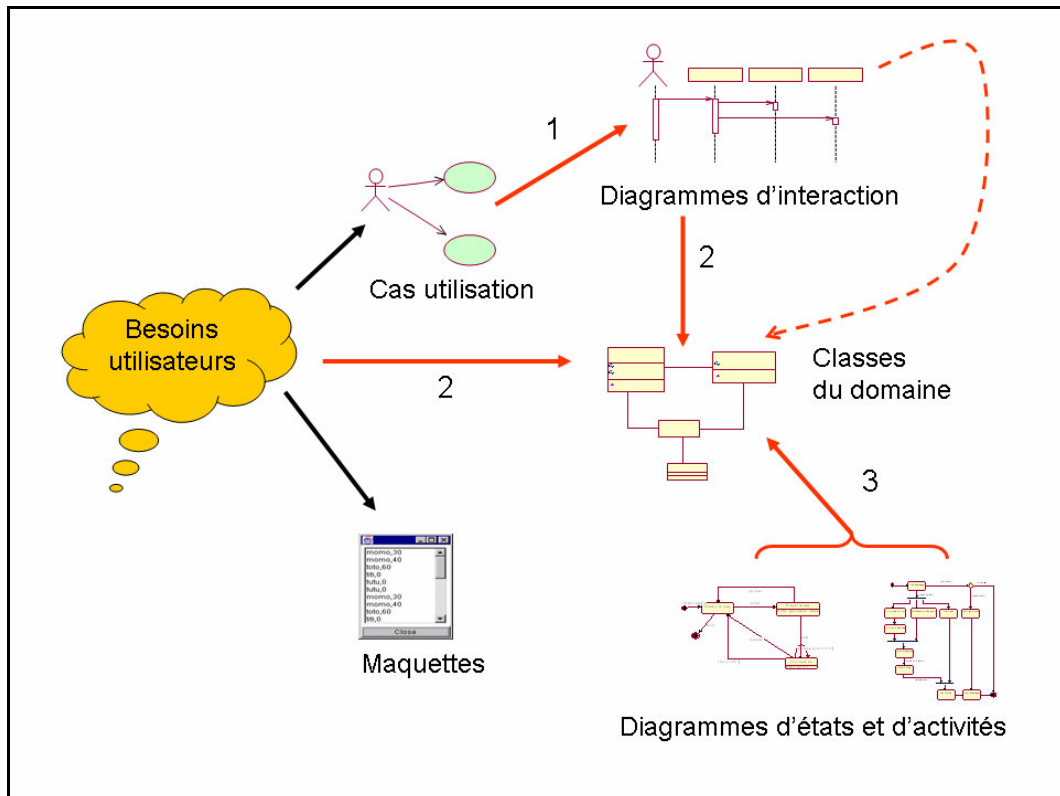
- Les cas d'utilisation et scénarios expriment un point de vue externe et fonctionnel sur le système. Ils constituent la base de la description des besoins.
- Des maquettes du systèmes (interfaces utilisateurs non opérationnels) peuvent aider à formuler les besoins en impliquant les utilisateurs.



Analyse domaine & application

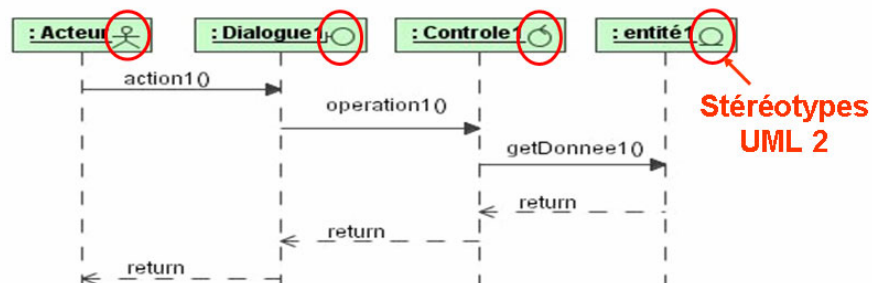
- Les diagrammes d'interaction que l'on peut associer aux cas d'utilisation font apparaître des classes d'objets candidates et donc passer d'une vue « fonctionnelle » à une vue « objets ».
- On «raconte les cas» en faisant communiquer des classes d'objets introduites au fur et à mesure.
- Les schémas 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.

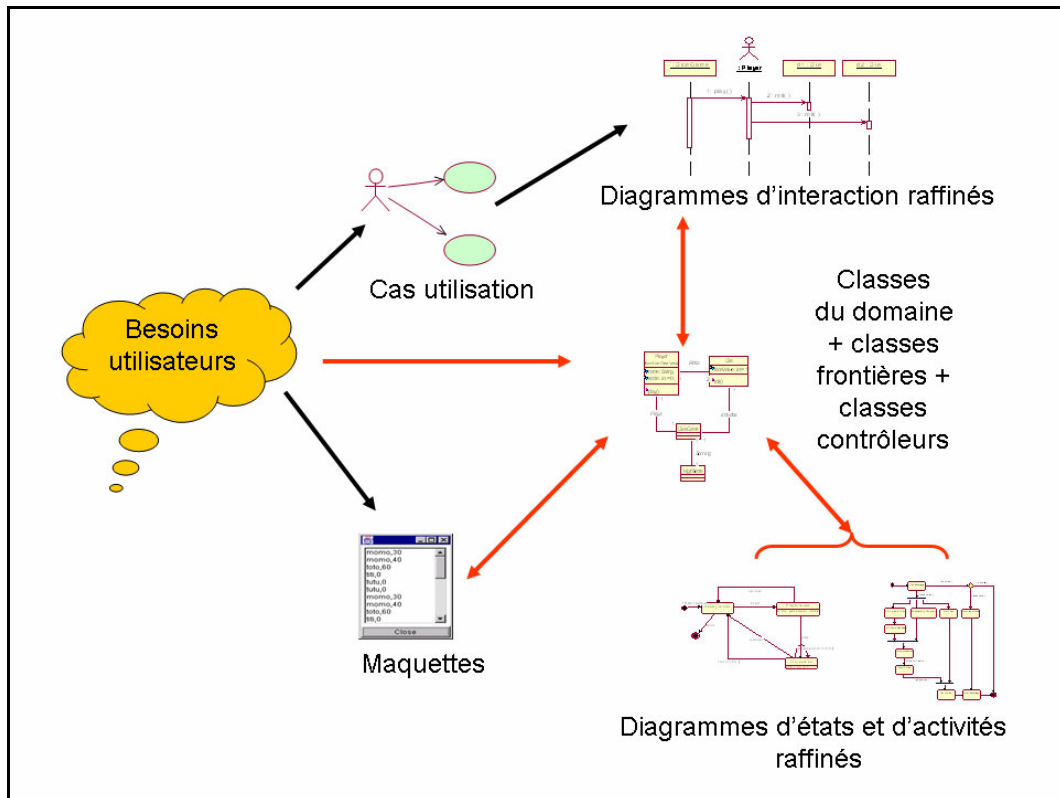
- Ces schémas de classe sont enrichis au fur et à mesure où des éléments nouveaux apparaissent.
- Ces éléments nouveaux peuvent être obtenus à ce stade par un approfondissement de l'analyse en utilisant diagrammes d'états et d'activités pour modéliser les aspects les plus complexes.



