Introduction à UML et à la modélisation objet

Sommaire

- 1. Introduction
  - Présentation
  - Les diagrammes d'UML
  - Notions d'objet et de classes
- 2. Vue structurelle
  - Diagramme de classes
  - Diagramme d'objets
- 3. Vue fonctionnelle
  - Diagramme de cas d'utilisation
  - Diagramme de collaboration
  - Diagramme de séquence

1. Introduction

1.1. Présentation

Définitions

UML = **U**nified **M**odeling **L**anguage
Langage unifié pour la modélisation objet

Langage de modélisation des applications construites à l'aide d'**objets**, indépendant de la **méthode** utilisée

Langage de modélisation : notations

Méthode : utilisation du langage de modélisation

(recueil des besoins, analyse, conception, mise en œuvre, validation…)

Objet : représentation du problème basée sur des entités (concrètes ou abstraites) du monde réel

La complexité des logiciels

- Les systèmes peuvent être décomposés selon
  - ce qu'ils font (approche fonctionnelle)
  - ce qu'ils sont (approche objet)

- L'approche objet gère plus efficacement la complexité
  - Modèles basés sur le monde réel → **stabilité**
  - Structure indépendante des fonctions → **évolutivité**
  - Approche modulaire → **maintenance**, **réutilisabilité**

Historique des langages OO

- Langages de programmation orientés objets
  - Simula (1967)
  - Smalltalk (1970)
  - C plus Classes (1980)
  - C++ (1985)
  - Eiffel (1988)
  - Java (1995)
- SGBD orientés objets

  Utilisation des bds avec un langage OO

- Genèse des méthodes d'analyse :

  Implémentation

  → Conception (solution informatique)

  → Analyse (comprendre et modéliser le problème) → …

Les méthodes d'analyse

- Méthodes orientées comportement :

  on s'intéresse à la dynamique du système ; *ex : réseaux de Pétri*

- Méthodes fonctionnelles :

  s'inspirent de l'architecture des ordinateurs

  on s'intéresse aux fonctions du système ; *ex : SADT*

- Méthodes orientées données :

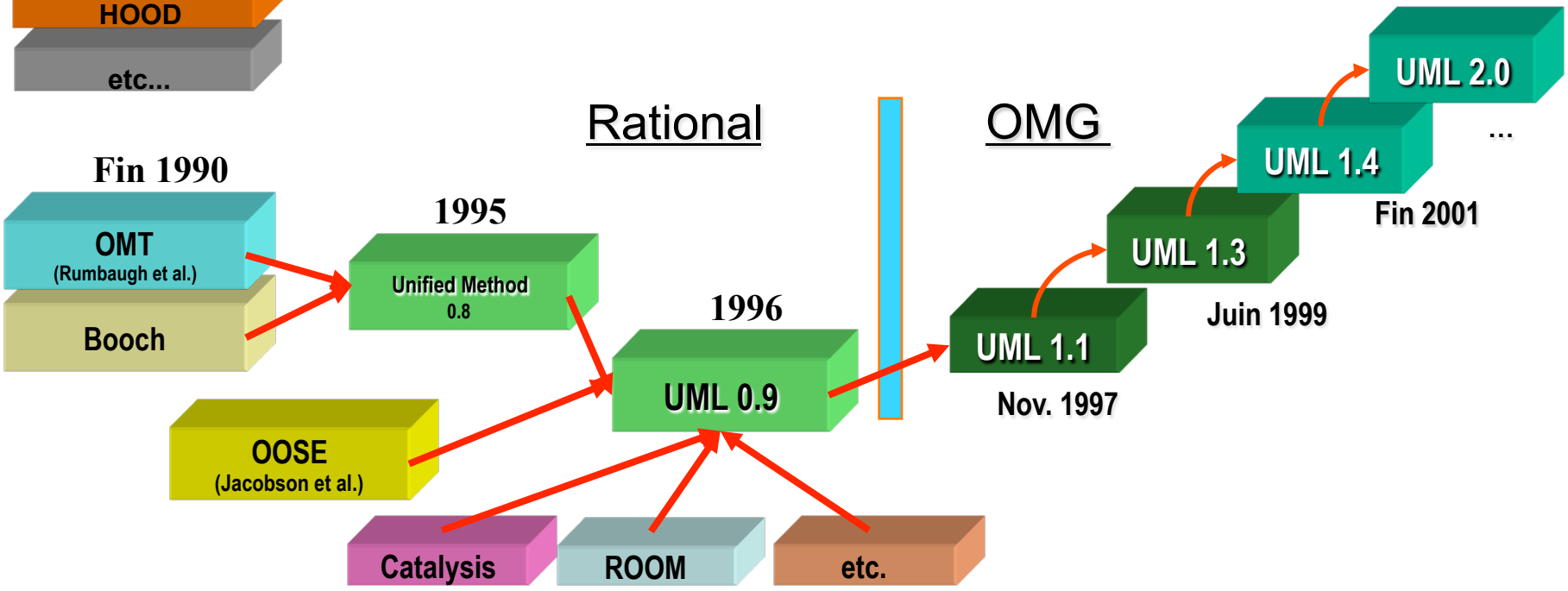
  on ne s'intéresse pas aux traitements ; *ex : Merise*

- Méthodes orientées objets :

  on ne sépare pas les données et les traitements ; *ex : Booch, OMT*

Genèse d'UML

- Utilisation d'un standard de modélisation « universel »
- Au départ, plus de 150 méthodes !!
- Unification progressive de plusieurs méthodes, de remarques des utilisateurs, des partenaires
- 1989 : création de l'**OMG** (Object Management group) ; groupe créé à l'initiative de grandes sociétés informatiques américaines afin de normaliser les systèmes à objets ; 1ère réalisation de l'OMG : CORBA (communication entre applications objets dans un système distribué hétérogène)



L'unification

- des méthodes
  - La guerre des méthodes ne fait plus avancer la technologie des objets
  - Recherche d'un langage commun unique
    - Utilisable par toutes les méthodes
    - Adapté à toutes les phases du développement
    - Compatible avec toutes les techniques de réalisation

- sur plusieurs domaines d'applications
  - Scientifique
  - Industriel
  - Gestion
  - Multimédia
  - …

Portée

- Reste au niveau d'un langage
  ne propose pas un processus de développement
  - ni ordonnancement des tâches,
  - ni répartition des responsabilités,
  - ni règles de mise en œuvre

- Certains ouvrages et AGL basés sur UML ajoutent cet aspect fondamental en méthodologie

Démarche

- Synthèse des points de vue
  - statique
  - fonctionnel
  - dynamique

- Démarche incrémentale

  de l'analyse à la conception par raffinages successifs

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 est **la notation standard** pour documenter les modèles objets

Adresses web

- Sites officiels
  - OMG : www.omg.org
  - UML : www.uml.org

- Ateliers de Génie Logiciel (AGL) utilisant UML
  - **Objecteering** ~~de Softeam~~ : www.objecteering.com
  - **Rational** : www.rational.com → IBM
  - Poseidon : www.gentleware.com
  - Plugins Eclipse : www.eclipse.org
    - EclipseUML : www.omondo.com → eUML2 : www.soyatec.com
    - UMLLab: www.uml-lab.com
  - Modelio (spinoff d'Eclipse):
    - www.modelio.[com|org]
- Liens francophones
  - Pierre Alain Muller : http://www.irisa.fr/triskell/members/pierre-alain.muller/teaching (2006)
  - Site francophone : uml.free.fr

1.2. Les diagrammes d'UML

Ressemblances avec Merise II

- MEA (Modèle Entité-Association) *vs* Diagramme de classes
  - entité ~ classes (attributs + méthodes)
  - Association, cardinalités
  - Généralisation-spécialisation ~ héritage
- CVO (Cycle de Vie d'un Objet) vs. Diagramme état-transition
  - Notion d'état
  - Transition entre états
- MC, MFC, MCTA (modèles fonctionnels)
  - Découpage intermédiaires des MFC ~ Hiérarchies de diagrammes de cas d'utilisation
  - MCTA ~ Diagramme d'activité

Différences avec Merise II

- Types d'application
  - Merise II : uniquement orientée vers des applications de gestion
  - UML : tout type d'applications
- Persistance des données
  - MEA : objets persistants, destinés à être stockés dans des fichiers
  - Diagramme de classes : tous les objets, persistants ou non, y compris le programme principal
- Lien entre données et traitements
  - MEA : que des attributs
  - UML : classes = tous les attributs (et paramètres et calculés) + méthodes associées
- Attributs UML
  - Peuvent être multivalués (tableaux, liste…)

Les 9 diagrammes

- Besoins des utilisateurs
  - ***Diagramme des cas d'utilisation*** ③
- Vue structurelle : aspects statiques
  - ***Diagramme de classes*** ①
  - ***Diagramme objet*** ②
- Vue fonctionnelle : interactions entre objets
  - ***Diagramme de collaboration*** ④
  - ***Diagramme de séquence*** ⑤
- Vue dynamique : dynamique des objets
  - Diagramme états-transition
  - Diagramme d'activités (enchaînement et synchronisation des activités pour une classe ≈ MCTA de Merise découpé par classes)
- Réalisation et déploiement
  - ***Diagramme de composants*** (unités physiques de code source et de code exécutable)
  - ***Diagrammes de déploiement*** (façon dont l'application sera distribuée sur le réseau)

UNIFIED MODELING LANGUAGE

1.3. Notions d'objet et de classe

Les objets

- Les objets du monde réel nous entourent, ils naissent, vivent et meurent.
- Les objets informatiques définissent une représentation simplifiée des entités du monde réel.
- Les objets représentent des entités
  - concrètes : avec une masse
  - abstraites : concept

Les objets sont des abstractions

- Une abstraction est un résumé, un condensé
- Mise en avant des caractéristiques essentielles
- Dissimulation des détails
- Une abstraction se définit par rapport à un point de vue

- Exemples d'abstractions
  - Une carte routière
  - Un nombre complexe
  - Un téléviseur
  - Une transaction bancaire
  - Une porte logique
  - Une pile
  - Un étudiant

Caractéristiques fondamentales des objets

- Objet = État + Comportement + Identité
- Communication entre objets

L'état

- L'état d'un objet :
  - regroupe les valeurs instantanées de tous les attributs d'un objet
  - évolue au cours du temps
  - à un instant donné est la conséquence de ses comportements passés

- Exemples
  - Pour un signal électrique : l'amplitude, la pulsation, la phase, …
  - Pour une voiture : la marque, la puissance, la couleur, le nombre de places assises, …
  - Pour un étudiant : le nom, le prénom, la date de naissance, l'adresse, …

Le comportement

- Le comportement
  - décrit les actions et les réactions d'un objet
  - regroupe toutes les compétences d'un objet
  - se représente sous la forme d'opérations (méthodes)

- Un objet peut faire appel aux compétences d'un autre objet

- L'état et le comportement sont liés
  - Le comportement dépend de l'état
  - L'état est modifié par le comportement

L'identité

- Tout objet possède une identité qui lui est propre et qui le caractérise
- L'identité permet de distinguer tout objet de façon non ambigüe, indépendamment de l'état
- Les langages objets utilisent généralement des pointeurs pour réaliser un identifiant. Un attribut identifiant n'est pas nécessaire.
- Il est possible de définir une contrainte d'unicité
- Une clé primaire dans une base de donnée relationnelle est une manière de réaliser l'identité (en l'insérant dans l'état)

Communication entre objets

- Application = société d'objets collaborant
- Les objets travaillent en synergie 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

- Les objets
  - ne vivent pas en ermites
  - Les objets interagissent les uns avec les autres
  - Les objets communiquent en échangeant des messages

Communication (suite)

- Catégories de messages :
  - **Constructeurs** (constructors) : créent des objets
  - **Destructeurs** (destructors) : détruisent des objets
  - **Accesseurs** (accessors) : renvoient tout ou partie de l'état
  - **Modifieurs** (mutators) : changent tout ou partie de l'état
  - **Itérateurs** (iterators) : traversent une collection d'objets

Les classes

- La classe
  - est une description abstraite d'un ensemble d'objets
  - peut être vue comme la factorisation des éléments communs à un ensemble d'objets
  - décrit le domaine de définition d'un ensemble d'objets

- Description des classes
  - Séparée en deux parties
    - La **spécification** d'une classe qui décrit le domaine de définition et les propriétés des instances de cette classe (type de donnée)
    - La **réalisation** qui décrit comment la spécification est réalisée

Conclusion

- Les objets naissent, vivent et meurent
- Les objets interagissent entre eux
- Les objets sont regroupés dans des classes qui les décrivent de manière abstraite
- La classe intègre les concepts de type et de module

UNIFIED MODELING LANGUAGE

2. Vue structurelle

2.1. Diagramme de classes

Définitions

- Structure statique d'un système
  - Classes (ensembles d'objets)
  - Relations entre classes (ensembles de liens entre objets)

- Opérations / méthodes
  - Opération : service qui peut être demandé à n'importe quel objet de la classe.
  - Méthode : implémentation d'une opération.
  - Chaque opération non abstraite d'une classe doit avoir une méthode qui fournit un algorithme exécutable comme corps (cet algorithme est donné dans un langage de programmation ou dans du texte structuré).

