CNAM - CRA Nancy
2003

Jacques Lonchamp

SIXIEME PARTIE

La modélisation objet.
UML

1. Les méthodes d'analyse et de conception

Une méthode :

- propose une *démarche*, distinguant les étapes du développement dans le cycle de vue du logiciel et exploitant au mieux les principes fondamentaux : modularité, réduction de la complexité, réutilisation, abstraction, etc.,
- propose des *formalismes (langages)* et des types de *documents (modèles),* qui facilitent la communication, l'organisation et la vérification,

Il existe de nombreuses méthodes :

- méthodes *fonctionnelles* de décomposition hiérarchique (dites de première génération) comme SADT, SA-SD, ... : application du principe diviser pour régner (problème $\rightarrow$ sous problèmes) ;
- méthodes *systémiques* (dites de deuxième génération) comme MERISE, SSADM, ... : séparation données/traitements, niveaux conceptuels, organisationnels, physiques.
- méthodes *objets* (dites de troisième génération) comme OMT, OOA, OOD, Hood, OOSE, OOM, Fusion, ...

Parmi les principaux objectifs des méthodes objets, on peut noter la volonté de :

- regrouper l'analyse des données et des traitements,
- établir un *couplage explicite entre les concepts du monde réel et les composants exécutables* (« réduire la distance sémantique entre le langage des concepteurs et celui des utilisateurs »),
- faciliter la réutilisation,
- simplifier les transformations entre le niveau conceptuel et l'implantation.

Au début des années 90, il existait une cinquantaine de méthodes objet, liées par un certain consensus autour d'idées communes : objets, classes, sous-systèmes, etc. Mais chacune possédait sa propre notation, ses points forts et ses manques, et son orientation privilégiée vers un domaine d'application. *L'idée d'une certaine unification des méthodes objet* est née de ce constat. La communauté informatique, autour de l'OMG ('Object Management Group') qui standardise les technologies de l'objet, s'est attachée à la définition d'un *langage commun unique*, utilisable par toutes les méthodes objets, dans toutes les phases du cycle de vie et compatible avec les techniques de réalisation du moment.

2. UML

Ce langage commun s'appelle UML ('Unified Modeling Language'). UML est une notation basée principalement sur les méthodes OOD (de Booch), OMT (de Rumbaugh) et OOSE (de Jacobson).

UML a été proposé afin de *standardiser les produits du développement* (modèles, notations, diagrammes) *sans standardiser le processus de développement*. Il est en effet très difficile de standardiser le processus de développement qui dépend des personnes, des applications, des cultures, etc. UML se propose de créer un langage de modélisation utilisable à la fois par les humains (forme graphique) et les machines (syntaxe précise). La Figure 1 résume UML avec ses 5 vues et ses 9 diagrammes.

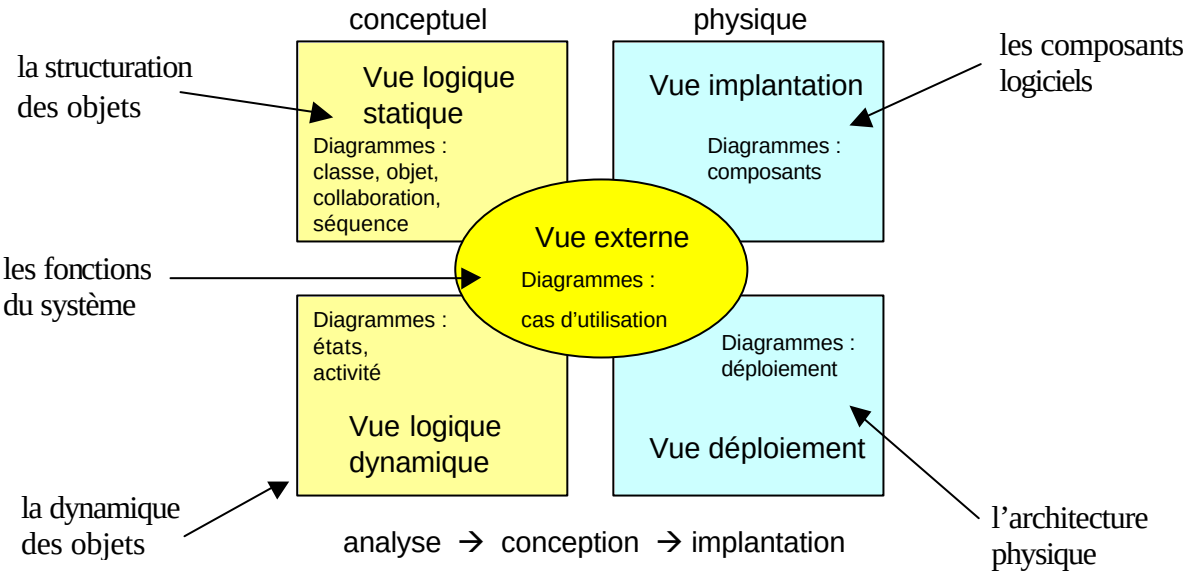


Figure 1 : les vues et les diagrammes de Merise

En résumé :

- UML est une *notation*, pas une méthode.
- UML est un langage de modélisation objet.
- UML convient pour toutes les méthodes objet.
- UML est dans le domaine public.

> *UML a pour but de documenter les modèles objets*.

3. Les cas d'utilisation

Formalisés par Ivar Jacobson dans Object-Oriented Software Engineering (Addison-Wesley, 1992), les cas d'utilisation ('use cases') servent à exprimer le comportement du système en termes d'actions et de réactions, *selon le point de vue de chaque utilisateur* (« approche centrée utilisateur »).

Les cas d'utilisation délimitent le système, ses fonctions (ses cas), et ses relations avec son environnement. Ils constituent un moyen de *déterminer les besoins* du système. Ils permettent *d'impliquer les utilisateurs* dès les premiers stades du développement pour exprimer leurs attentes et leurs besoins (analyse des besoins). Ils constituent un fil conducteur pour le projet et la base pour les tests fonctionnels (cf. Figure 2).

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.