- On passe ensuite de l'analyse du domaine à **l'analyse de l'application** : dialogue avec le système, logique de l'application...
- Dans les diagrammes d'interactions et de classes on complète les classes « métiers » (modèles, entités) par des classes « dialogue » non détaillées à ce stade (vues, écrans) et des classes contrôleurs pour gérer les interactions (schéma MVC).

Ex :





Conception du système

Sont introduits :

- une **architecture** (exprimée sous forme de packages) le plus souvent en couches.
- **des classes techniques liées** :
 - aux détails des interfaces utilisateurs,
 - aux structures de données et à l'accès aux données (DAO) (fichiers - texte, XML -, sérialisation, bases de données relationnelles, bases de données embarquées...)
 - à la concurrence des traitements et à la distribution des classes sur le réseau (utilisation d'un « middleware » type RMI, CORBA, SOAP...)
 - aux **patrons de conception** (« design patterns ») qui sont des « briques » de schémas de classes standardisées qui répondent à des problèmes fréquemment rencontrés.
- Les modèles peuvent être **raffinés** (ex: algorithmes) et **optimisés** (ex: chemins d'accès aux classes).

Architecture classique en 3 couches

Présentation
(Packages UI)



Classes de dialogue

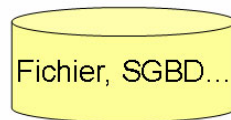
Application
(Packages applicatifs et utilitaires)



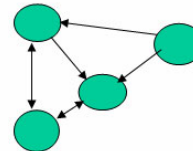
Implantation des cas : classes métiers + contrôleurs

Persistance, distribution
(Packages techniques)

Classes techniques (ex: couche accès aux données DAO)



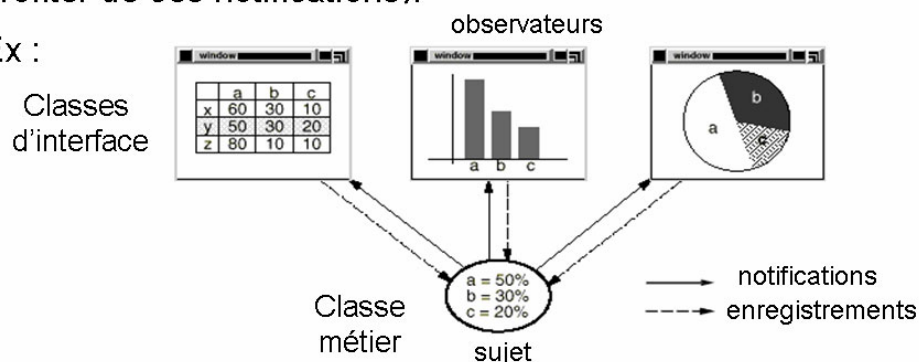
Fichier, SGBD...



Exemple de patron de conception

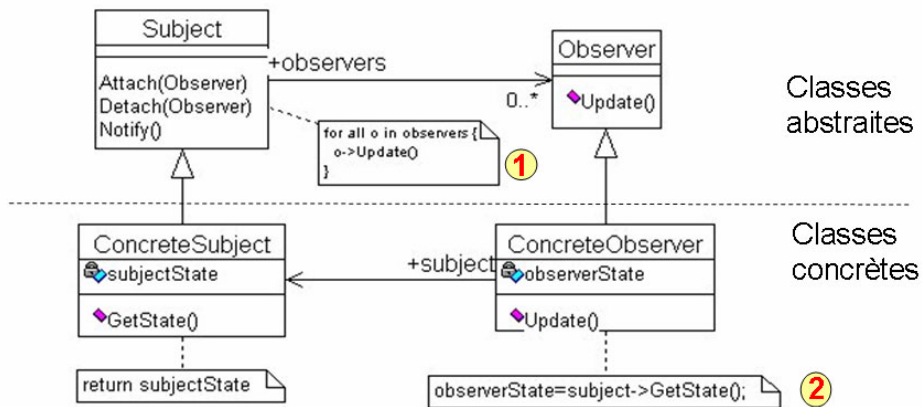
Le pattern 'Observer' répond au besoin d'un lien 1 vers n entre objets. Tout changement du **sujet** doit être automatiquement notifié à tous les **observateurs** (dont le nombre est inconnu au départ et qui s'enregistrent pour profiter de ces notifications).

Ex :



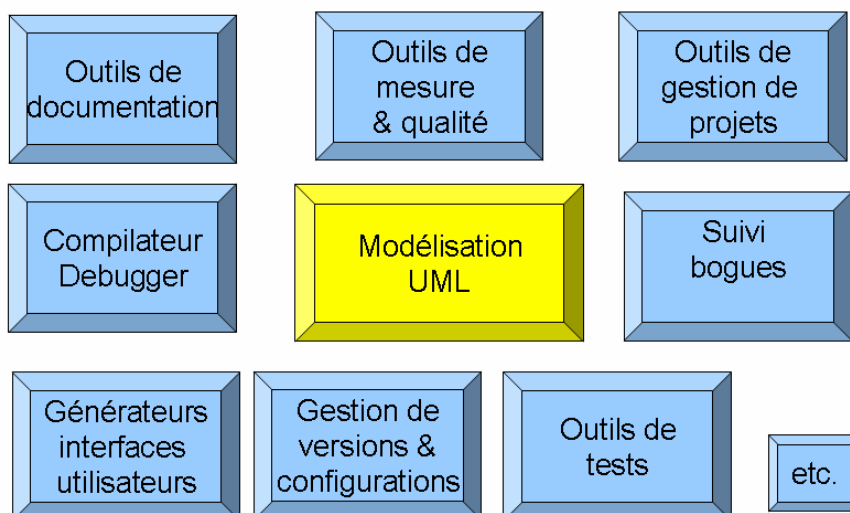
Remarque : même principe que les auditeurs (listeners) Java mais en sens inverse ! Des objets métiers vers l'interface et non de l'interface vers les autres classes.