Niveau de détail d'analyse

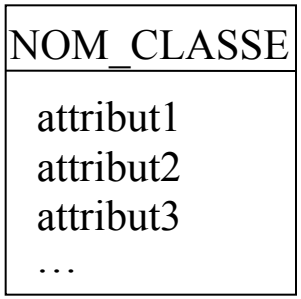
Spécification : plusieurs niveaux de détails :

- Niveau de détail d'**analyse**
  Pas de précision sur la mise en œuvre
  Indépendant du logiciel
  Plusieurs niveaux de précision au fil des itérations d'analyse

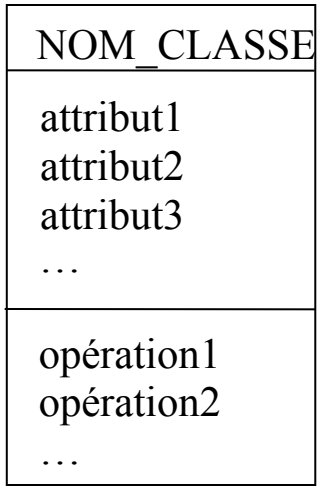
  - simplifié
    - Uniquement le nom de la classe

  - intermédiaire
    - Nom de la classe
    - Nom des attributs, ou des opérations

  - complet
    - Nom de la classe
    - Noms des attributs
    - Noms des méthodes

Niveau de détail de conception

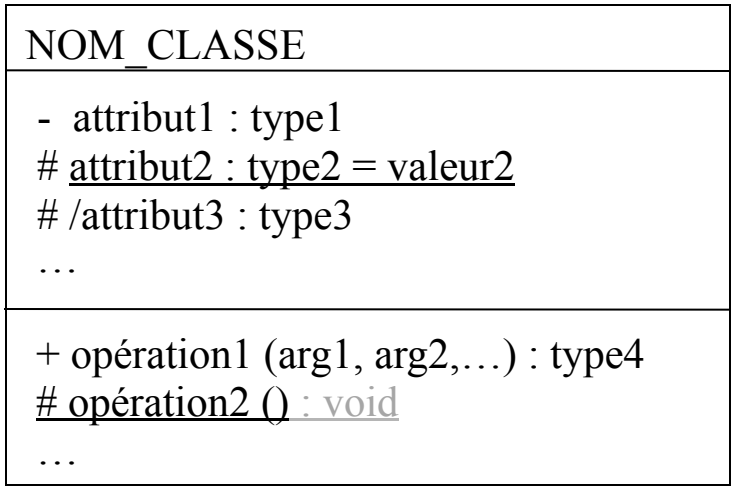
- Niveau de détail de **conception**

  Identification de l'interface des classes :

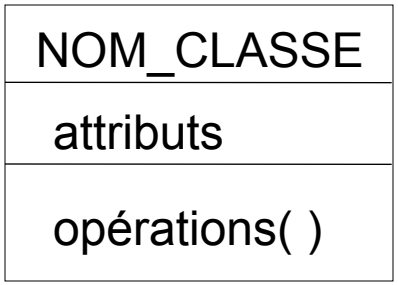
  type des objets, comportement externe, façon interne de les mettre en œuvre

  Indépendant du logiciel

  - Attributs :
    - Type
    - Valeurs par défaut
    - Degré de visibilité
    - Caractéristique
  - Opérations :
    - Signature
    - Degré de visibilité
    - Caractéristique



Notation des attributs et opérations



- Attribut

  <visibilité> <nomAttribut> : <type> = <valeur par défaut>

- Opération

  <visibilité> <nomOpération> (listeParamètres) : <typeRetour>

  Paramètre d'une opération

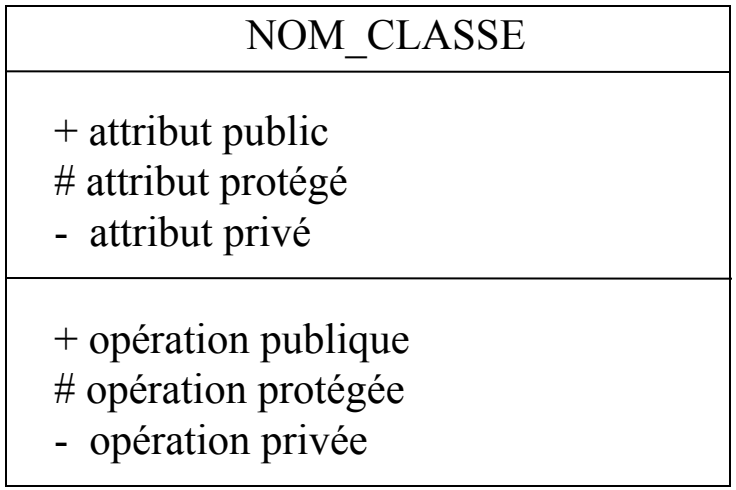
  <nom> : <type> = <valeur par défaut>

Visibilité

Visibilité = degré de protection

- \+ : publique (accessible à toutes les classes)
- \# : protégé (accessibles uniquement aux sous-classes)
- \- : privé (inaccessible à tout objet hors de la classe)

Pas de visibilité par défaut



Type

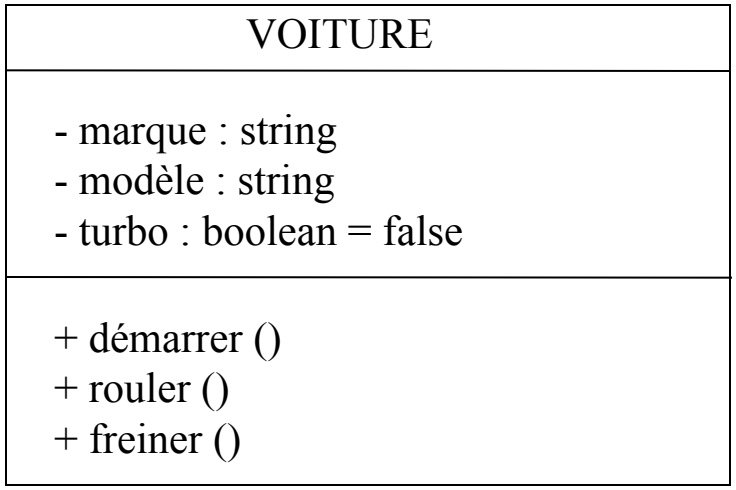
- Attribut
  - type de base (entier, booléen, caractère, tableau…)

    integer, double, char, string, boolean, currency, date, time, void
  - Au niveau analyse, pas d'attribut d'instance (on utilise une association)
  - Au niveau conception, on décide quelles associations peuvent être représentées par des attributs

- Opération
  - Type quelconque (de base ou classe)

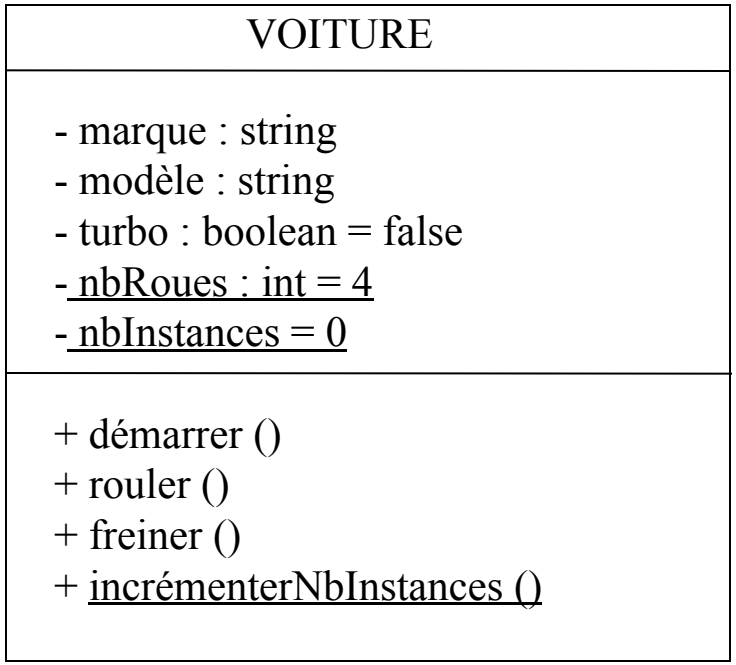
Valeur par défaut des attributs

- Affectée à l'attribut à la création des instances de la classe
- En analyse, on se contente souvent d'indiquer le nom des attributs



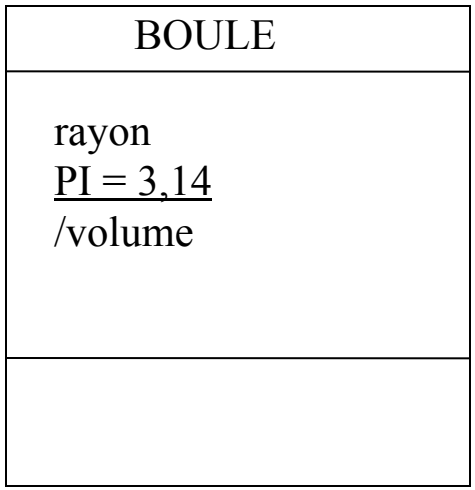
Attributs et opérations de classe

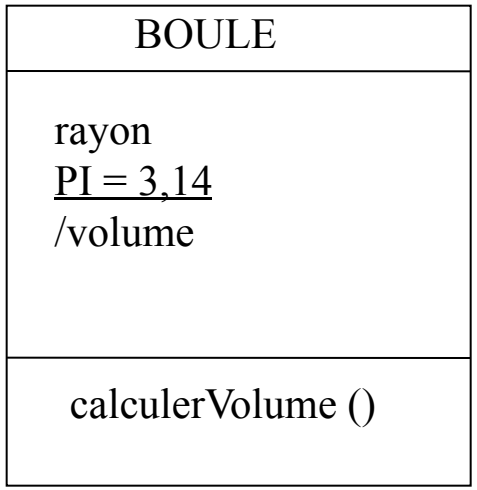
- Notation : nomAttribut ou nomOpération
- Attribut partagé par toutes les instances de la classe
- Opération sur des attributs de classes



Attributs et opérations dérivés

- Notation : /nomAttribut
- Propriété redondante, dérivée d'autres propriétés déjà allouées
- En conception, un attribut dérivé peut donner lieu à une opération qui encapsulera le calcul effectué





Les relations entre classes

- La composition
- L'agrégation

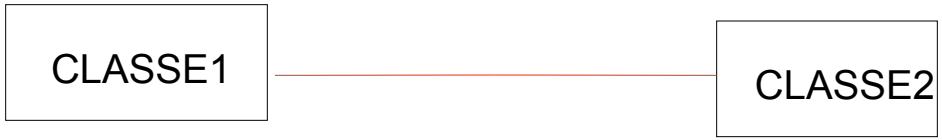
Association

- La généralisation

Héritage

L'association

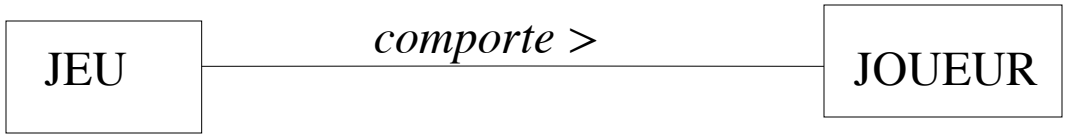
- L'association exprime une connexion sémantique entre classes
- La plupart des associations sont binaires : connectent 2 classes
- Notation :



Nommage des associations

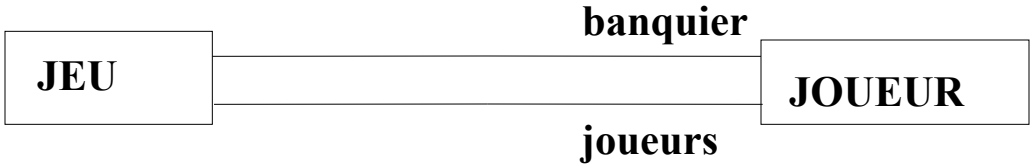
- Indication du sens de **lecture** de l'association
- Une association peut se lire dans les 2 sens, en fonction des besoins
- Usage : forme verbale, active ou passive – Pb : confusion avec le comportement

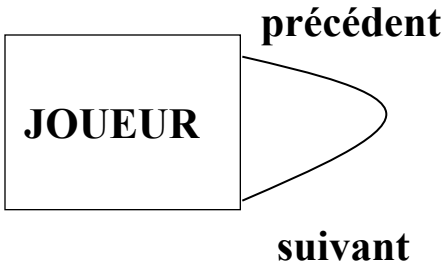
  ➔ nommage des extrémités : plus en phase avec la représentation statique



Nommage des rôles

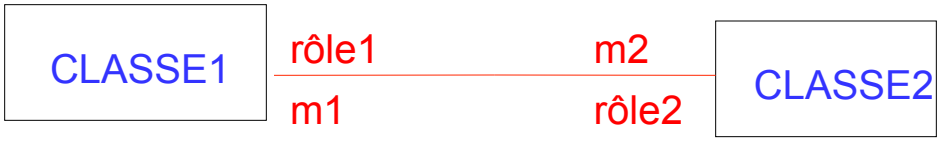
- Le rôle décrit comment une classe voit une autre classe à travers une association
- Une association a par essence 2 rôles, selon le sens dans lequel on la regarde
- Usage : Forme nominale





Multiplicité (Arité)

- Chaque rôle porte une indication de multiplicité : nombre d'objets de la classe considérée pouvant être liés à un objet de l'autre classe
- Information portée par le rôle



Valeur :	signification :
1	Un et un seul
0..1	Zéro ou un
M .. N	De M à N (entiers naturels)
*	De zéro à plusieurs
0 .. *	De zéro à plusieurs
1 .. *	De un à plusieurs

Lecture d'un association

« Un JEU comporte plusieurs joueurs »

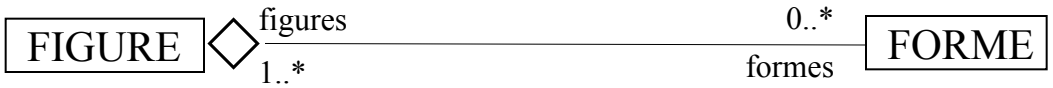
« Un JOUEUR joue dans un seul jeu »

L'agrégation

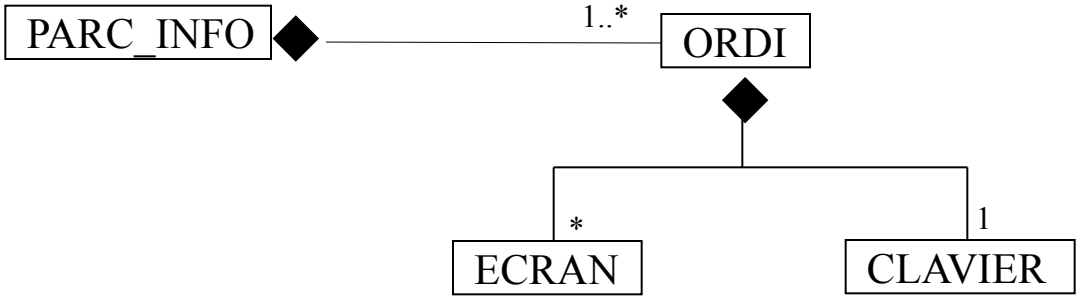
- Connexions bidirectionnelles dissymétriques
  - une des extrémités est prédominante par rapport à l'autre
  - Ne concerne qu'un seul rôle
- Représentation des relations de type :
  - tout et parties
  - composé et composants
  - maître et esclaves
- Deux type d'agrégation :
  - Agrégation partagée (par référence) – notion de co-propriété
    - La création (resp. la destruction) des composants est indépendante de la création (resp. la destruction) du composite.
    - Un objet peut faire partie de plusieurs composites à la fois.
  - Composition (par valeur) :
    - Cas particulier de l'agrégation : attributs contenus physiquement par l'agrégat
    - La création (resp. la destruction) du composite entraîne la création (resp. la destruction) des composants.
    - Un objet ne fait partie que d'un seul composite à la fois.