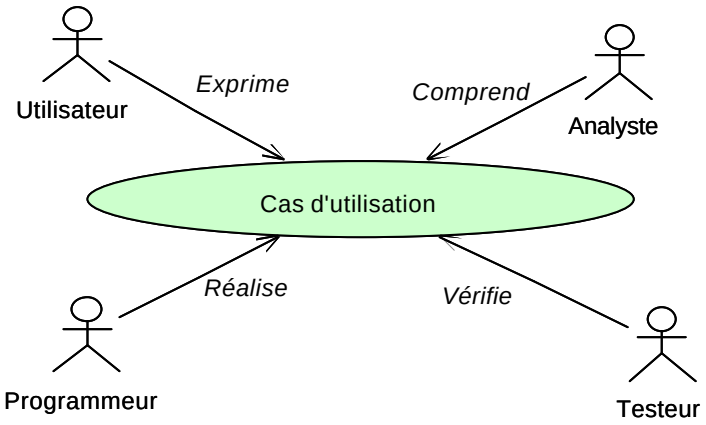


Figure 2 : les cas d'utilisation, fil conducteur du projet

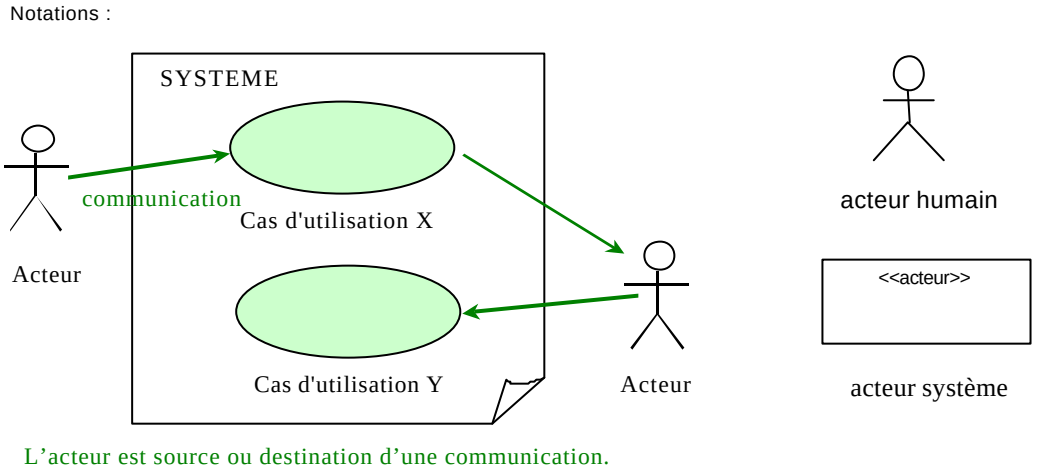


Figure 3 : notations des cas d'utilisation

Les cas d'utilisations représentent le dialogue entre l'acteur et le système de manière abstraite. Les communications sont orientées (avec une flèche) ou non.
Les cas d'utilisations peuvent aussi être vus comme des *classes de scénarios*.
Enfin, les cas peuvent être *structurés* par les relations <<Inclut>> (un cas inclut obligatoirement un autre cas) et <<Etend>> (un cas est une variante d'un autre) (cf. Figure 4). On peut aussi avoir de la généralisation/spécialisation entre acteurs et entre cas (cf. paragraphe 6).

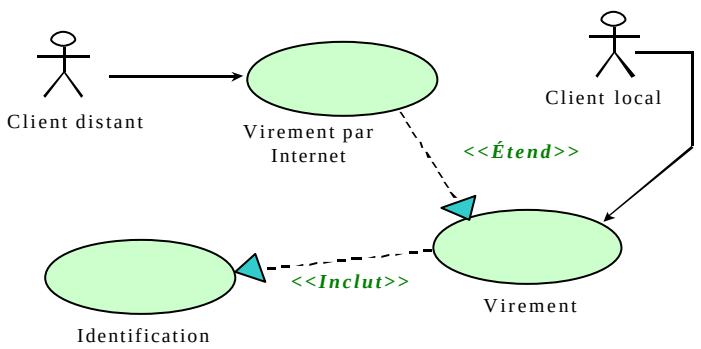


Figure 4 : structuration des cas d'utilisation

Démarche d'utilisation des cas d'utilisation :
(1) Identifier les acteurs et les cas.
(2) Décrire les cas en écrivant des scénarios sous forme textuelle.
(3) Dessiner les diagrammes de cas.
(4) Récapituler les cas, les structurer et s'assurer de la cohérence d'ensemble.

4. Les objets

Les objets informatiques définissent une représentation simplifiée des entités du monde réel, qu'il s'agisse d'entités concrètes (avec une «réalité physique») ou conceptuelles. Les objets sont des *abstractions* qui mettent en avant les caractéristiques essentielles et dissimulent les détails. Une abstraction se définit par rapport à un *point de vue* (tout dépend de l'intérêt que l'on porte à l'objet).
Exemples d'objets : une transaction bancaire, un client, un pop-up menu.

Les trois caractéristiques fondamentales des objets sont *l'état, le comportement, l'identité. L'état* correspond aux *valeurs instantanées de tous les attributs* (ou données membres) de l'objet. L'état évolue au cours du temps. L'état d'un objet à un instant donné est la conséquence de ses comportements passés.
Exemples : pour un signal électrique : valeurs de l'amplitude, de la pulsation, de la phase, ... ; pour une voiture : valeurs de la marque, de la puissance, de la couleur, du nombre de places assises, de la quantité d'essence, ...

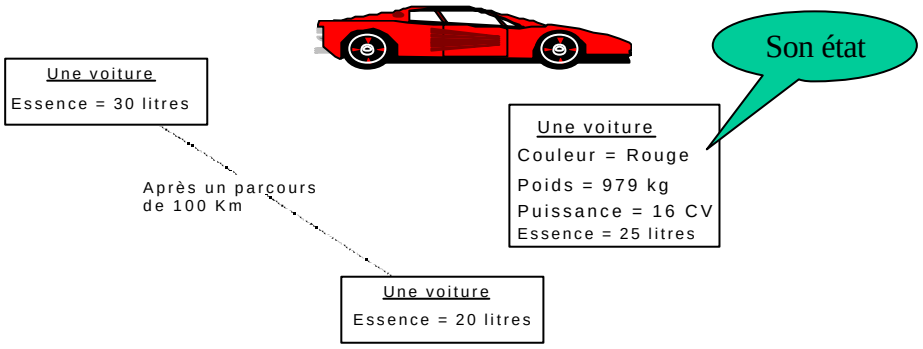


Figure 5 : objet et état

Le comportement décrit les actions et les réactions d'un objet (ou 'compétences' d'un objet). Le comportement se matérialise sous la forme d'un ensemble d'*opérations* (ou méthodes ou fonctions membres). On distingue souvent les opérations selon leur finalité de constructeur, accesseur, modificateur, destructeur et itérateur.