1. Lorsque le sujet est modifié, il doit avertir (méthode **Notify**) les observateurs qui se sont enregistrés (méthodes **Attach/Detach**).
2. Lorsqu'un observateur est averti de la modification d'un sujet, il se met à jour (méthode **Update**). Pour cela, il interroge le sujet pour connaître son nouvel état (méthode **GetState()**)



Travailler avec des outils

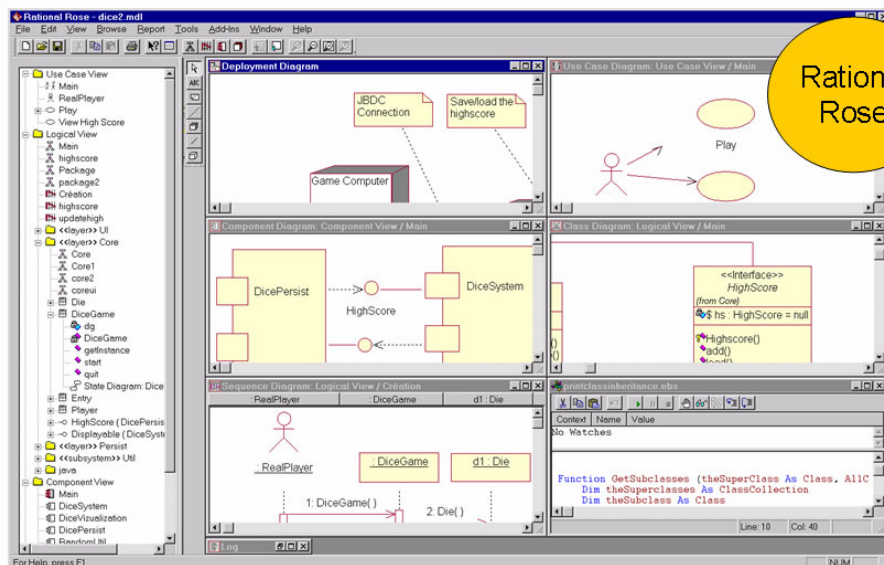
L'outil de modélisation UML est au cœur de l'outillage pour le développement logiciel.



Modélisation UML

- Dessin des diagrammes : en respectant la syntaxe & sémantique des diagrammes.
- Référentiel : tous les éléments créés dans les modèles sont stockés dans une base de données.
- Navigation : à travers les documents.
- Support multi-utilisateurs : plusieurs utilisateurs doivent pouvoir travailler en parallèle sur le même modèle (gestion de versions).
- Génération de code : génération de squelettes dans différents langages cibles.
- « Reverse engineering » : lire le code existant et générer ou mettre à jour les diagrammes existants (certains...).
- Intégration avec les autres outils du développeur : éditeur de code, compilateur, debugger, gestionnaire de versions...

- Couverture de tous les niveaux d'abstraction : des packages jusqu'au code.
- Import/export : vers d'autres outils de conception.



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, C++ ou VB/C#/.net

- UML permet bien entendu 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 fondamentalement 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 a 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

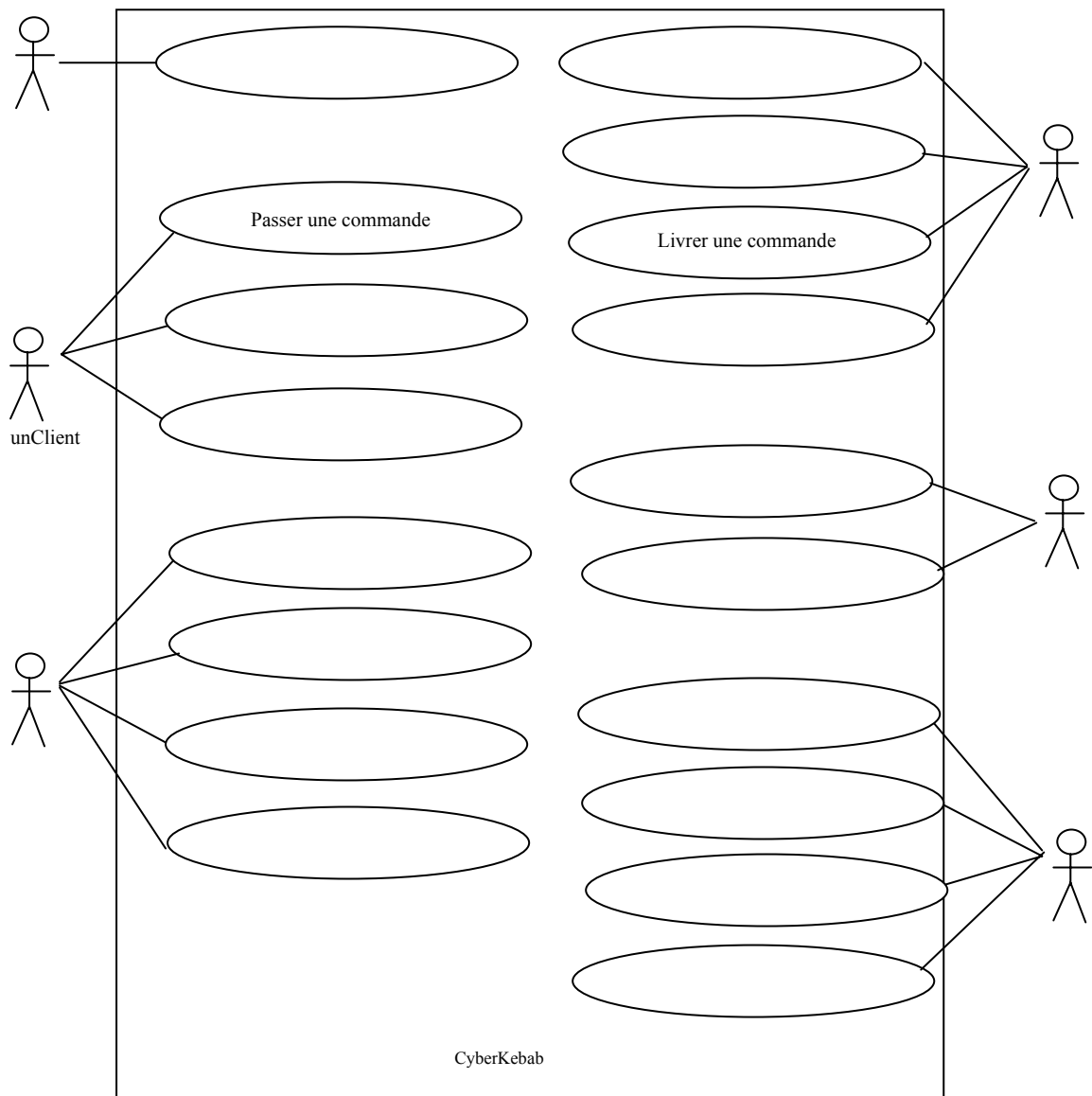
particulier; c'est le gérant qui affecte la commande à un PointKebab et à un livreur. Le gérant cherche en général à optimiser la distance parcourue ainsi que les activités des PointKebabs et des collaborateurs.