Exemples

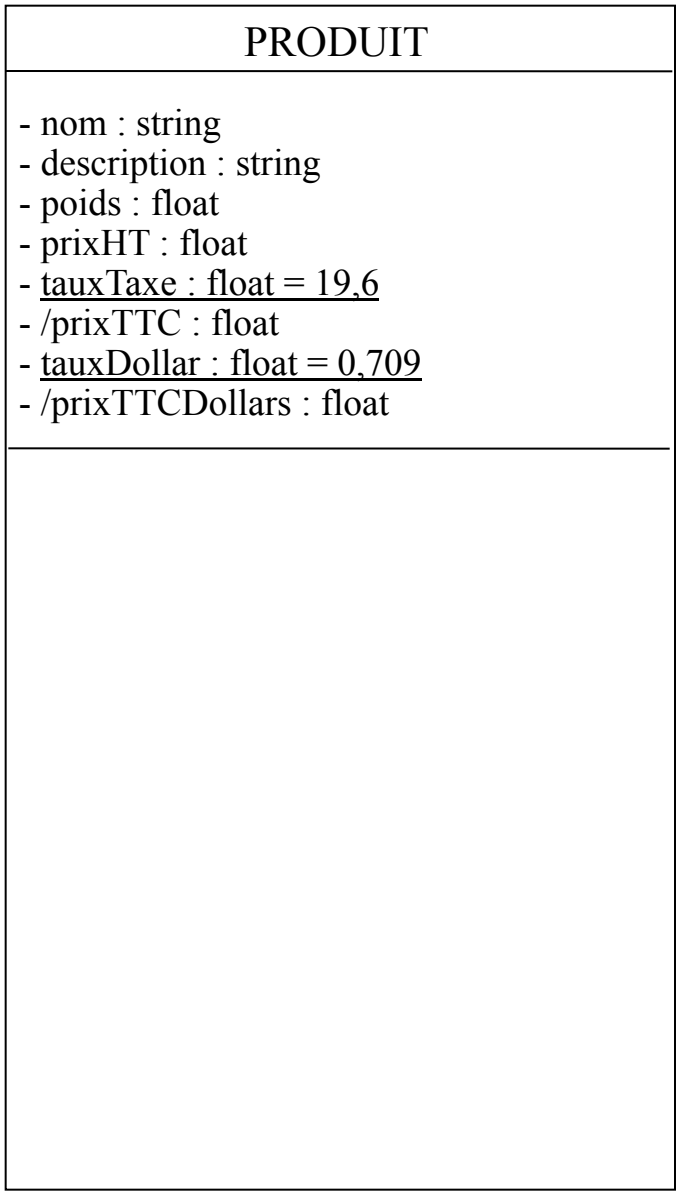
- Agrégation



- Composition

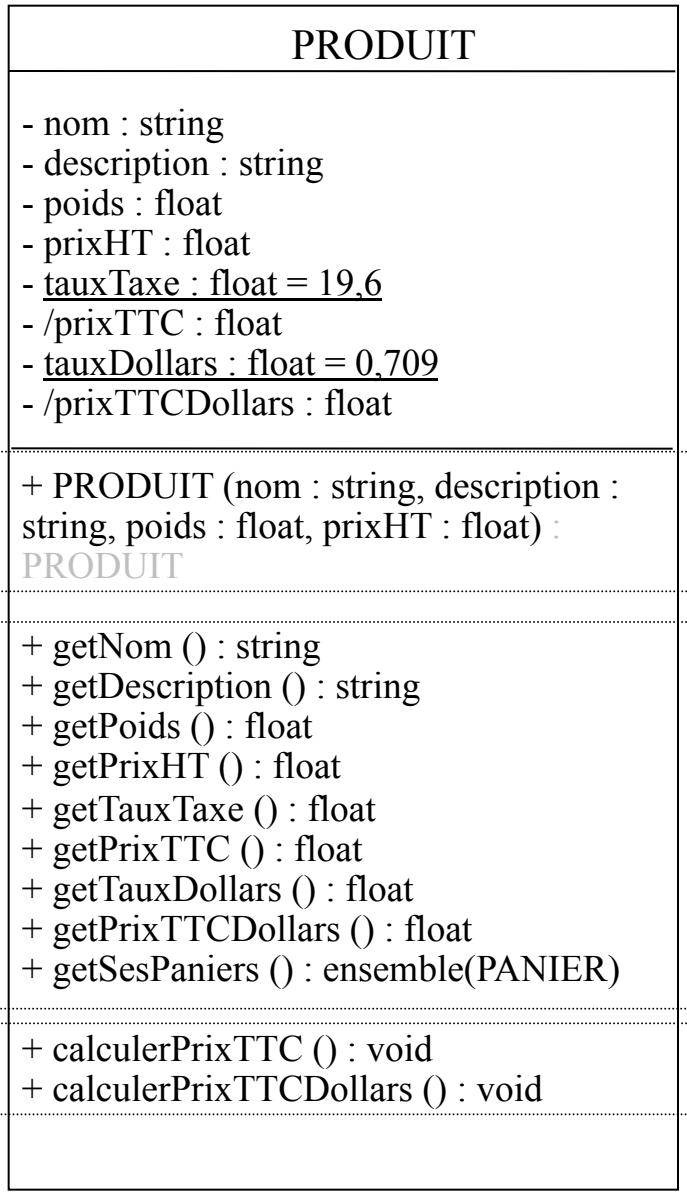


Méthode - *1ers attributs*



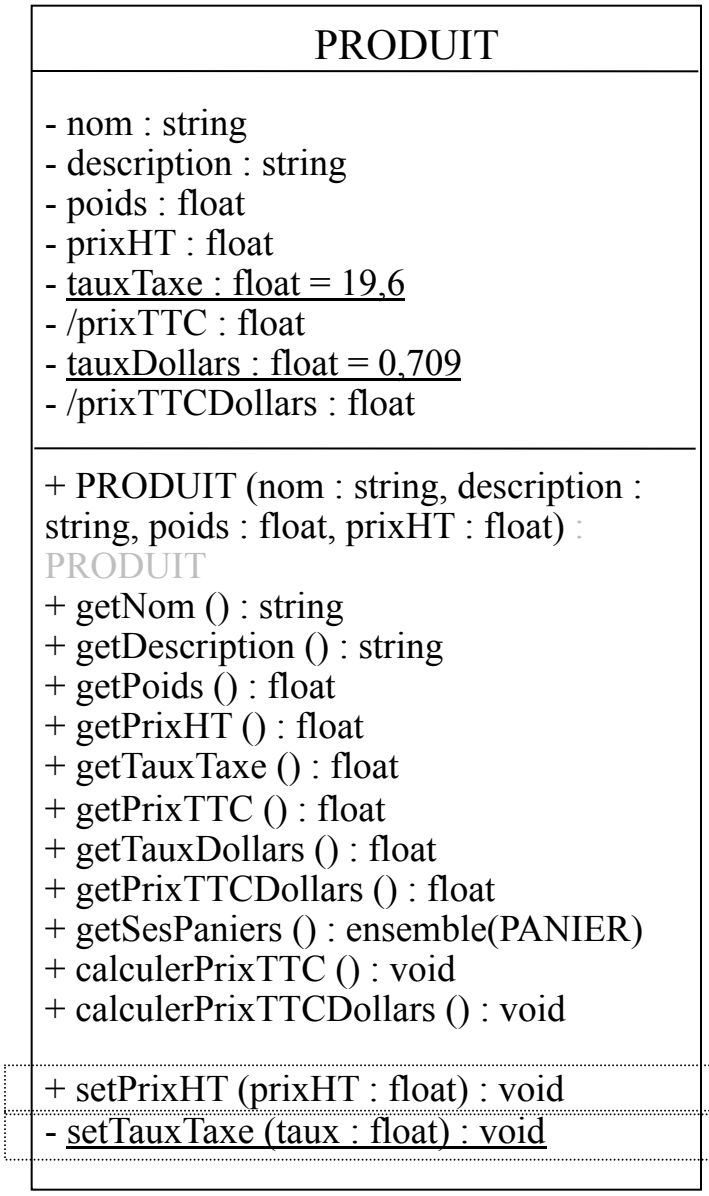
- Attributs décrivant l'objet
- Attributs de classes
- Attributs dérivés

Méthode - *1ères méthodes*



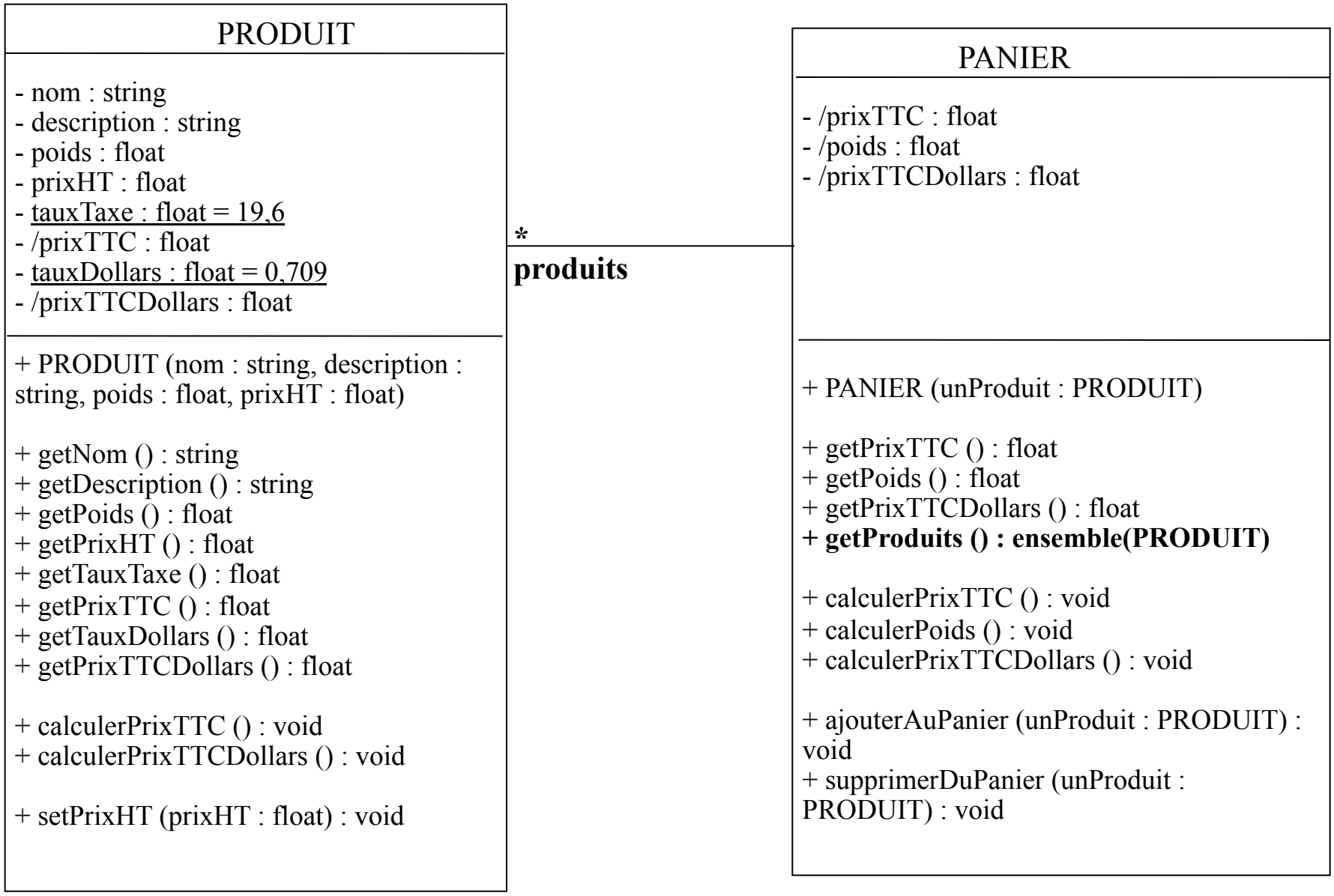
- Constructeur (s)
- Accesseurs (1 pour chaque attribut)
- Calcul des attributs dérivés