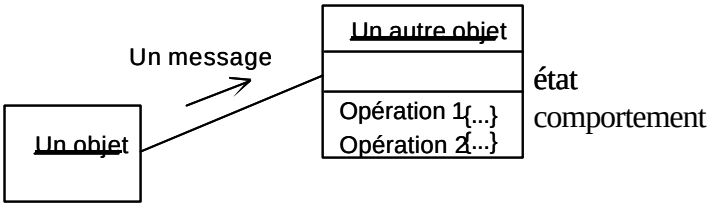


Figure 6 : comportement d'un objet

Un objet peut faire appel aux compétences d'un autre objet par l'invocation d'une opération (ou *'envoi de message*'). Les messages servent à implanter les flots de données, les flots de contrôle, les événements , etc.

L'état et le comportement sont liés :
– le comportement dépend de l'état (cf. la pré condition de l'opération); le message peut être accepté ou refusé;
– l'état est modifié par le comportement (cf. la post condition de l'opération).

Tout objet possède une *identité* interne (non modifiable) qui lui est propre et qui le caractérise. L'identité permet de distinguer tout objet de façon non ambiguë, *indépendamment de son état* (de ses attributs). Les langages objets utilisent des *identifiants internes*. Les identifiants et clés primaires des bases de données sont inutiles en objet.

Intérêt des objets :

- Concurrence : les objets sont autonomes et évoluent indépendamment les uns des autres.
- Maîtrise de la complexité :
    - l'encapsulation permet de se concentrer sur un objet et un seul,
    - il est possible de tester de façon quasi exhaustive un objet.
- Evolution facilitée : rajouter de nouveaux objets est simple.
- Réutilisation possible.

Difficulté des objets :

- Tout n'est pas naturellement objet.
- Tester un système OO est difficile.
- Penser objet oblige à changer de mentalité !

5. Les collaborations

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).

5.1. Les diagrammes de collaboration

Les diagrammes de collaboration mettent l'accent sur les relations « spatiales » entre objets. Les messages peuvent être numérotés pour introduire une dimension temporelle. De nombreuses notations annexes permettent de caractériser les messages. Ils sont souvent utilisés pour décrire grossièrement la réalisation des cas d'utilisation.
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.

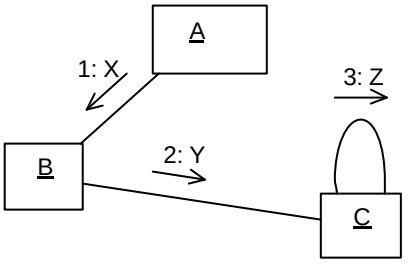


Figure 7 : un diagramme de collaboration

5.2. Les diagrammes de séquence

Ils mettent l'accent sur les relations temporelles. 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. Ils peuvent être utilisés aussi bien pour préciser la réalisation des cas d'utilisation que pour spécifier de manière très détaillée la dynamique d'un ensemble d'objets (appels de méthodes).

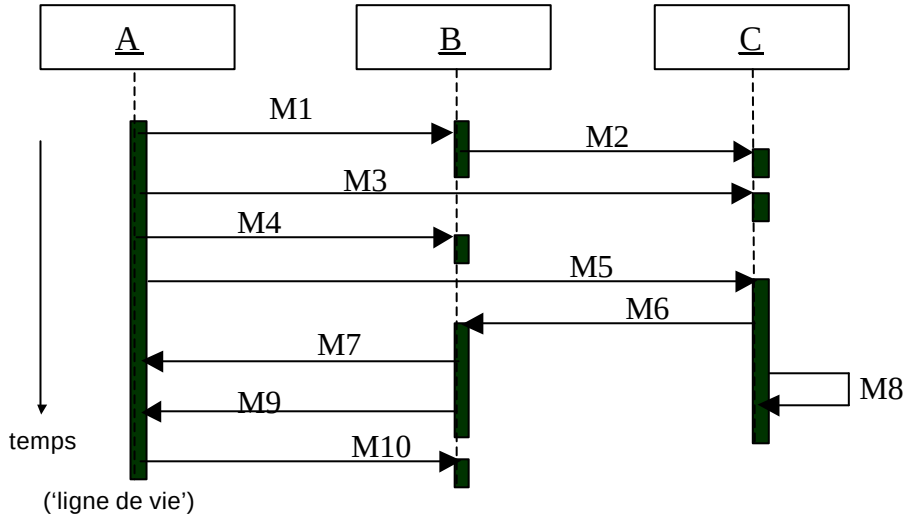


Figure 8 : un diagramme de séquence

6. Les classes

Le monde qui nous entoure est constitué de très nombreux objets. Pour comprendre le monde, l'être humain a tendance à regrouper les éléments qui se ressemblent. Regrouper des objets suivants des critères de ressemblance s'appelle *classer*. Les humains ont classé les animaux les plantes, les champignons, les atomes, ...

La classe est la *description abstraite d'un ensemble d'objets*. Elle peut être vue comme la factorisation des éléments communs à un ensemble d'objets.

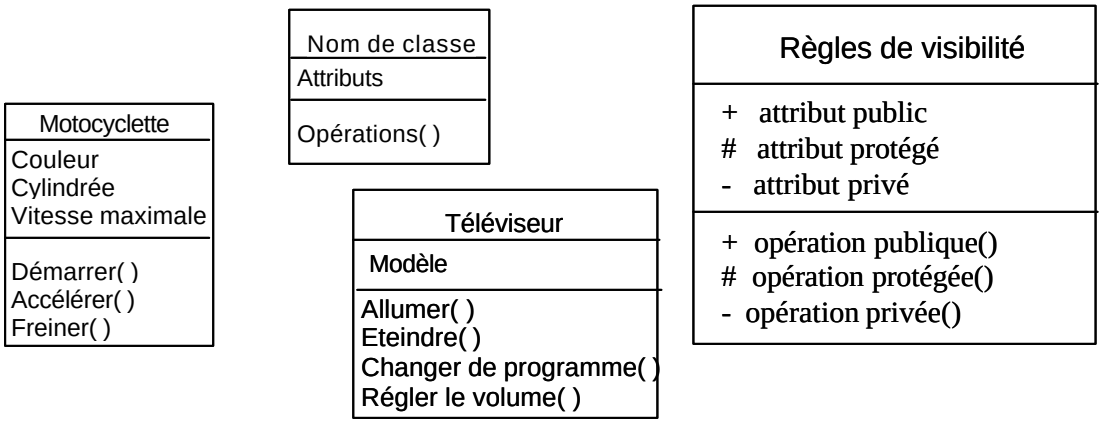


Figure 9 : des exemples de classe

La classe intègre les concepts de *type* (en tant que «moule à instances ») et de *module* (avec l'idée d'interface + corps).