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. Ornithologie

Une association d'ornithologie (d'étude des oiseaux) souhaite réaliser une application de gestion des observations faites par ses adhérents, appelée PIAFS. Son objectif principal est de stocker toutes les observations et d'établir des cartes de présence des espèces d'oiseau sur le territoire géré par l'association.

Pour chaque observation, l'adhérent qui l'a réalisée saisit : son nom, le nom de l'espèce observée (nom courant ou nom scientifique), le nombre d'individus observés, le lieu de l'observation (nom ou code postal de la commune et un descriptif précis du lieu), la date et heure de l'observation, les conditions météo au moment de l'observation.

Les observations saisies par les adhérents sont dans un état 'à valider'. Tant qu'elles ne sont pas validées par un adhérent salarié de l'association elles restent modifiables.

Un adhérent salarié peut valider une observation. Lors de cette opération le logiciel contrôle automatiquement que le nom d'espèce est connu (toutes les espèces connues sur ce territoire sont répertoriées), que l'observateur est adhérent de l'association, que la commune existe sur le territoire (toutes les communes du territoire sont répertoriées), que les dates et heures sont correctement saisies et que tous les champs sont remplis. Au vu de ces contrôles et après lecture de toutes les informations saisies, l'adhérent salarié fait passer l'observation dans l'état 'validé' ou dans l'état 'non validé'. Seules les observations validées sont conservées, les autres sont automatiquement supprimées chaque semaine.

A partir des observations validées, PIAFS doit permettre d'afficher :

- des cartes de présence par espèce avec un cumul du nombre d'individus de l'espèce observés (ce traitement peut être demandé par tout adhérent),
- des cartes des observations réalisées par chaque adhérent (ce traitement ne peut être demandé que par les adhérents salariés de l'association).

Ces cartes sont construites grâce à la connaissance des coordonnées géographiques des communes.

Travail à faire

Dessiner le diagramme des cas d'utilisation du logiciel PIAFS.

5. Cyber-cartes

L'application web « cyber-cartes » doit permettre :

- à un internaute de s'inscrire pour créer un compte; il choisit alors un login et un mot de passe; il devient, après validation du compte par l'administrateur, un client de « cyber-cartes »;
- à un administrateur de se connecter ; après quoi il peut valider un ou plusieurs comptes correspondant chacun à une demande d'un internaute et se déconnecter ;

- à un client de se connecter à l'application ; après quoi il peut :
 - créer une (ou plusieurs) carte(s) électronique(s) avec obligatoirement un texte personnalisé et dans la(les)quelle(s) il peut en plus :
 - ajouter une ou plusieurs images animées,
 - ajouter une mélodie.
 - expédier une ou plusieurs cartes par e-mail à un destinataire dont il fournit l'adresse e-mail sous la forme d'une chaîne de caractères.
 - se déconnecter.

Travail à faire

Dessinez le diagramme des cas d'utilisation de « cyber-cartes ».

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, équilibre, 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 pour lequel on a déjà construit les cas d'utilisation.

Travail à faire

Dessiner le diagramme des classes du domaine avec les classes, les associations, les relations d'héritage, les multiplicités (cardinalités), les attributs.

3. Ornithologie

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

Travail à faire

Dessiner le diagramme des classes du domaine avec les classes, les associations, les relations d'héritage, les multiplicités (cardinalités), les attributs.

4. 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.

5. 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).

6. 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()'.

7. 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.

8. Bataille navale

Le jeu de la bataille navale se compose d'un tableau de n lignes et m colonnes et d'un ensemble de bateaux positionnés sur ce tableau.

Chaque bateau comporte un ensemble de taille fixe de cases. Il y a les croiseurs qui comportent 3 cases, les escorteurs avec 2 cases et les sous-marins avec une seule case.

Chaque case est caractérisée par sa position dans le tableau (n° de la ligne et n° de la colonne) et par son état : intacte ou touchée.

Les bateaux doivent toujours être séparés par au moins une case vide.

Les sous-marins ont la possibilité de plonger. Lorsqu'ils plongent ils ne peuvent pas être touchés.

Exemple : tableau 10 sur 10 avec 2 bateaux de chaque type

	X								
				X			X	X	X
				X					
				X					
								X	X
	X								
	X								
					X				

Travail à Faire

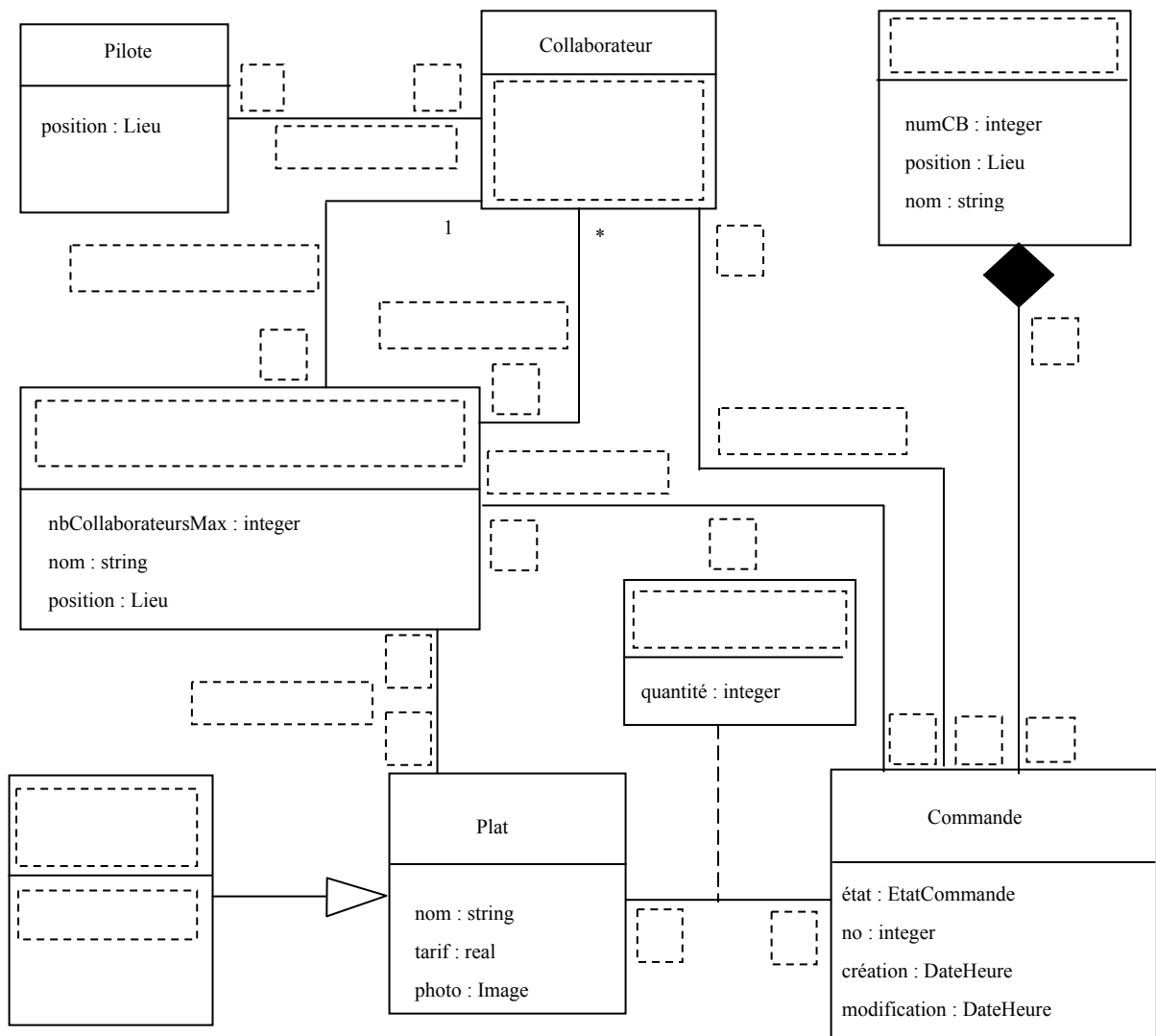
- 1) Donner le schéma de classes UML traduisant cette description du jeu (classes, associations, relations d'héritage, multiplicités, attributs).
- 2) Quand et comment prendre en compte la contrainte 'les bateaux doivent toujours être séparés par au moins une case vide' ?