Méthode - *Autres méthodes*

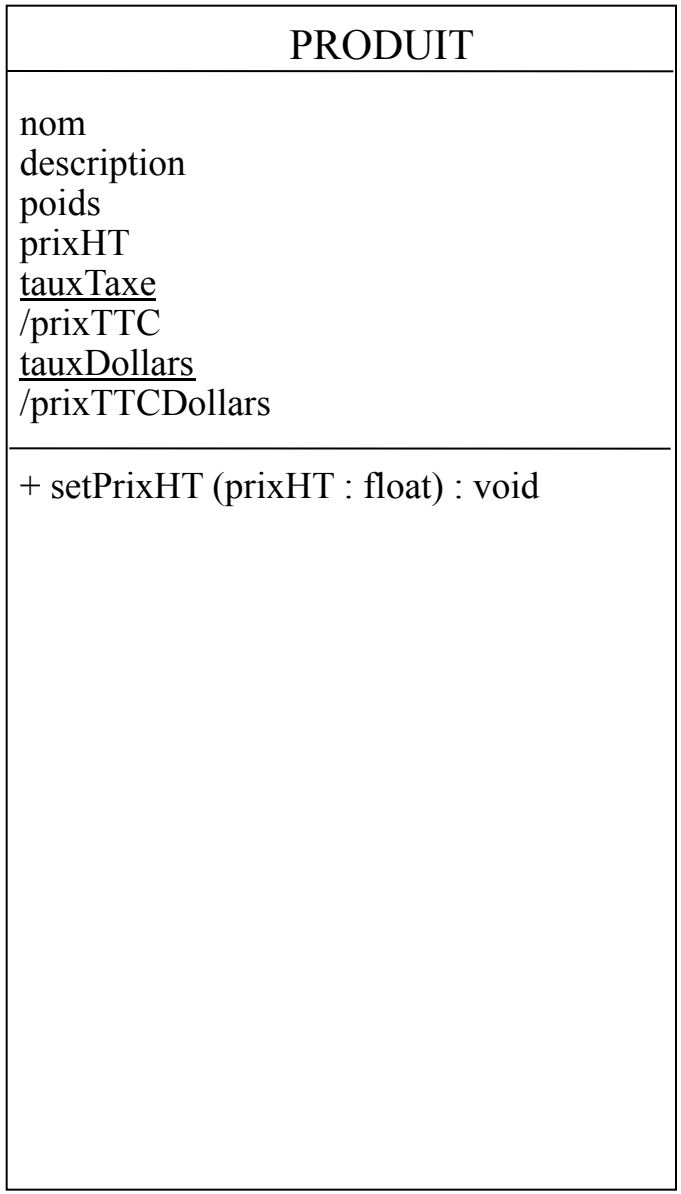


- Modifieurs
  - Publics
  - Privés
- Autres méthodes

Méthode - *Associations et rôles*



Méthode - *Version "light"*

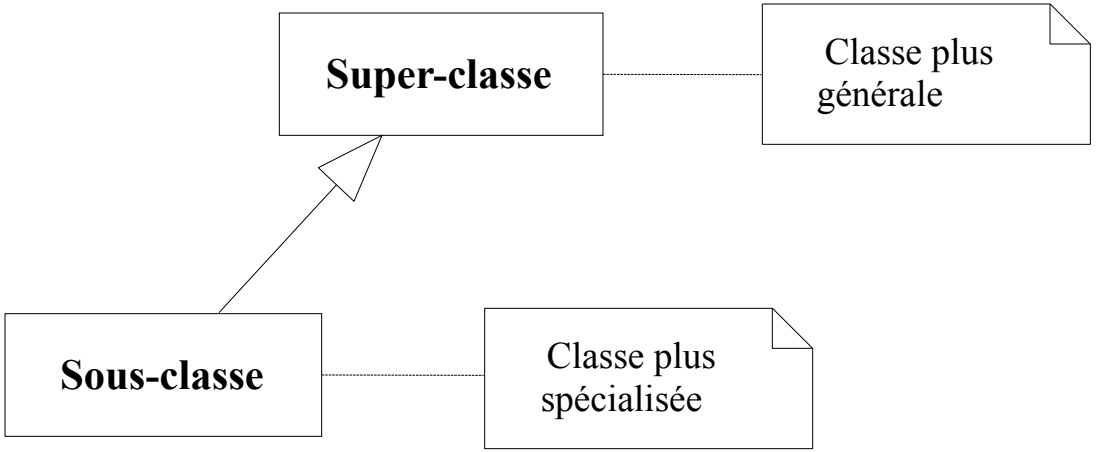


- Attributs
  - Explicite : nom
  - Implicite :
    - Visibilité privée
    - Type, valeur par défaut
- Méthodes
  - Explicite : méthodes spécifiques
  - Implicite :
    - Constructeur
    - Accesseurs "getToto" (1 par attribut et rôle)
    - Modifieurs [privés] "setToto"
    - Calcul des attributs dérivés
- Attributs explicites + accesseurs / modifieurs implicites = **Propriété**

Hiérarchies de classes

- Gérer la complexité

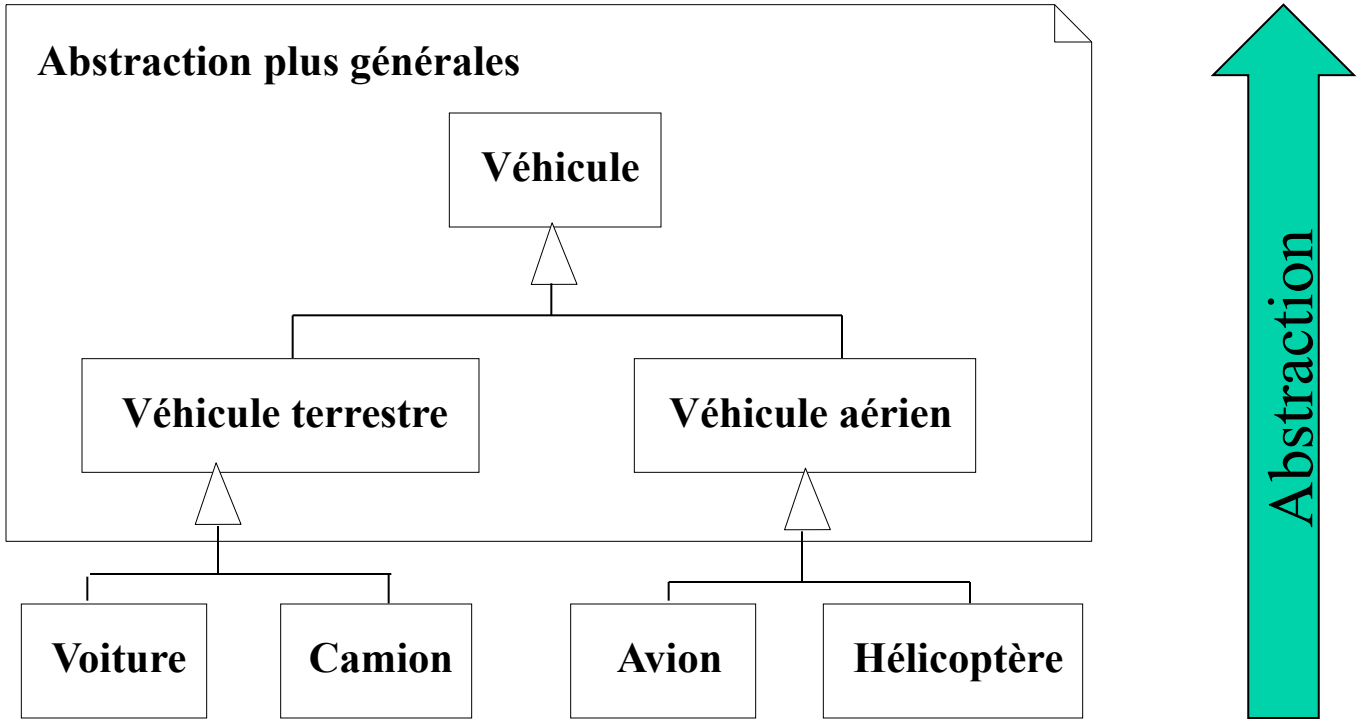
  Arborescences de classes d'abstraction croissante
- Généralisation : Super-classes
- Spécialisation : Sous-classes



Généralisation

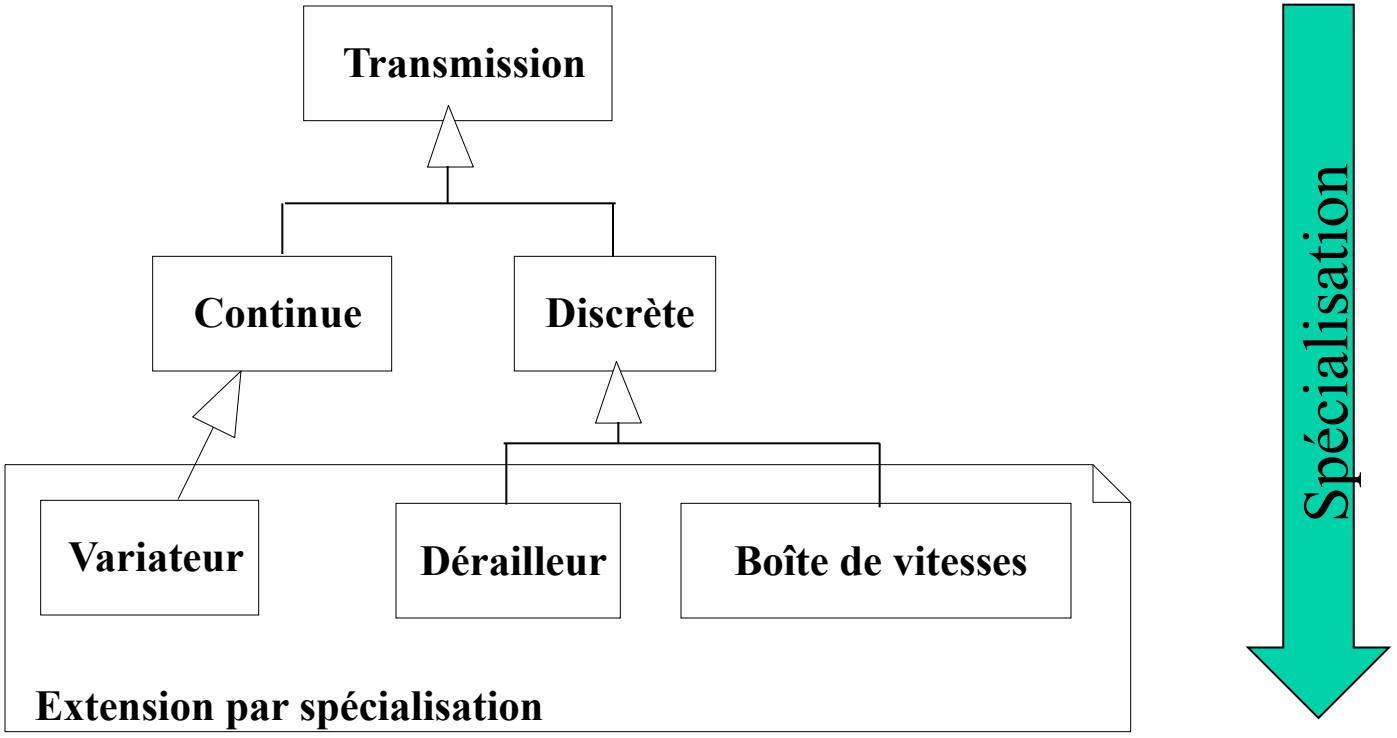
Factoriser les éléments communs

attributs, opérations et contraintes



Spécialisation

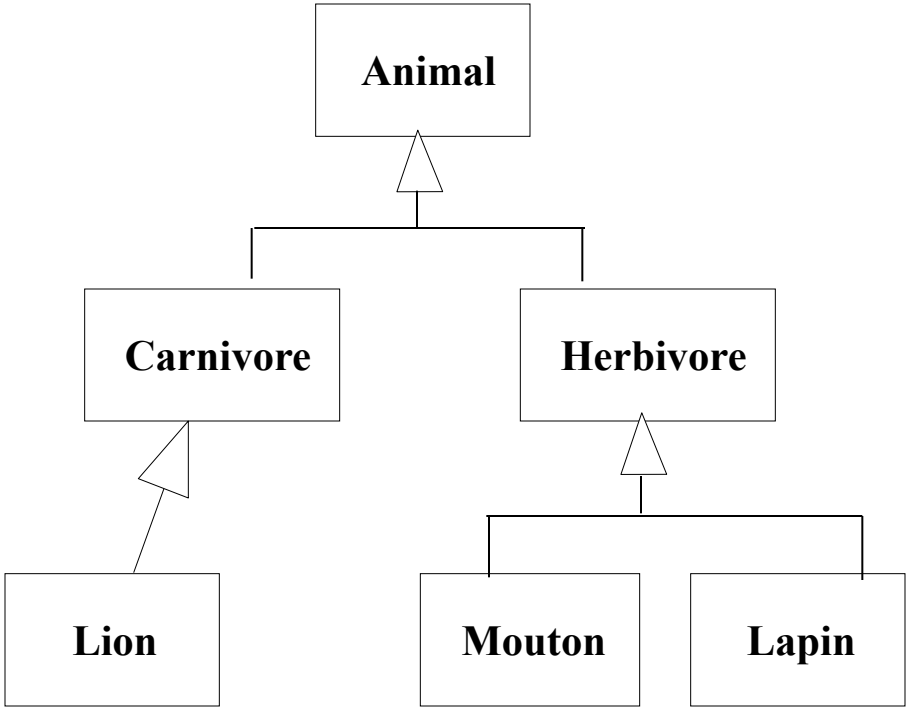
Extension cohérente d'un ensemble de classes



Propriétés de la généralisation

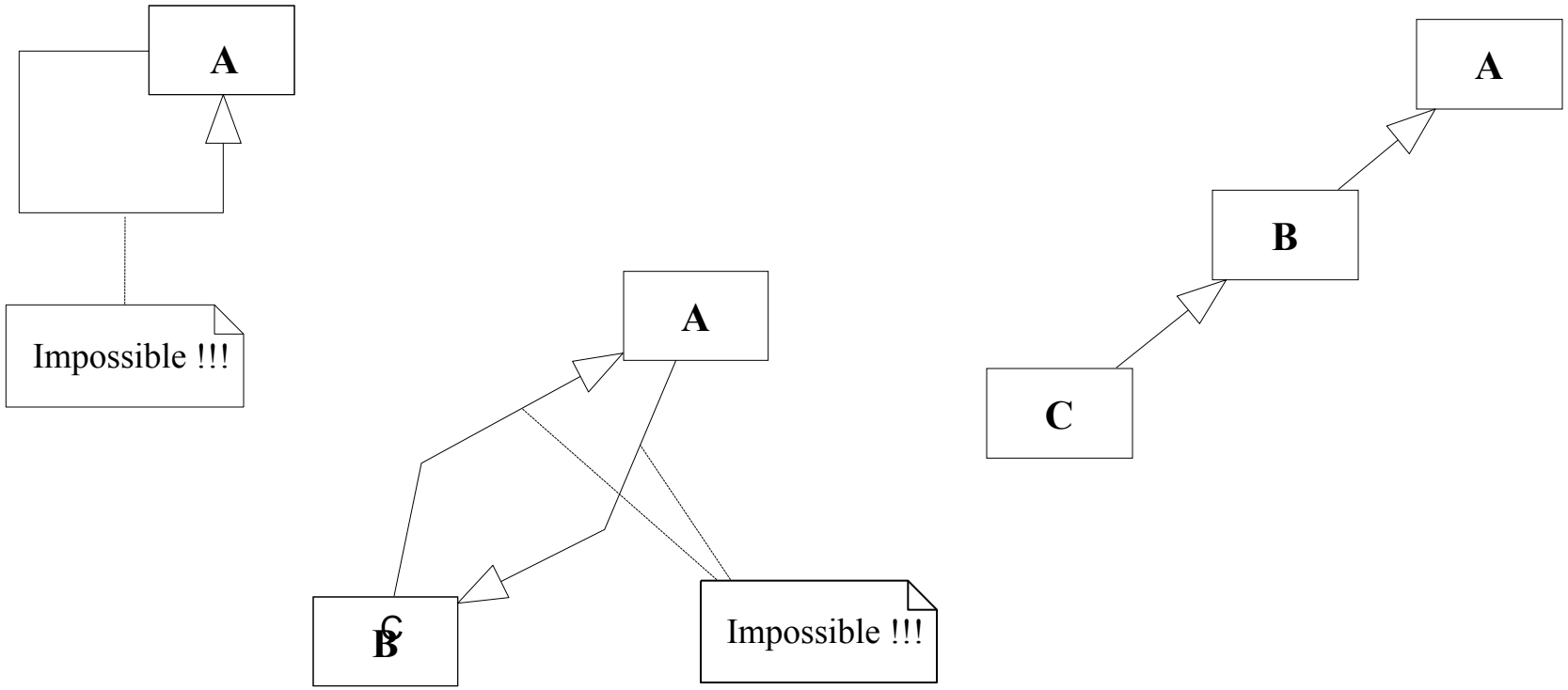
Signifie toujours : ***est un*** ou ***est une sorte de***