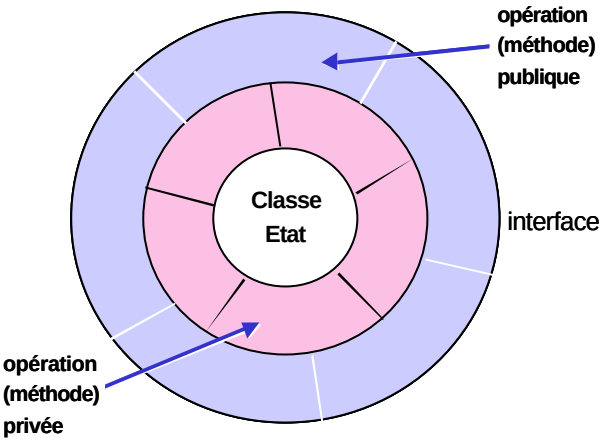


Figure 10 : interface et corps d'une classe

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 et la généricité. 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.

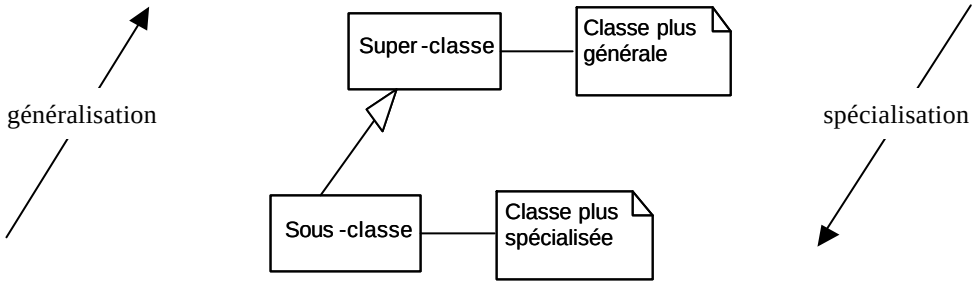


Figure 11 : généralisation/spécialisation

Exemple :

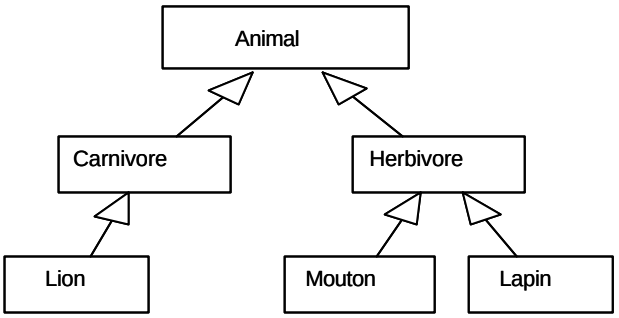


Figure 12 : un exemple de hiérarchisation.

Les instances 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.
On peut dire aussi que le type Lion est un *sous type* du type Carnivore. Tout lion est un carnivore (on parle aussi de relation 'est-un' ou 'isa' en anglais). L'ensemble des lions est un sous ensemble des carnivores. Mais l'ensemble des attributs d'un lion est un sur ensemble de l'ensemble des attributs d'un carnivore ! (d'où le mot *extends* qu'utilise Java pour lier sous-classe et super-classe).

La relation de spécialisation/généralisation est une relation non-réflexive, non-symétrique et transitive. L'héritage multiple (plusieurs super classes) est possible.

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.
Exemple : on paye des employés de 2 types ('mensualisés' et 'à la tache'). Il suffit d'envoyer la méthode calculerPaie() à tous les employés. Si un nouveau type d'employé apparaît le programme de paie n'a pas à être modifié.

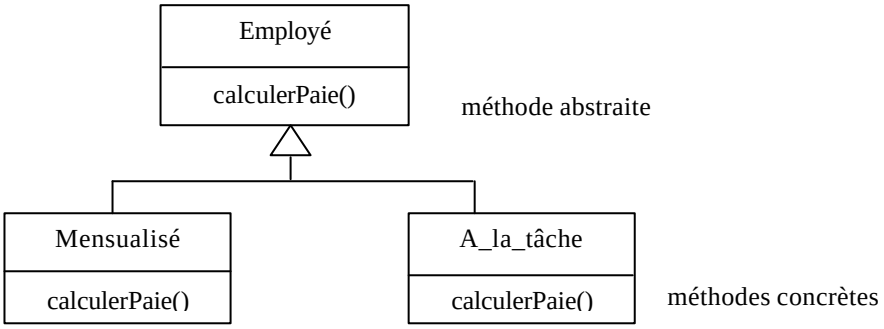


Figure 13 : polymorphisme

7. Les diagrammes de classes

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). 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 multiplicité) indiquent le nombre d'instances d'une classe pour chaque instance de l'autre classe.

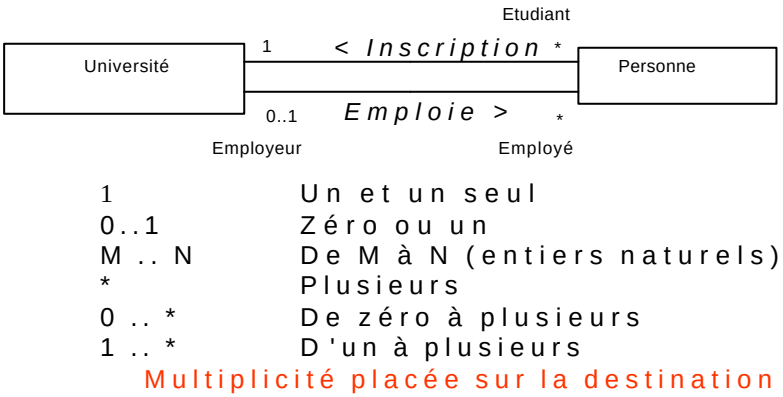


Figure 14 : les associations et leurs caractéristiques

Exemple : association Inscription entre Etudiant et Université (cf. Figure 14); lien d'inscription entre l'objet Pierre et l'objet Université de Nancy. Rôles Employeur et Employé de l'association Emploie.