9. Méta modèle UML

Donner le modèle de classes UML qui permet d'instancier n'importe quel diagramme de cas UML avec des instances d'acteurs, des instances de cas, des instances d'interactions, etc. Un tel modèle est souvent appelé un 'méta-modèle' car c'est un modèle qui décrit tous les composants d'un autre modèle.

On rappelle qu'un diagramme de cas d'utilisation contient 6 types de composants : acteur, cas, interaction (entre un acteur et un cas), héritage (entre 2 acteurs ou entre 2 cas), relation d'extension (entre 2 cas) et relation d'inclusion (entre 2 cas).

10. Annexe



Lieu est un type permettant de situer dans l'espace.

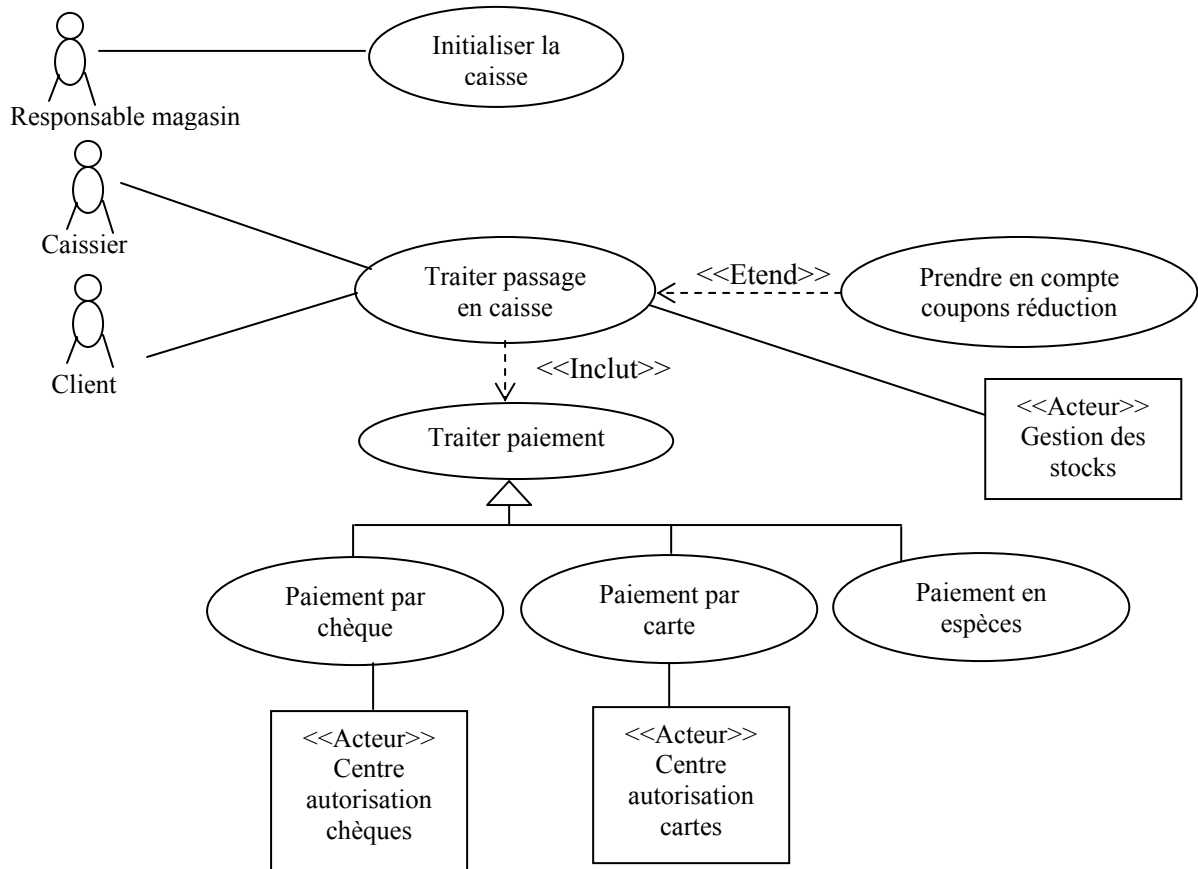
DateHeure est un type permettant de situer dans le temps.

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. Ornithologie

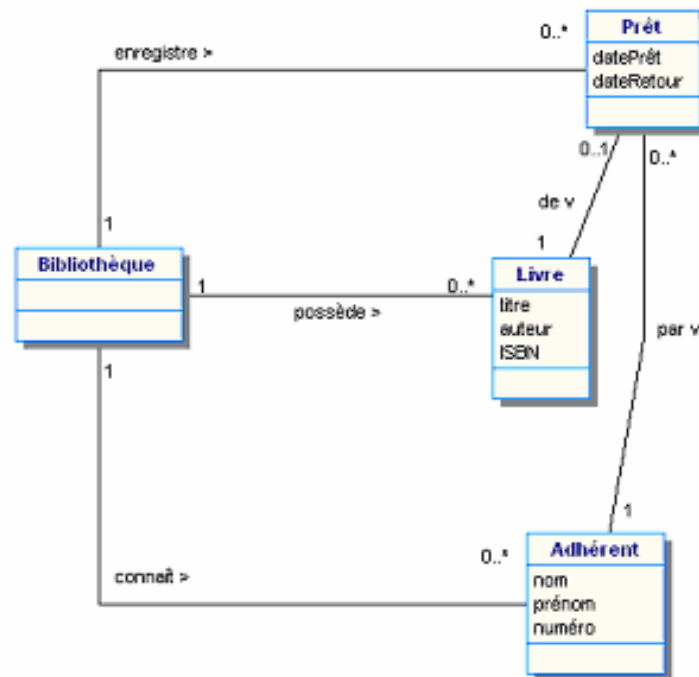
On reprend l'énoncé pour lequel on a déjà construit les cas d'utilisation.