(polymorphisme d'héritage)

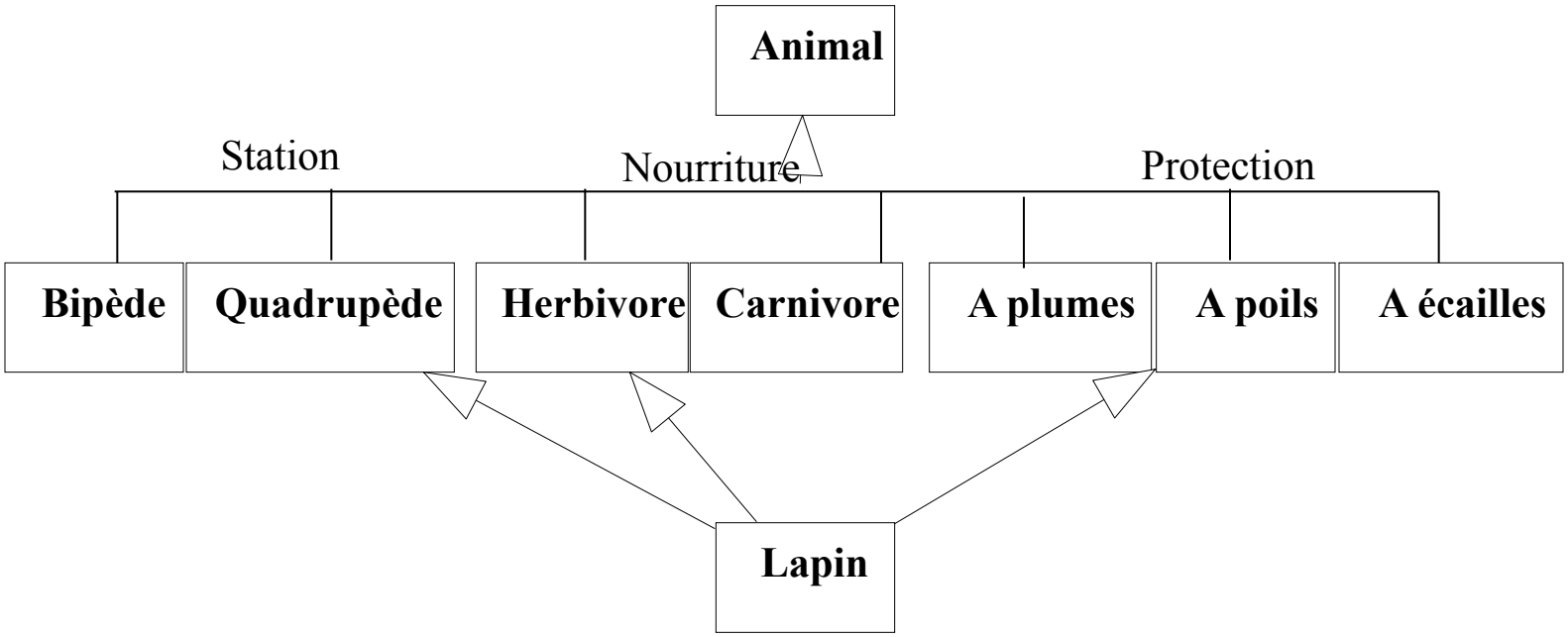


Propriétés de la généralisation

Non-réflexive, non-symétrique, transitive



Généralisation/Spécialisation multiple

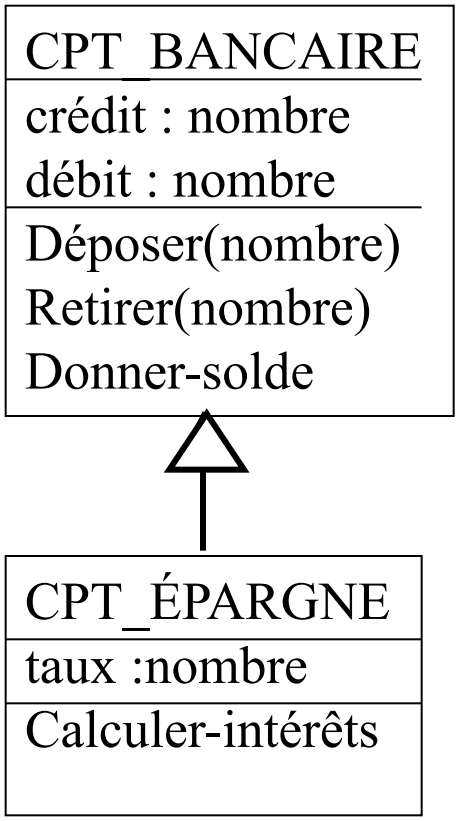


L’héritage

- Technique la plus utilisée pour réaliser la généralisation
- Construire une classe à partir d’une ou plusieurs autres classes, en partageant des attributs, des opérations et parfois des contraintes, au sein d'une hiérarchie de classes.

Exemple

Généralisation

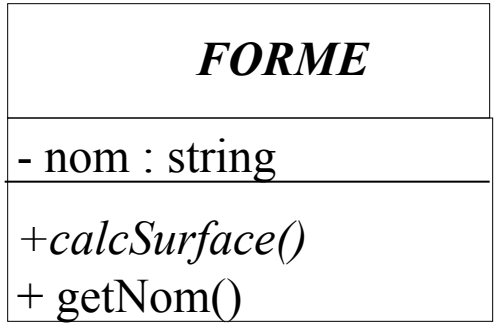


Spécialisation

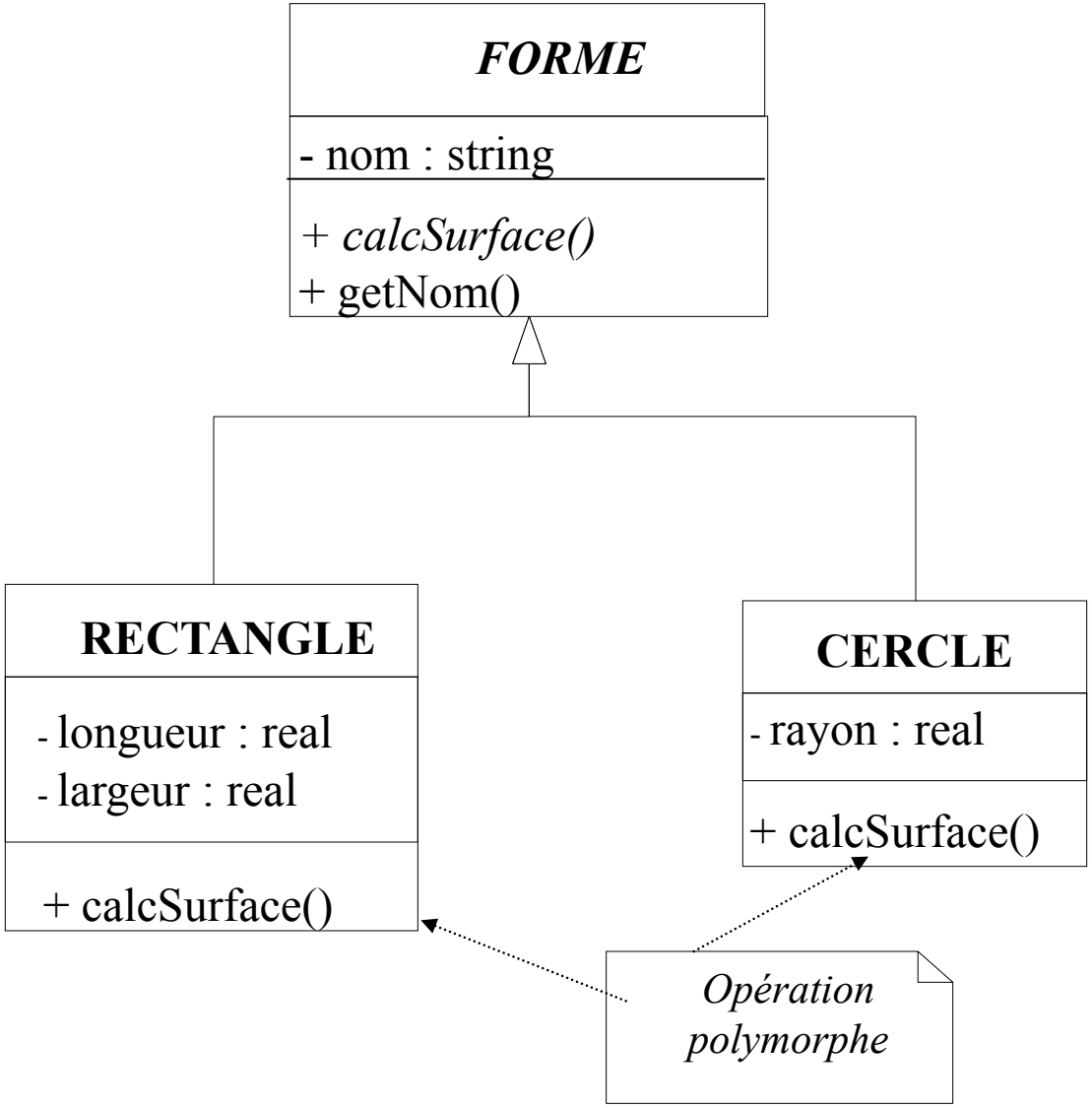
Classe et Opération abstraites

- Classe qui ne peut avoir aucune instance directe ; on écrit son nom en *italique*

- Opération incomplète qui a besoin de la classe fille pour fournir une implémentation ; on écrit son nom en *italique*.



Polymorphisme



Conclusion

- Les classes sont connectées par des relations
- L'association exprime une connexion sémantique
- L'agrégation est une forme d'association plus forte
- La généralisation permet d'ordonner les objets au sein de hiérarchies de classes

UNIFIED MODELING LANGUAGE

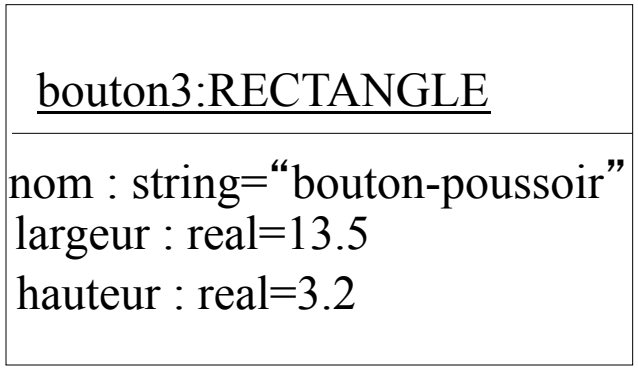
2.2. Diagramme d'objets

Définition

- Ils modélisent les instances d'éléments qui apparaissent sur les diagrammes de classe.
- Ils montrent un ensemble d'objets et leurs relations à un moment donné.
  - Instances nommées

  - Instances anonymes

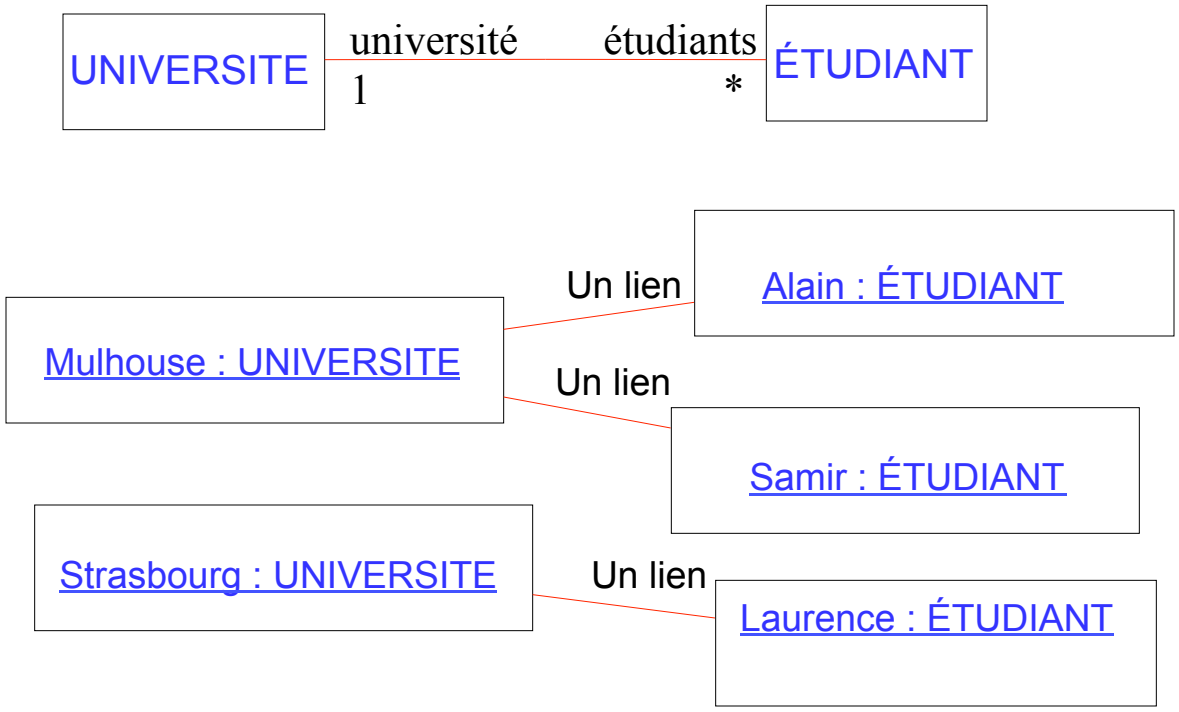
  - Instances avec valeurs d'attributs



Association/Lien

- Une association est une abstraction des liens qui existent entre les objets instances des classes associées
- Les associations se représentent de la même manière que les liens.