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.

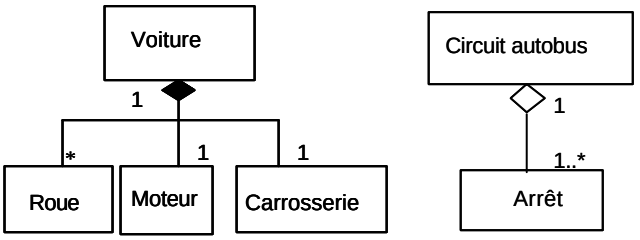
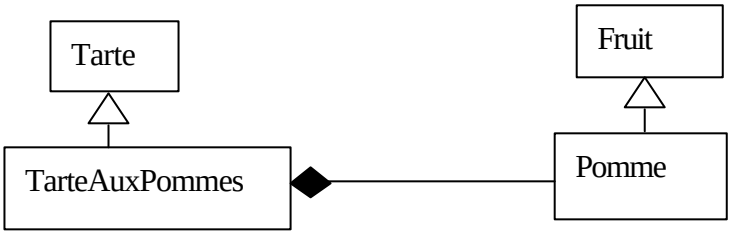


Figure 15 : Composition et autre forme d'agrégation

Attention à 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.
Exemple :



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

Autres concepts :

- Attributs d'associations :

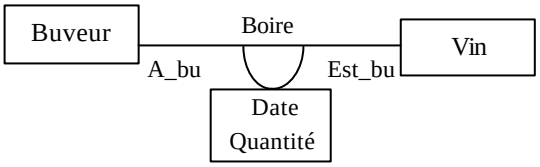


Figure 16 : attributs d'association

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

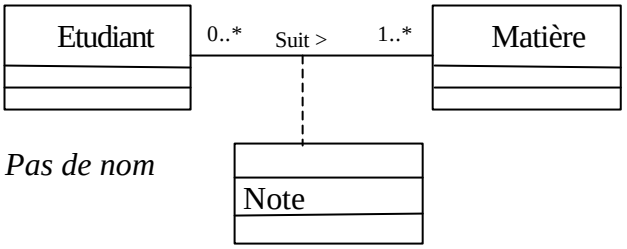


Figure 17 : classe –association

- *Association d'arité n :* représentée par un losange avec n 'pattes' auquel peut être associé une classe porteuse d'attributs et d'opérations.

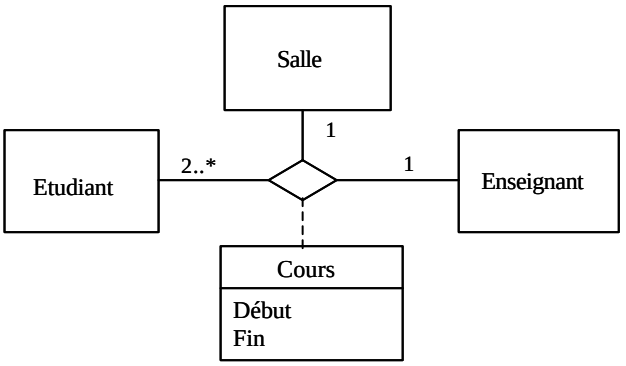


Figure 18 : association n-aire

- Paquetage (sous modèle) : c'est une notion qui peut apparaître dans beaucoup de diagrammes pour spécifier le regroupement d'éléments au sein d'un sous système (cas, classes, objets, composants, autres paquetages, ...).

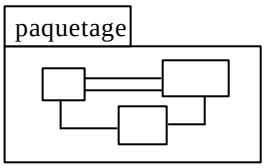


Figure 19 : un paquetage

- Contraintes d'intégrité entre associations (inclusion, exclusion, ...).
- Classes abstraites : elles sont non instanciables directement ; elles décrivent des mécanismes généraux et laissent non décrits certains aspects (méthodes abstraites) ; elles sont spécialisées par des classes concrètes (instanciables) qui précisent les méthodes abstraites.

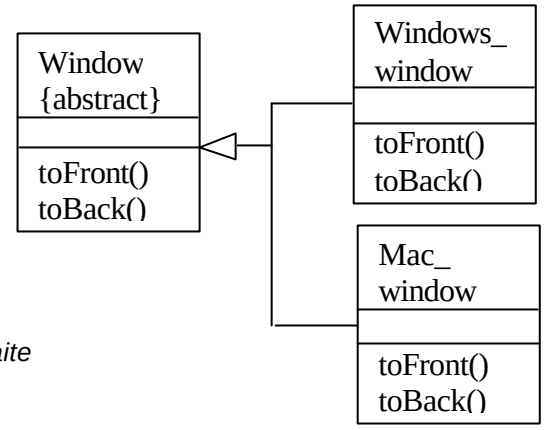


Figure 20 : classe abstraite

- Interfaces : ce sont des classes ne contenant que des opérations abstraites ; elles précisent les fonctionnalités que les classe qui les implantent doivent fournir.

7. Les diagrammes d'état

Les diagrammes d'état servent a décrire le comportement des objets (classes) complexes. Ce sont des diagrammes d'état étendus avec des *conditions* (gardes) sur les transitions et des *actions* sur les transitions et les états (cf. Figure 21).

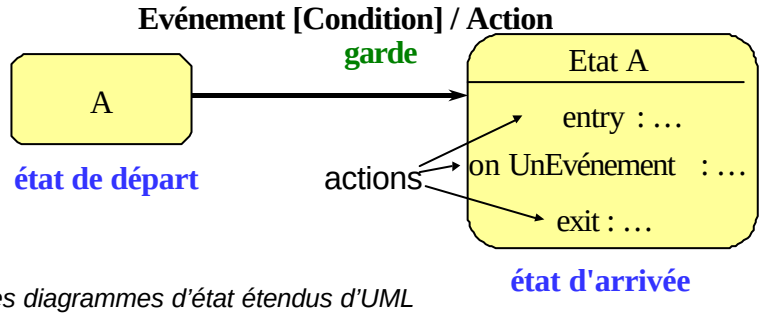


Figure 21 : les diagrammes d'état étendus d'UML

Ces diagrammes permettent aussi la *décomposition d'états en sous états* (cf. Figure 22). On parle aussi *d'état composite*.