Travail à faire

Dessiner le diagramme de séquences correspondant à la saisie d'une observation.

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

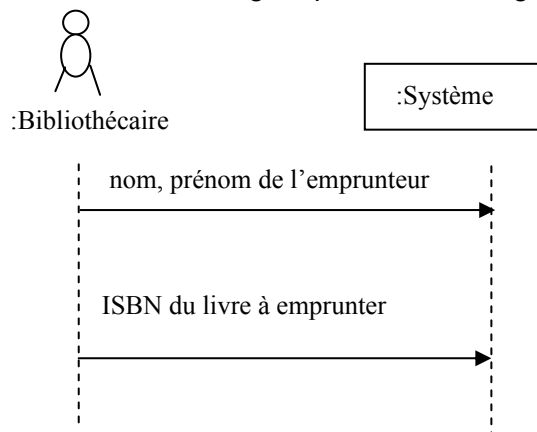
Au cours de l'analyse de la gestion d'une bibliothèque on a retenu les classes métier suivantes.



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.



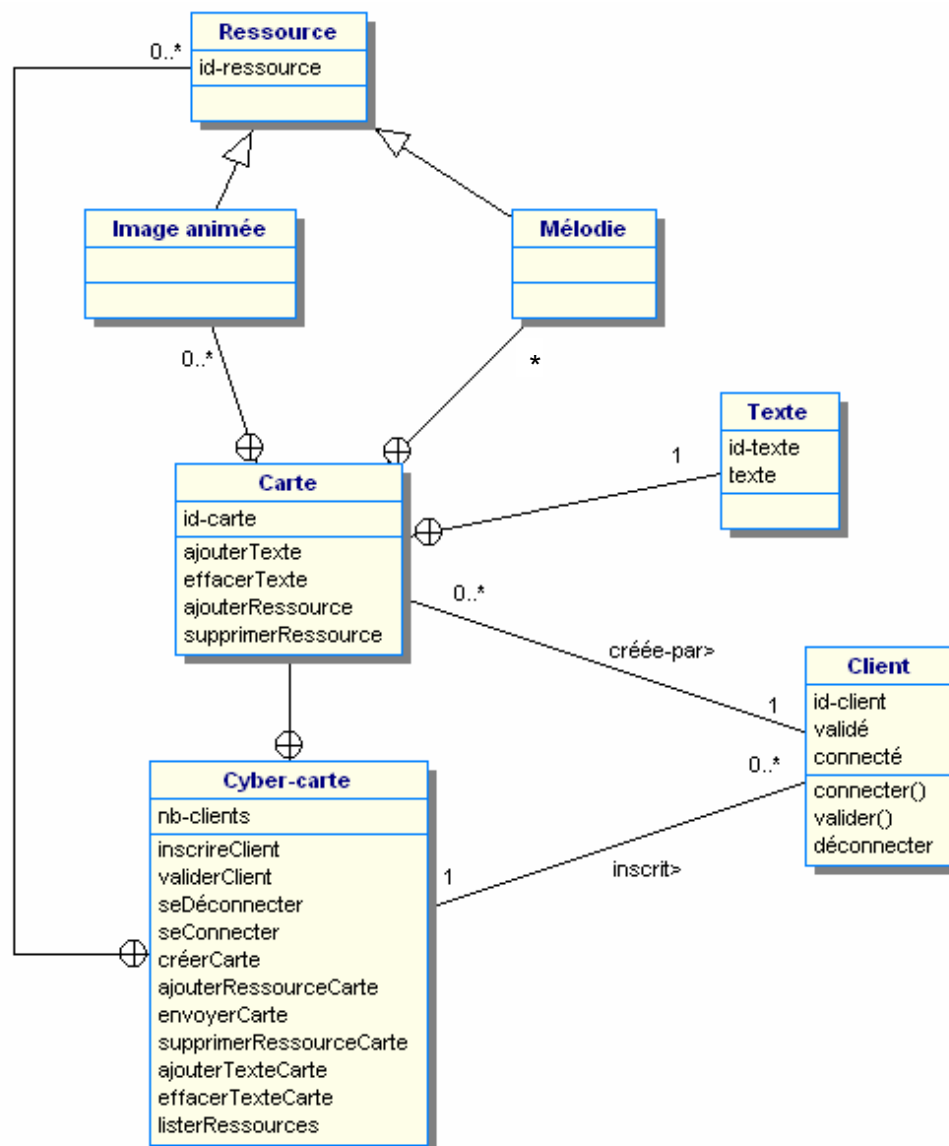
Les tables de la classe Bibliothèque (table de tous les objets livre, table de tous les objets adhérents et table de tous les prêts pour une durée 15 jours) ont des méthodes :
- trouverLivre(ISBN), trouverAdhérent(nom, prénom) et trouverPrêt(n° prêt) qui retournent les objets cherchés,

- 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.

4. Cyber-cartes

On reprend l'énoncé pour lequel on a déjà construit les cas d'utilisation. En vous basant sur le diagramme de classes qui suit, dessiner un diagramme de séquences décrivant un client qui se connecte, crée une carte électronique, ajoute un texte et une image animée, l'envoie à une adresse e-mail et se déconnecte.



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

- Modéliser avec un diagramme d'activités la dynamique de ce système.
- 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

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

3. Ornithologie

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

Travail à faire

Dessiner le diagramme d'états d'une observation.

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.