Une association



3. Vue de l'utilisateur

Diagramme de cas d'utilisation (DCU)

Use-Case Diagrams (UCD)

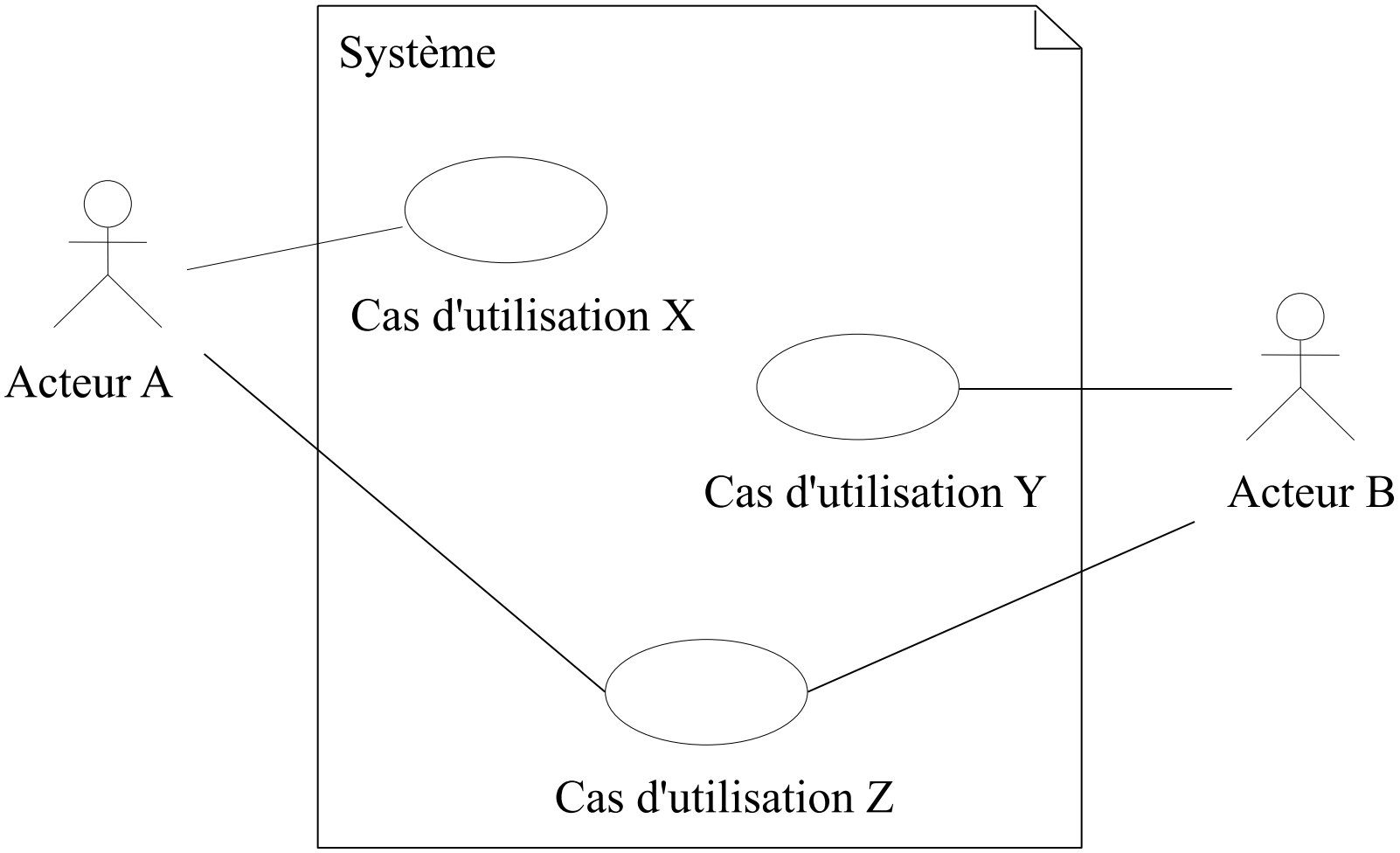
Objectifs

- Expression du comportement du système **selon le point de vue de l'utilisateur**
- Interaction entre le système informatique à développer et un **utilisateur** (ou acteur) interagissant avec le système.
- Description d'une séquence d'actions réalisées par le système et qui produit un résultat observable pour un **acteur**

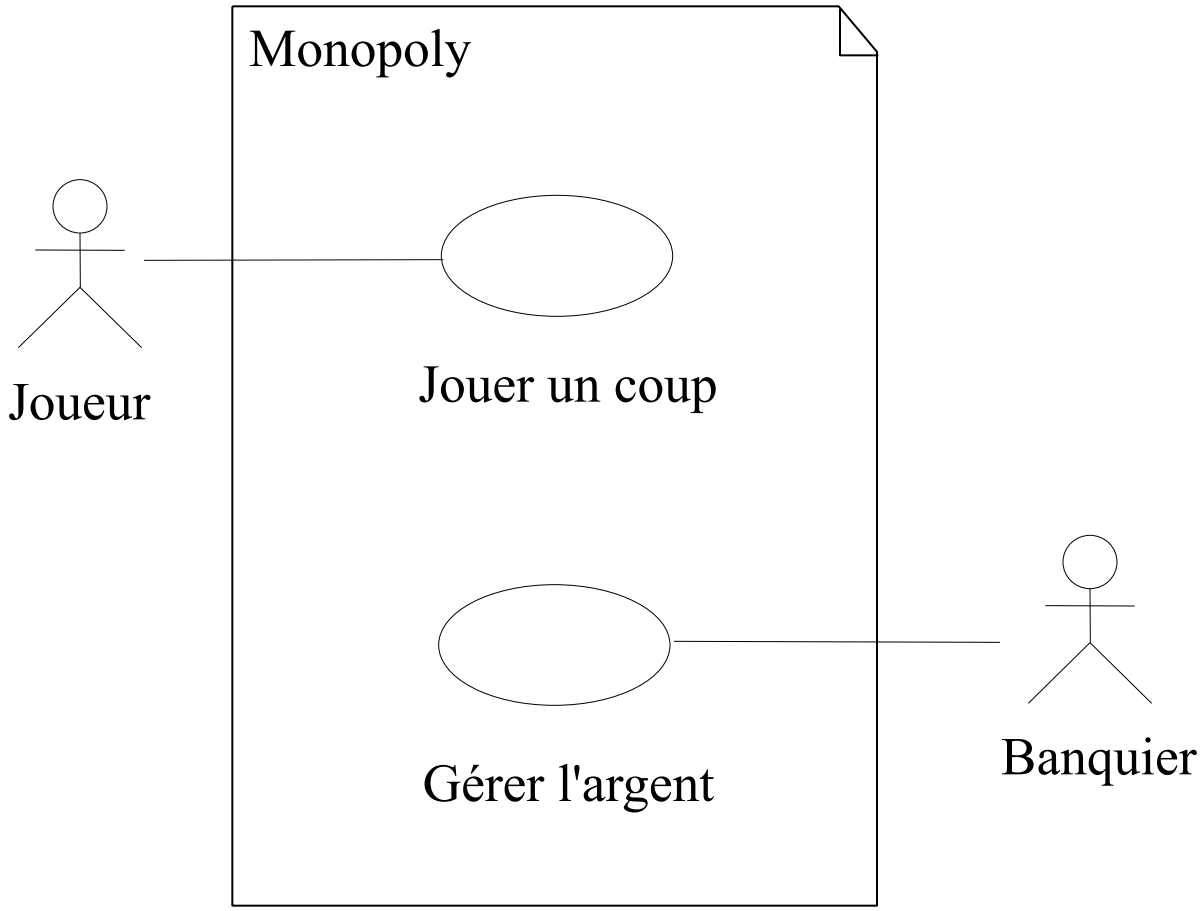
Utilité

- Constituent un moyen de déterminer les buts et l'étendue (les frontières) d'un système
- Utilisés par les utilisateurs finaux pour exprimer leur attentes et leur besoins ➔ permettent d'impliquer les utilisateurs dès les premiers stades du développement
- Support de communication entre les équipes et avec les clients
- Découpage du système global en grandes tâches qui pourront être réparties entre les équipes de développement
- Permettent de concevoir les Interfaces homme-Machine
- Constituent une base pour les tests fonctionnels

Notation


Système
Cas d'utilisation X
Acteur A
Cas d'utilisation Y
Acteur B
Cas d'utilisation Z

Exemple 1 : *Monopoly*



Les acteurs - *Définition*

- Un acteur est une **personne** ou un (autre) **système** qui interagit avec ~~un~~ le **système**, en échangeant de l'information (en entrée et en sortie)
- On trouve les acteurs en observant les **utilisateurs directs** du système, ceux qui sont responsable de sa **maintenance**, ainsi que les **autres systèmes** qui interagissent avec le système
- On doit raisonner en terme d'**utilisation** au lieu de communication
- Comparaison acteur Merise II/UML :
  - Modèle de Contexte de Merise II : pas les acteurs directs
  - Cas d'utilisation d'UML : acteurs directs

Les acteurs - *Représentent un rôle*

- On ne doit pas raisonner en terme d'entité physique, mais en terme de **rôle** que l'entité physique joue
- 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 (joueur, banquier)
- Plusieurs personnes peuvent également jouer le même rôle, et donc agir comme le même acteur (tous les joueurs)

Les cas d'utilisations - *Définition*

- Les cas d'utilisation :
  - représentent le **dialogue** entre l'acteur et le système de manière abstraite
  - **Ensemble de scénarios** au sein d'une description unique

- Cas d'utilisation *vs.* scénario :
  - **Cas d'utilisation** : représente un cas en général, une représentation générale et synthétique d'un ensemble de scénarios similaires.
    (exemple : « un joueur joue un coup»)
  - **Scénario** : cas d'utilisation spécifique
    (exemple : « le joueur Pierre joue : il obtient 7 avec les dés, se déplace rue de Belville et achète la propriété»)

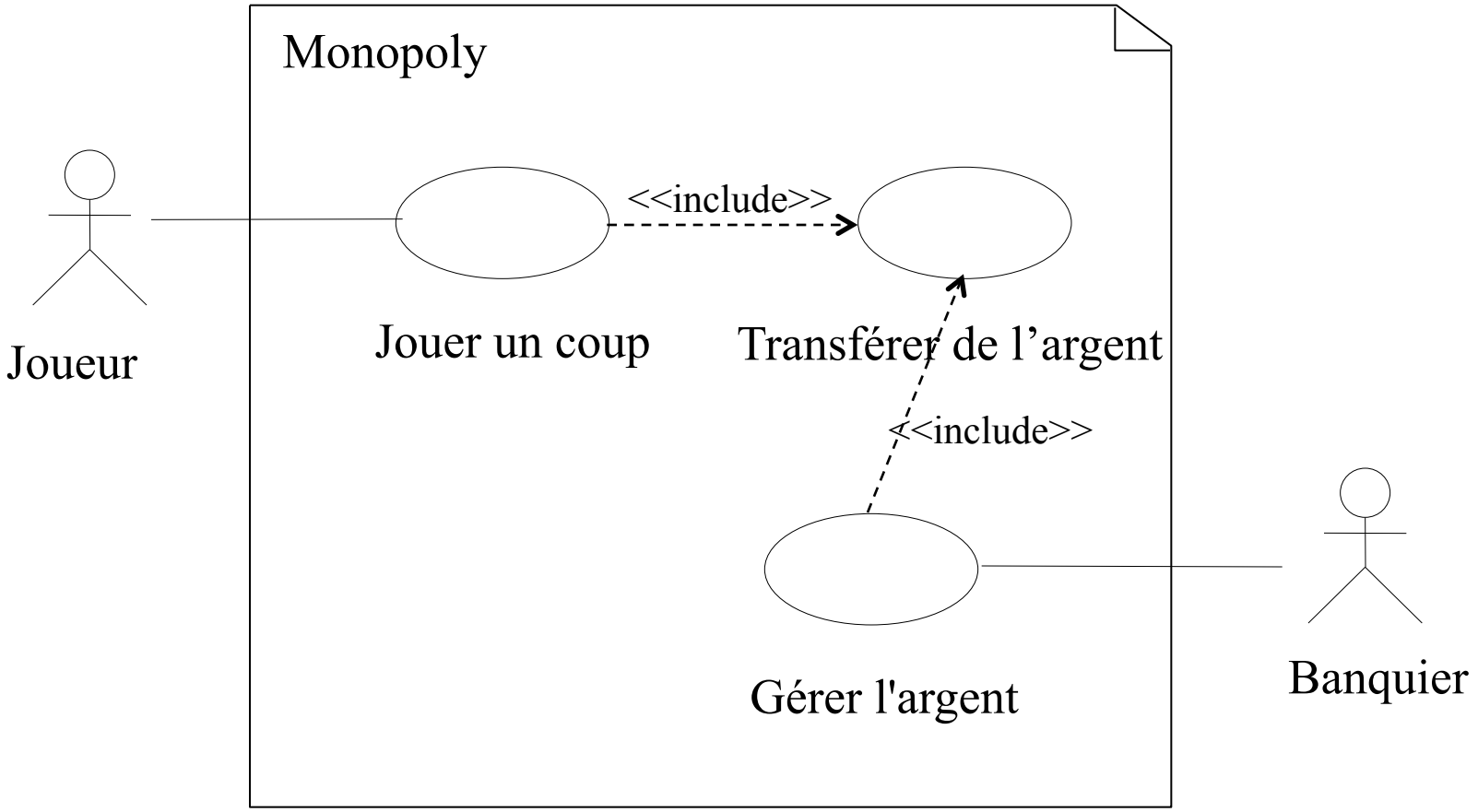
Conception d'un DCU

- Questions à se poser pour déterminer les acteurs :
  - Quels sont les rôles possibles ?