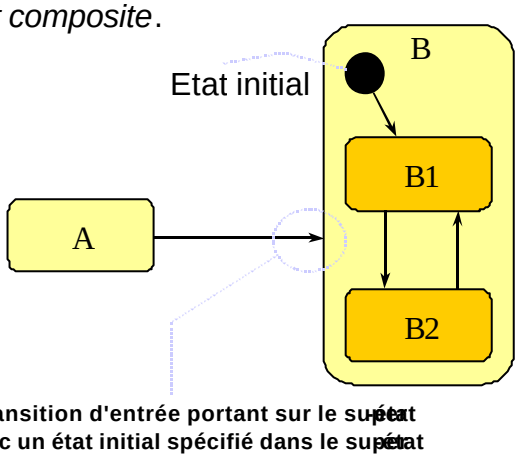


Figure 22 : décomposition d'états

Enfin, il est possible de décrire du *parallélisme entre sous systèmes,* ce qui rapproche les diagrammes d'état des réseaux de Petri (cf. Figure 23).

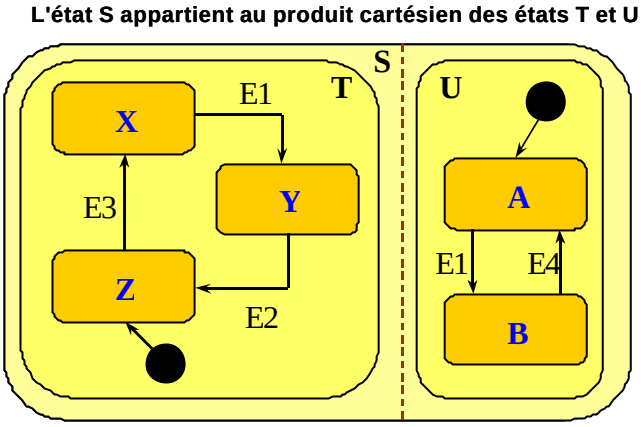


Figure 23 : du parallélisme au sein d'un état

8. Les diagrammes d'activité

Ils permettent de décrire un *flot de contrôle* entre opérations (calculs). Il s'agit en gros d'*organigrammes incluant éventuellement du parallélisme* (cf. Figure 24).

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.

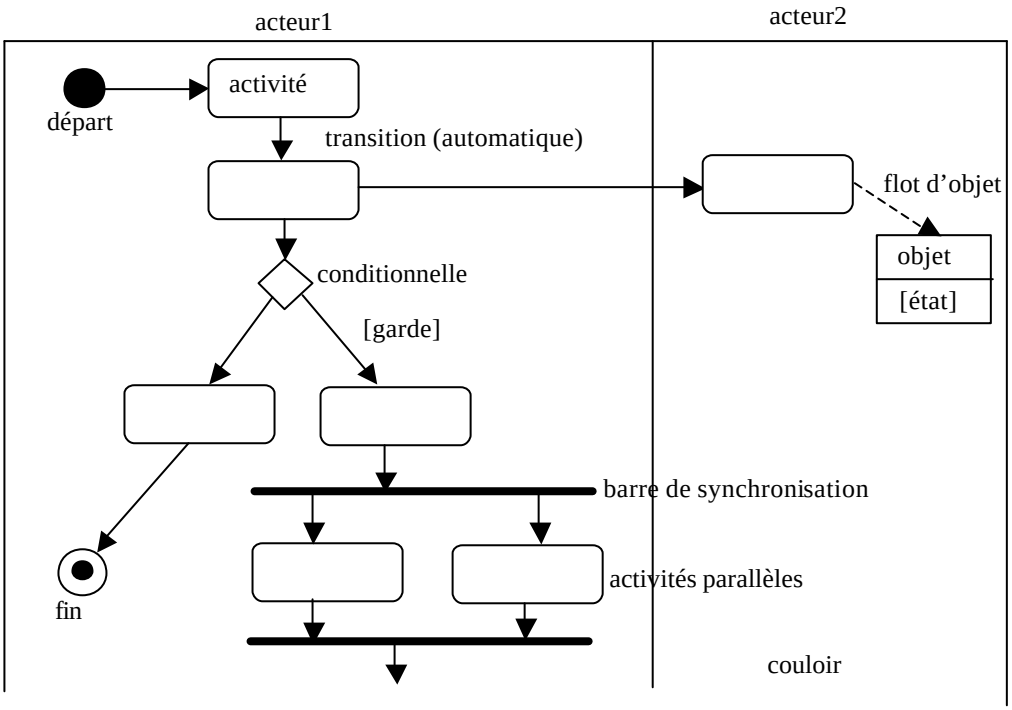


Figure 24 : notations des diagrammes d'activité

9. Autres éléments

9.1. Les diagrammes de composants

Les composants sont des codes (sources, exécutables), des scripts, des fichiers, des tables, etc. (cf. Figure 25).

9.2. Les diagrammes de déploiement

Ces diagrammes illustrent (cf. Figure 25) :

- la disposition physique des différents matériels (ou noeuds) qui entrent dans la composition du système,
- la répartition des composants (cf. diagrammes de composants) au sein des noeuds,
- les supports de communication entre noeuds.

9.3. Le langage OCL

UML offre un langage formel pour *l'expression des contraintes* (Object Constraint Language).

C'est un langage déclaratif typé. Il peut servir à spécifier les invariants de classe, les pré et post conditions des opérations, les gardes, etc.