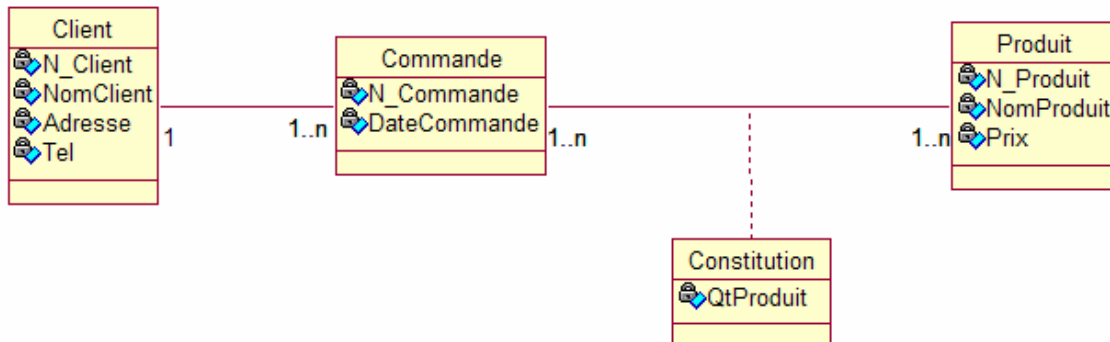
On supposera qu'un thread n'envoie pas d'événement. Il ne fait que les recevoir. On supposera que les événements reçus par le thread sont : « début », « ressource attendue », « ressource OK », « processeur OK », « fin » :

- « début » correspond au démarrage du thread (start en java, execv 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

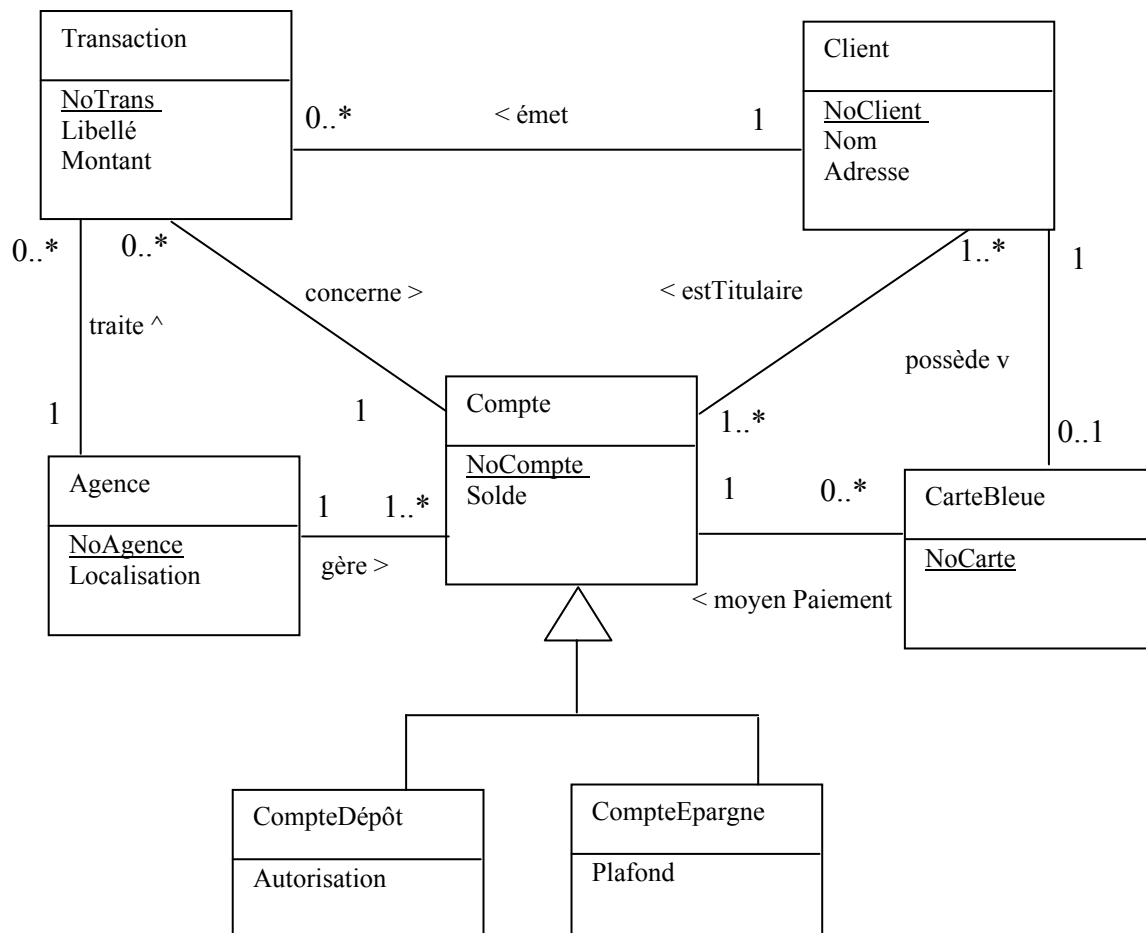
Exercice 1

Traduire le diagramme de classes UML suivant en relationnel.



Exercice 2

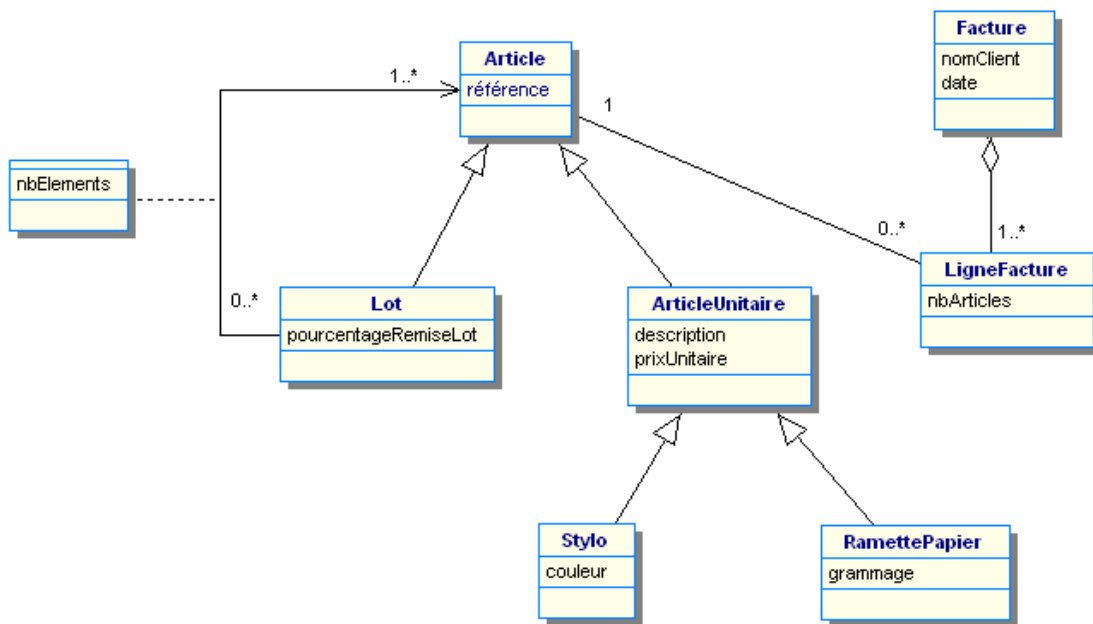
Traduire le diagramme de classes UML suivant en modèle logique relationnel.



Exercice 3

Soit le schéma de classes ci-dessous.