- Questions à se poser pour déterminer les cas d'utilisation :
  - Quelles sont les tâches de l'acteur ?
  - Quelles informations l'acteur doit-il créer, sauvegarder, modifier, détruire ou simplement lire ?
  - L'acteur devra-t-il informer le système de changements externes ?
  - Le système devra-t-il informer l'acteur de conditions internes au système ?

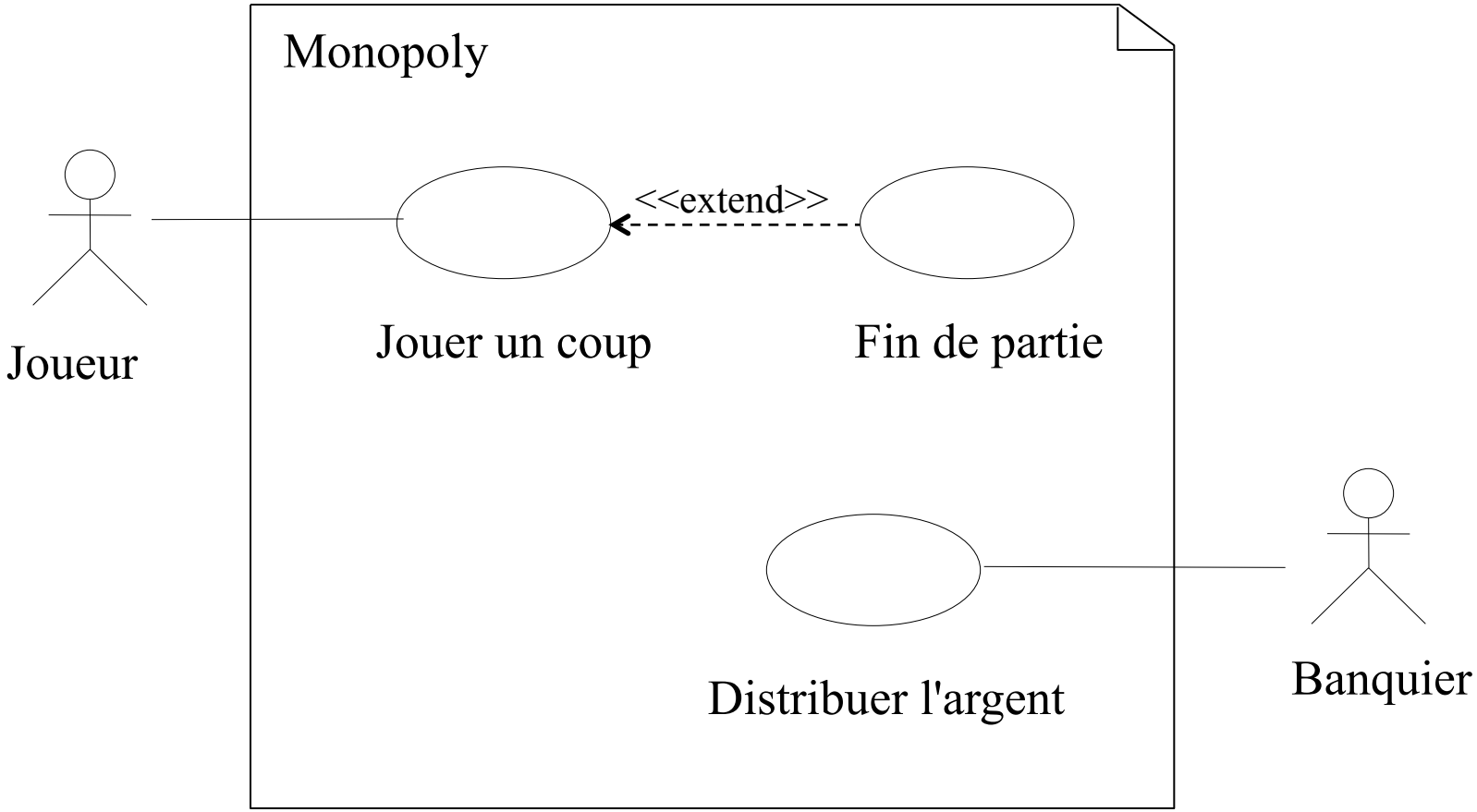
Description plus fine - Relation d'inclusion <<include>>

<<include>> permet de factoriser les parties de traitement communes à plusieurs cas d'utilisation



Description plus fine - Relation d'extension <<extend>>

<<extend>> permet de modéliser la partie d'un cas d'utilisation considérée comme facultative dans le comportement du système (cas d'exception)



Spécification

- Pour chacun des cas d'utilisation on fournira une spécification sous la forme
  - d'un texte (pas de norme établie)
  - d'un diagramme (de séquence ou de collaboration)

- Spécification sous forme textuelle - plusieurs étapes :
  1. Rédigée par l'utilisateur
  2. Proposition technique

- Chaque étape de spécification peut donner lieu à une version du diagramme de classes, de plus en plus détaillée

Exemple de format de spécification sous forme textuelle

Système : nom du système
Acteur primaire : nom de l'acteur qui initie le cas
Objectif : nom du cas (description synthétique)
Précondition : pour la réalisation du cas
Scénarios :

1 – 1ère étape
2 – 2ème étape
…

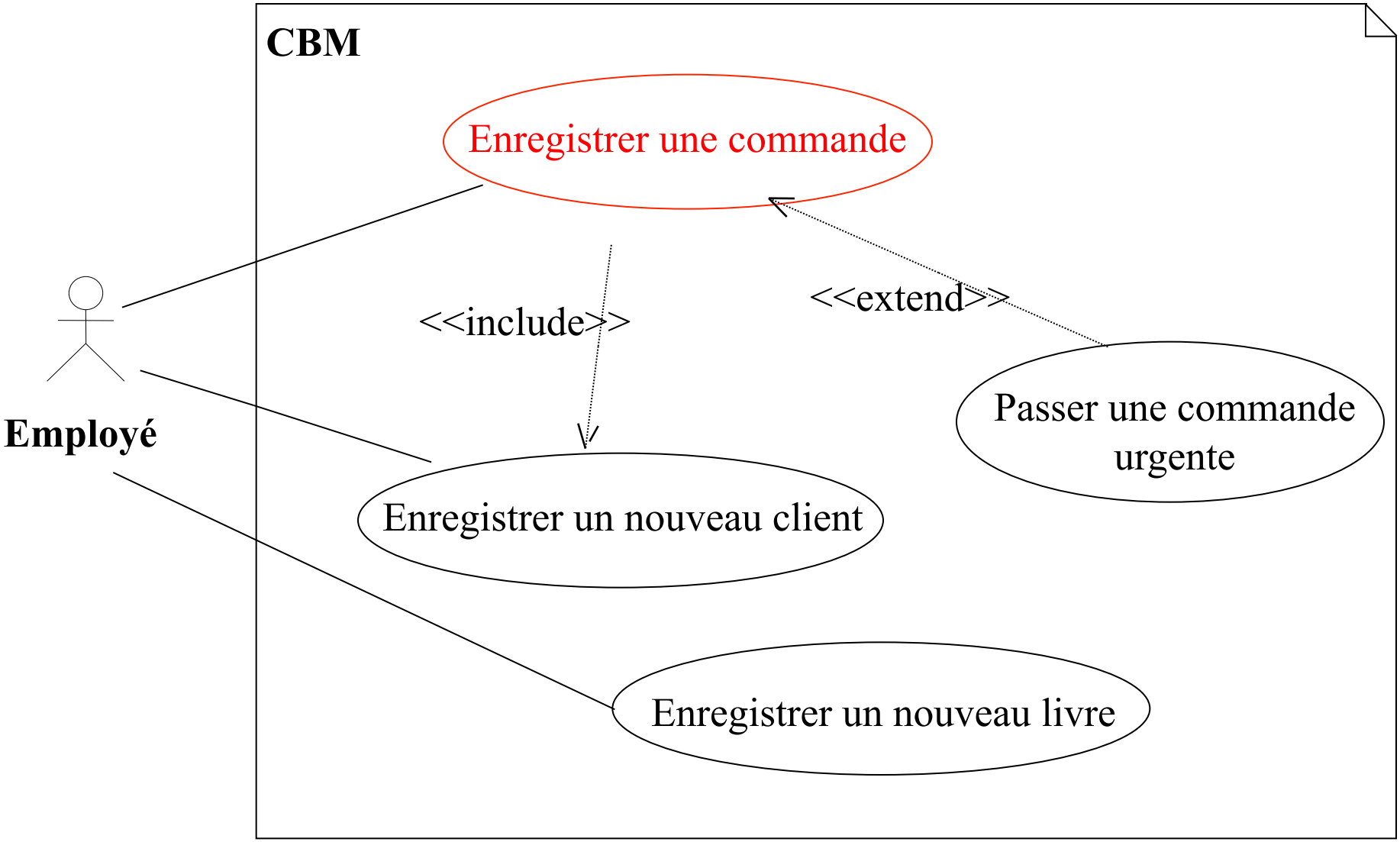
Exception :

1a – cas particulier a de la 1ère étape
1b – cas particulier b de la 1ère étape
2a – cas particulier de la deuxième étape
…

Exemple 2 : *La CBM*

- La CBM (Computer Books by Mail) est une société de distribution d'ouvrages d'informatique qui agit comme intermédiaire entre les librairies et les éditeurs.
- Elle prend des commandes en provenance des libraires, s'approvisionne (à prix réduit) auprès des éditeurs concernés et livre ses clients à réception des ouvrages.
- Il n'y a donc pas de stockage.
- Seules les commandes des clients solvables sont prises en compte.

Diagramme de cas d'utilisation



Spécification textuelle du cas *Enregistrer une commande*

1ère version : Rédaction par l'utilisateur

Système : CBM
Acteur primaire : l'employé de la coopérative
Objectif : enregistrer une commande de livres
Précondition : le libraire existe
Scénarios :

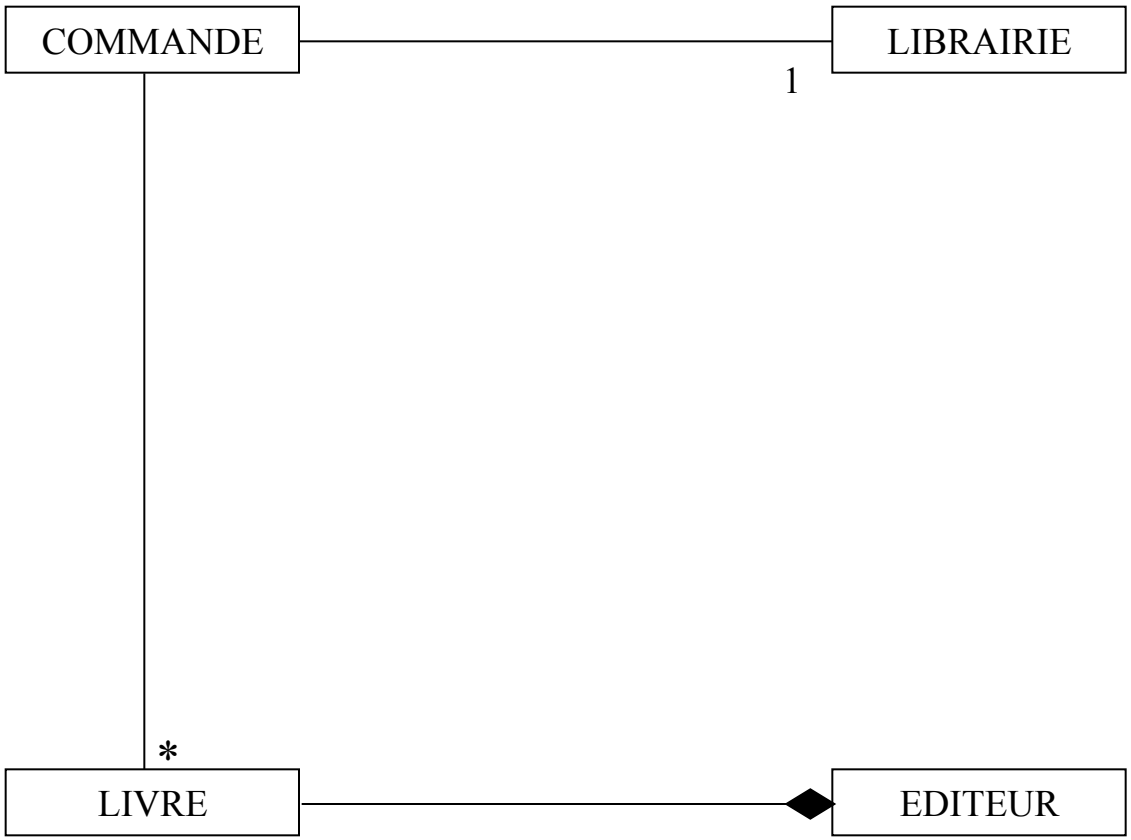
1 - l'employé vérifie la solvabilité du libraire
2 - l'employé vérifie l'existence du livre
3 - l'employé précise la quantité

Exception :

1a - le libraire n'est pas solvable (l'employé est informé)
2a - le livre n'existe pas (l'employé est informé)

Diagramme de classes

1ère version : Spécification simplifiée



Spécification textuelle du cas *Enregistrer une commande*

2ème version : Proposition technique

Système : CBM
Acteur primaire : l'employé de la coopérative
Objectif : enregistrer une commande de livres
Précondition : le libraire existe
Scénarios :

1 - l'employé vérifie la solvabilité du libraire
* *une instance de commande est créée*

2 - l'employé vérifie l'existence du livre
* *un lien entre l'instance de commande et une instance de livre est créé*

3 - l'employé précise la quantité
* *la quantité est ajoutée (classe associative)*