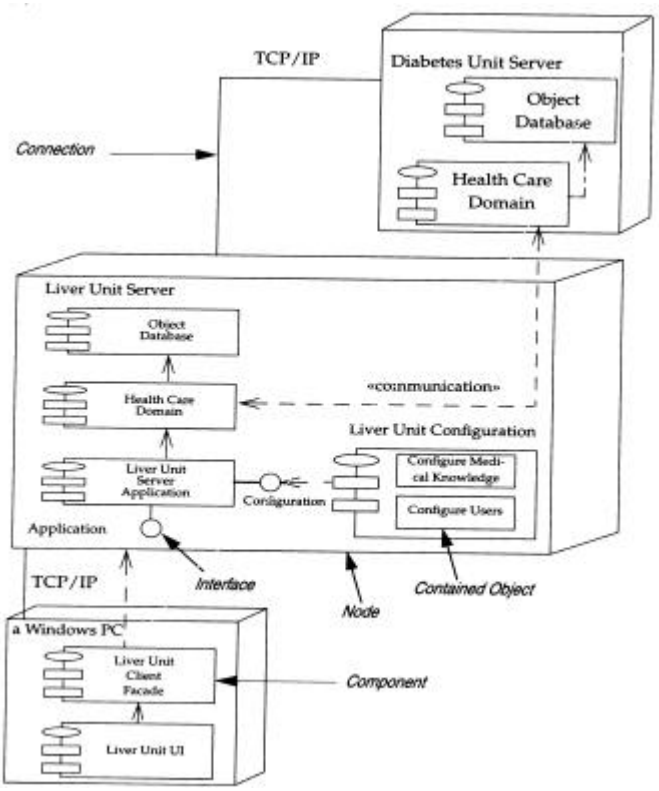


Figure 25 : un diagramme de déploiement

10. Eléments d'une démarche

UML n'impose pas de processus. La 'démarche naturelle', impliquée par la notation UML, *part des cas d'utilisation* qui expriment un point de vue fonctionnel sur le système.

Puis les *diagrammes de collaboration et de séquence associés aux cas* font apparaître les classes d'objets impliquées dans le système et donc passer à une vue 'objets' (cf. figure 26).

Mais on peut aussi bien passer à une vue *'hiérarchie de fonctions'* à partir des cas d'utilisation, comme le montre la Figure 27.

Diagrammes essentiels par phase :

- Analyse des besoins : cas d'utilisation.
- Analyse du système : diagrammes de classes, de collaboration, d'activités (enchaînement des cas).
- Conception : diagrammes de classes, de séquences, d'activités (conception des méthodes), d'états, de composants, de déploiement.

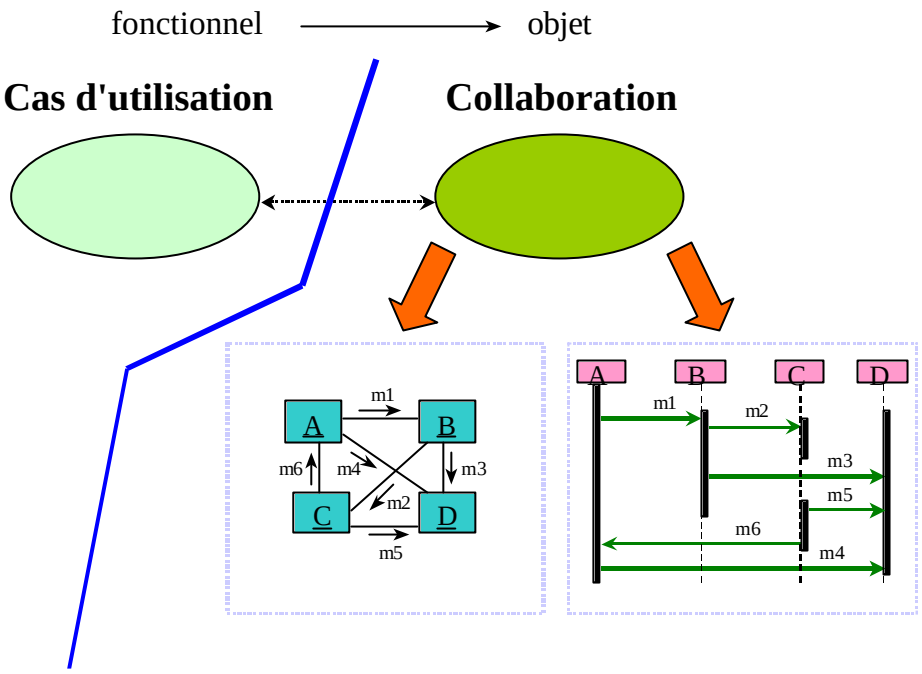


Figure 26 : des cas d'utilisation aux classes

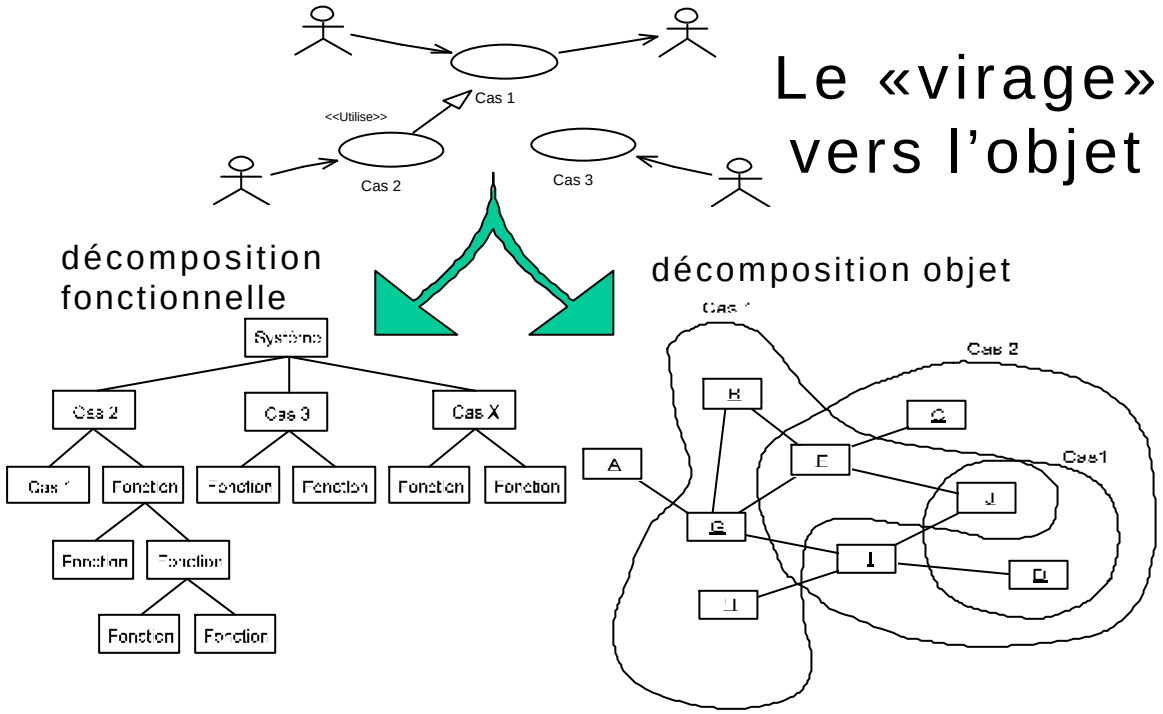


Figure 27 : des cas d'utilisation aux fonctions ou aux objets

EXERCICES

Exercice 6.1 *: Le GAB - cas d'utilisation*
Modélisation d'un GAB (Guichet Automatique de Banque). Les principales fonctions sont les suivantes :