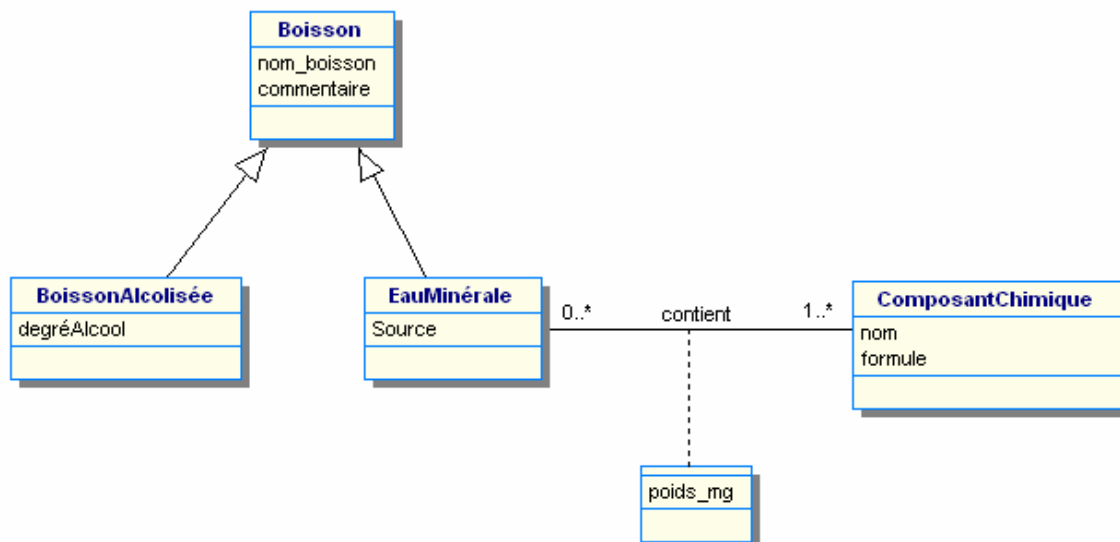
- D'après ce schéma, un lot peut-il contenir un lot ?
- Traduisez ce schéma en relationnel avec la stratégie qui consiste à associer une table par classe de l'arbre d'héritage,
- Même question avec la stratégie qui consiste à associer une seule table à tout l'arbre d'héritage.



Exercice 4

Soit le schéma de classes ci-dessous.

- Traduisez ce schéma en relationnel avec la stratégie qui consiste à associer une table par classe de l'arbre d'héritage,
- Même question, avec la stratégie qui consiste à associer une seule table à tout l'arbre d'héritage.



Exercice 5

a) Modéliser le système de fichiers décrit ci-après à l'aide d'un diagramme de classes.

Le système de fichiers est une arborescence de dossiers et de fichiers contenue dans un dossier racine (le 'root directory'). Les dossiers contiennent des (sous-)dossiers et/ou des fichiers. Chaque utilisateur a un dossier à son nom (son 'home directory'). Chaque fichier/dossier a un utilisateur qui le possède ('owner'). Chaque utilisateur peut lire certains fichiers.

b) Traduire ce schéma de classes en relationnel.

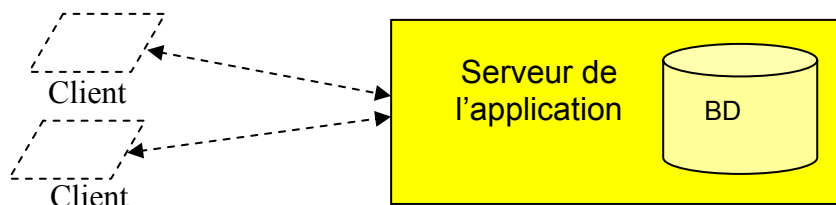
Etude de cas UML

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 connecté (à l'aide d'un nom de login et d'un mot de passe spécifique à l'application et mémorisé sur le serveur), un utilisateur a la possibilité de :

- planifier des réunions virtuelles (choix de son nom, description de son sujet, date de début, durée prévue, description de son ordre du jour, type et éventuellement participants autorisés) 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 (l'animateur quand il existe ou l'organisateur – cf. ci-dessous),
- entrer (virtuellement) dans une réunion précédemment ouverte (seulement les participants autorisés si réunion privée),
- 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 types de réunions peuvent être organisés :

- 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 particulier 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.

L'administrateur du système peut ajouter/supprimer des utilisateurs avec leur nom, login, fonction et mot de passe initial. Les utilisateurs peuvent modifier leur mot de passe.

2. Travail à faire

a) Partie 1

Etablir le diagramme des cas d'utilisation du système. Vous pouvez introduire des relations entre cas (héritage, « extends » et « include ») mais en réfléchissant bien à leur validité.

Conseils

L'héritage entre acteurs signifie que l'acteur qui hérite peut faire tout ce que l'acteur dont il hérite peut faire. Cela simplifie le dessin du diagramme mais n'a aucun impact sur le programme qui sera réalisé.

Les relations « extends » et « include » ne doivent pas servir à décrire des enchaînements d'actions élémentaires (c'est le rôle des diagrammes d'activités). Les cas d'utilisation sont des fonctionnalités complètes du point de vue des utilisateurs qui peuvent (parfois) être liées entre elles.

b) Partie 2

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.

Conseils

L'héritage entre classes est une notion statique (qui impacte fortement le programme réalisé). Quand un objet d'une sous classe est créé il prend ce type particulier et hérite de toutes les propriétés de la super classe. Il conserve ce type et ne peut plus en changer. Si on veut représenter quelque chose de plus dynamique, c'est-à-dire qui évolue dans le temps, le concept d'héritage ne convient pas.

c) Partie 3

Expliciter quelques cas par des diagrammes de séquences : 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. L'objectif est de trouver éventuellement de nouvelles classes et surtout les méthodes utiles à la réalisation des cas.

Conseils

A l'origine de chaque cas on a un acteur qui envoie un message (la partie cliente de l'application n'est pas étudiée et pas représentée). Le message envoyé par l'acteur doit arriver à une classe bien définie qu'il faut introduire dans le diagramme de séquences (quand on écrit l'application cliente il faudra appeler cette classe particulière). Cette classe correspond à une « façade » (cf. le design pattern « façade » en annexe).

d) Partie 4

Donner les diagrammes d'états des classes utilisateur et réunion.

Conseils

On cherchera à représenter les états valables pour tous les utilisateurs et toutes les réunions.

L'objectif est de trouver de nouvelles propriétés (attributs et méthodes) utiles.

e) Partie 5

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.

Conseils

Essayer d'utiliser toutes les possibilités de la notation objet UML (héritage, agrégation, associations, multiplicités...).

Un dossier d'analyse doit 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.

Annexe : le design pattern « façade »

La façade consiste à créer un point d'entrée unique pour accéder à un sous-système. La façade est unidirectionnelle : de l'extérieur (les modules utilisant le sous système) vers l'intérieur. Elle peut se charger de créer certains composants.