- distribution d'argent à tout porteur d'une carte de la banque (autorisation d'un certain montant par le Système d'Information de la banque) ou d'une carte VISA (autorisation à distance par le Système d'Autorisation VISA),
- consultation du solde, dépôt en numéraire et de chèques pour les possesseurs d'une carte de la banque.

Toutes les transactions sont sécurisées (code personnel vérifié avec le code enregistré sur la puce de la carte ; la carte est avalée après trois échecs).
Il faut parfois recharger le GAB et retirer des choses ...
Identifier les acteurs et les cas. Structurer les cas.

Exercice 6.2 : *Le GAB - diagramme d'activités et diagramme de séquence*
a) Modéliser le retrait d'argent avec une carte VISA avec un diagramme d'activités. La carte peut être invalide. Si elle est valide, le client doit taper son code. La carte est avalée après trois essais infructueux. Le SA VISA autorise un certain montant ou refuse tout retrait. Une carte non récupérée est avalée. Les billets non récupérés par le client sont repris. Un ticket est toujours imprimé pendant que les billets sont proposés.

b) Modéliser le scénario nominal (succès) avec un diagramme de séquence.

Exercice 6.3 : *Le cirque – diagramme de classes*
Le propriétaire d'un cirque souhaite informatiser une partie de la gestion de ses spectacles. Proposer un modèle conceptuel UML (diagramme de classes) 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).

Exercice 6.4 *: L'institut de formation – diagramme de classes*

Il s'agit d'établir le schéma des données pour la gestion des formations d'un institut privé. Un cours est caractérisé par un numéro de cours (NOCOURS), un libellé (LIBELLE), une durée en heures (DUREE) et un type (TYPE). Un cours peut faire l'objet dans l'année de plusieurs sessions identiques. Une session est caractérisée par un numéro (NOSES), une date de début (DATE) et un prix (PRIX). Une session est le plus souvent assurée par plusieurs animateurs et est placée sous la responsabilité d'un animateur principal. Un animateur peut intervenir dans plusieurs sessions au cours de l'année. On désire mémoriser le nombre d'heures (NBH) effectué par un animateur pour chaque session.

Un animateur est caractérisé par un numéro (NOANI), un nom (NOMA) et une adresse (ADRA).

Chaque session est suivie par un certain nombre de participants. Un participant est une personne indépendante ou un employé d'une entreprise cliente. Un participant est caractérisé par un numéro (NO-PAR), un nom (NOMP) et une adresse (ADRP). Dans le cas d'un employé, on enregistre le nom (NO-MEN) et l'adresse de l'entreprise (ADREN). On désire pouvoir gérer d'une manière séparée (pour la facturation notamment) les personnes indépendantes d'une part, et les employés d'autre part.

Exercice 6.5 *Gestion de parc informatique – diagramme de classes*

Une entreprise souhaite informatiser la gestion de son parc informatique (ordinateurs, imprimantes, etc.) pour en optimiser la maintenance.

Proposer un schéma de classes UML modélisant les spécifications ci-dessous (classes, associations entre classes, cardinalités des associations, attributs des classes).

Un ordinateur est caractérisé par son numéro d'inventaire, son adresse réseau (adresse IP), son modèle, la date de son acquisition, la date de la prochaine maintenance planifiée et le système d'exploitation installé.

Sur chaque ordinateur est installé un ensemble de logiciels caractérisés par un numéro de licence, un nom et une version.

Grâce à un système de mots de passe, chaque ordinateur peut être utilisé par plusieurs employés mais, pour des raisons de sécurité des données, un employé n'a le droit d'utiliser qu'un seul ordinateur. Un employé est caractérisé par son nom, son prénom et sa fonction dans l'entreprise.

Les ordinateurs sont reliés à un certain nombre de périphériques en réseau (imprimantes, scanners, etc.). Chaque périphérique est caractérisé par un numéro d'inventaire, son adresse IP, son type, son modèle, sa date d'acquisition et la date de la prochaine maintenance planifiée. Les périphériques pouvant servir à plusieurs ordinateurs simultanément, un indice de priorité est affecté à chaque ordinateur pour chaque périphérique auquel il est connecté.

Chaque ordinateur et chaque périphérique est localisé dans un bureau donné. Les bureaux sont caractérisés par un numéro de bureau et le numéro du bâtiment dans lequel ils se trouvent. Un numéro de bureau est unique dans un bâtiment donné.

Exercice 6.6 *Cas d'utilisation, diagramme de classes, diagramme de séquence*

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 le responsable formation reçoit une demande de formation d'un employé. Cet employé peut éventuellement consulter le

catalogue des formations offertes par les organismes agréés par l'entreprise. Cette demande est instruite par le responsable qui transmet son accord ou son refus à l'employé.

En cas d'accord, le responsable cherche la formation adéquate dans le catalogues des formations agréées qu'il tient à jour. Il informe l'employé du contenu de la formation et lui soumet le 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 avertir au plus vite le responsable formation pour que celui-ci demande l'annulation de l'inscription.

A la fin de la formation l'employé transmet une appréciation sur le stage suivi et un document attestant sa présence.

Le responsable formation contrôle la facture envoyée par l'organisme de formation.

a) Dessiner le diagramme des cas d'utilisation.

b) Dessiner le schéma des 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.

c) Dessiner le diagramme de séquence associé à la demande initiale de l'employé décrite dans le deuxième paragraphe de l'énoncé ; assurer la cohérence avec votre réponse à la question précédente.