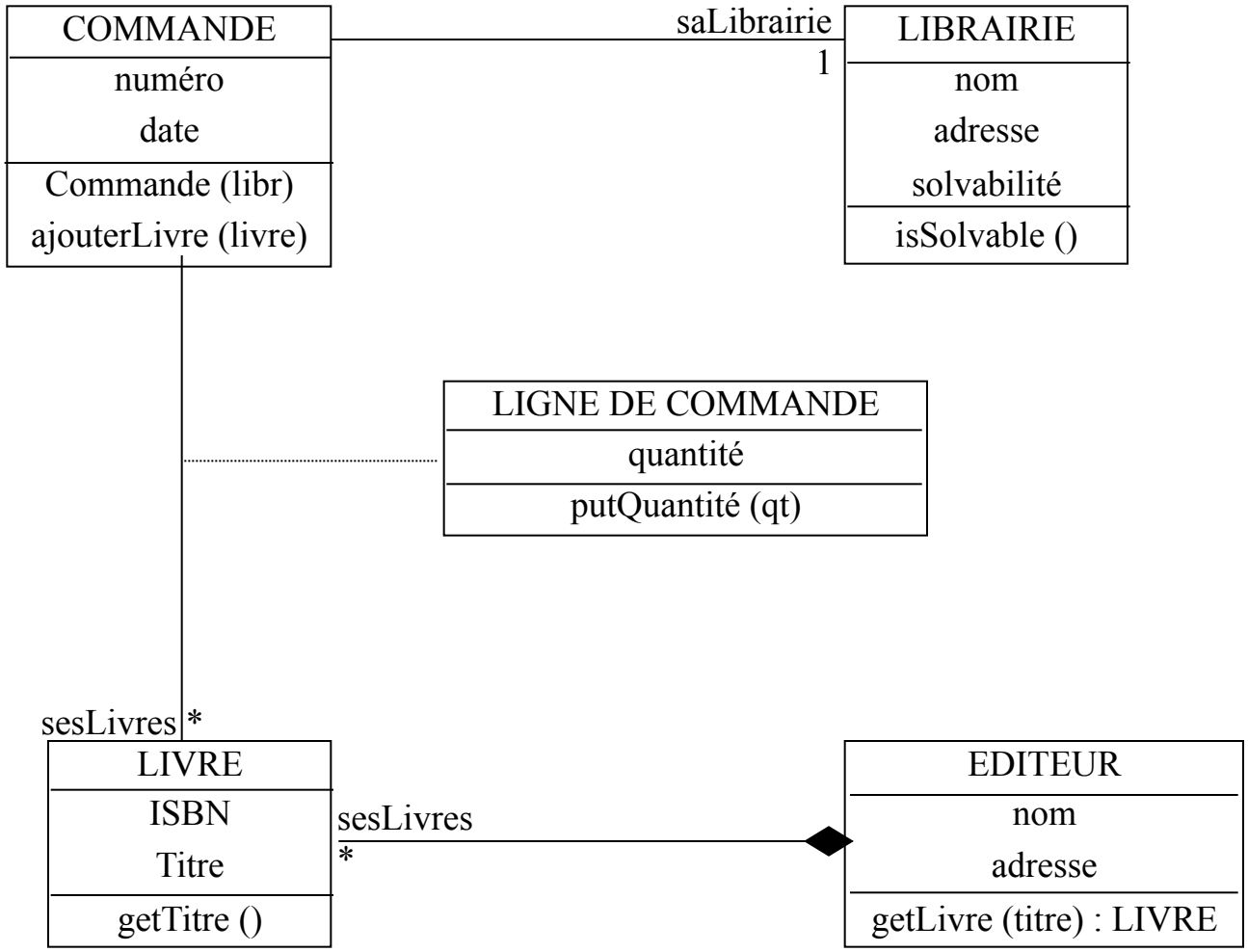
Exception :

1a - le libraire n'est pas solvable
* *un message est affiché*

2a - le livre n'existe pas
* *un message est affiché*

Diagramme de classes

2ème version : Spécification intermédiaire



4. Vue fonctionnelle

4.1. Diagramme de collaboration (DCO)

4.2. Diagramme de séquence (DES)

Rôle

- Description de scénarios particuliers
- Représente le fonctionnement du système du point de vue du concepteur
- Mise en valeur des passages de messages (flots de données ou de contrôles, évènements) entre acteur et objet, ou entre objets, de manière chronologique

- On représente le fonctionnement au niveau des instances

UNIFIED MODELING LANGUAGE

4.1. Diagramme de collaboration (DCO)

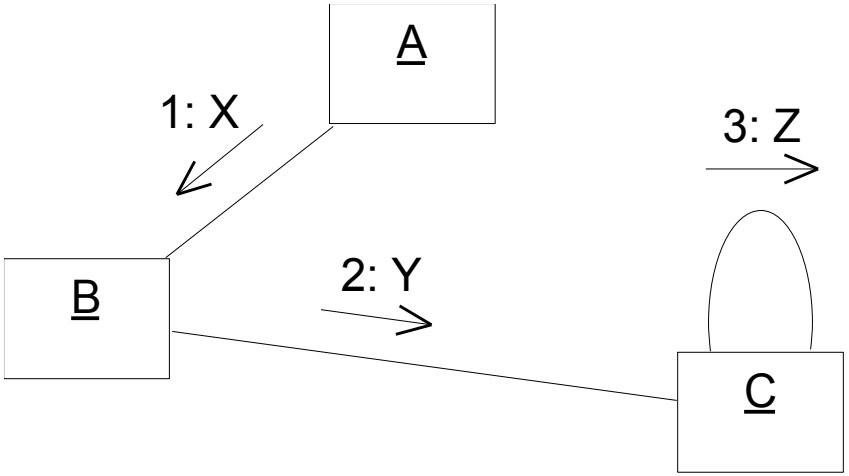
Diagramme de collaboration

- Comporte :
  - des objets dans une situation donnée
  - les liens qui relient les objets qui se connaissent
  - les messages échangés entre les objets, représentés le long de ces liens

- L'ordre d'envoi des messages est matérialisé par un numéro de séquence

Notation

- *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 un message **Z**.*



Exemple : *La CBM*

- Enregistre une commande de livres

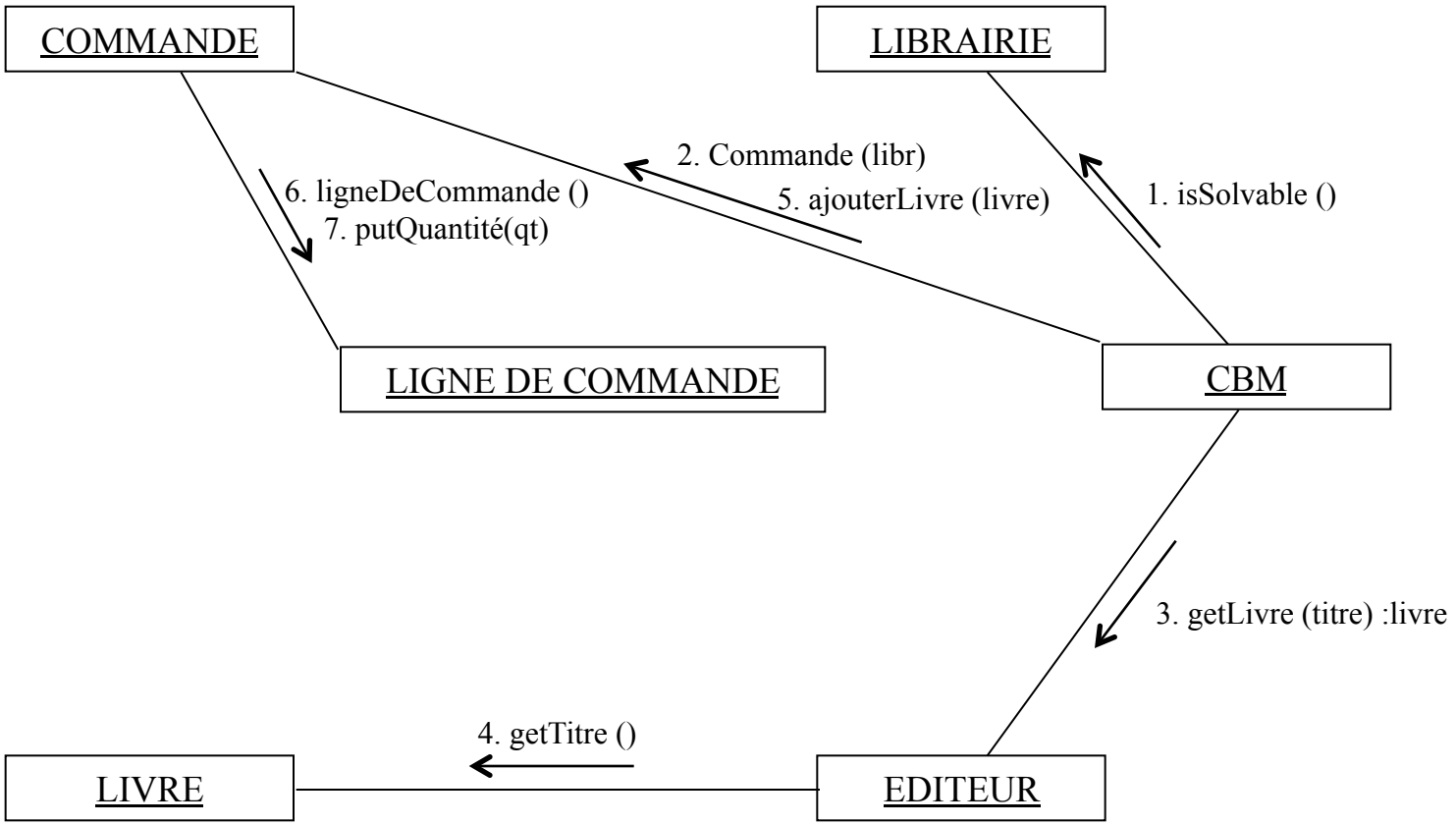
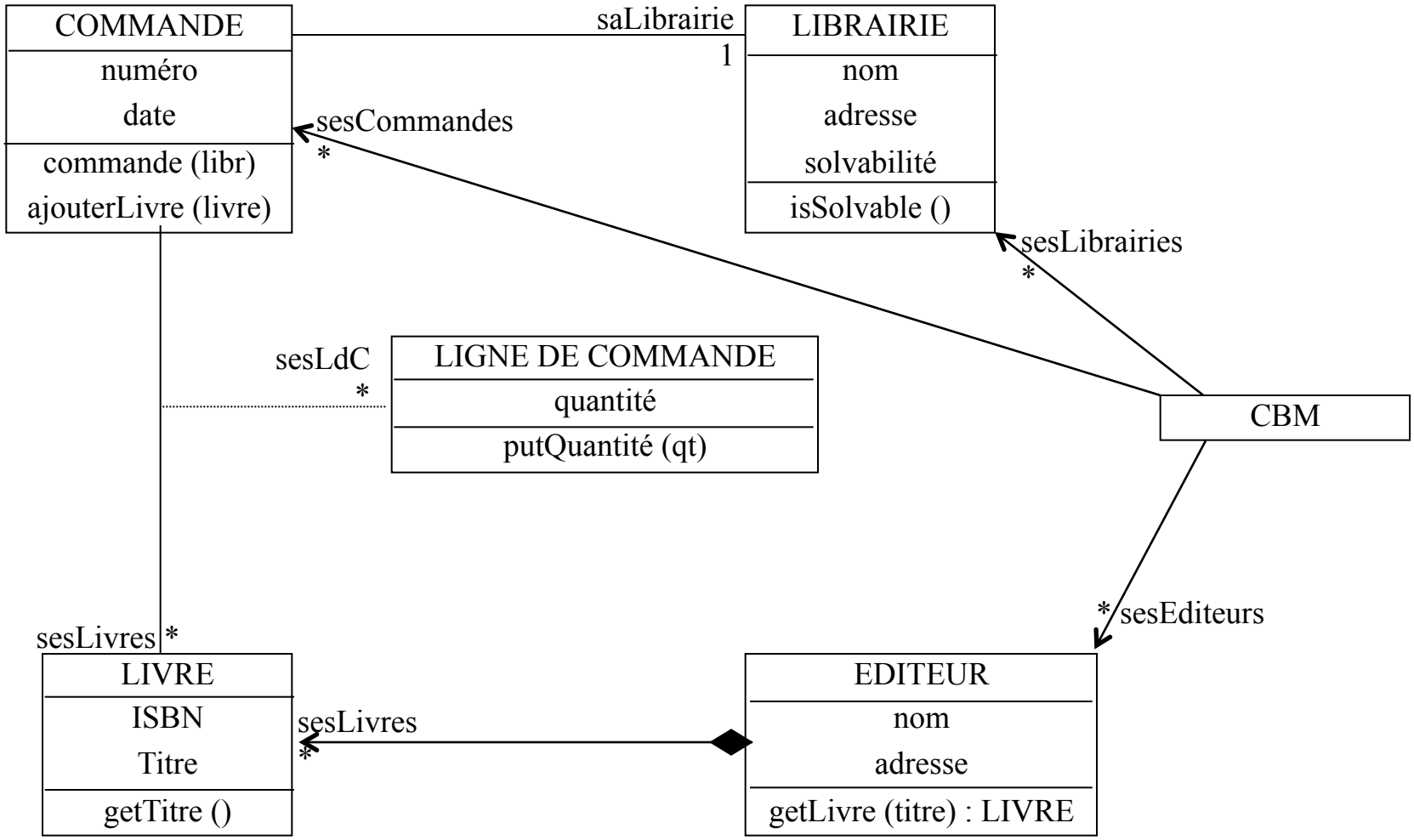


Diagramme de classes de la CBM

3ème version



UNIFIED MODELING LANGUAGE

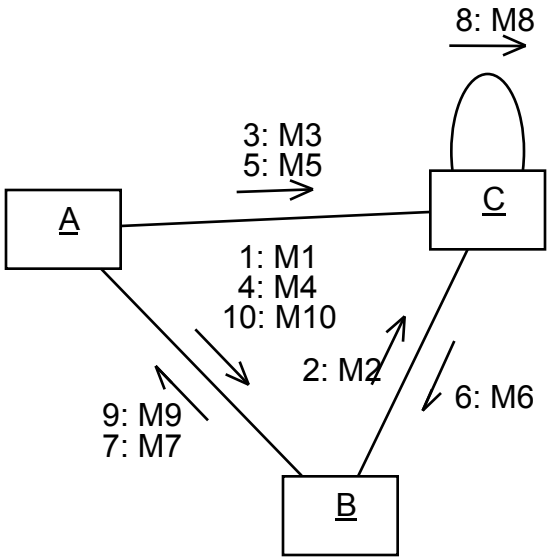
4.2. Diagramme de séquence (DES)

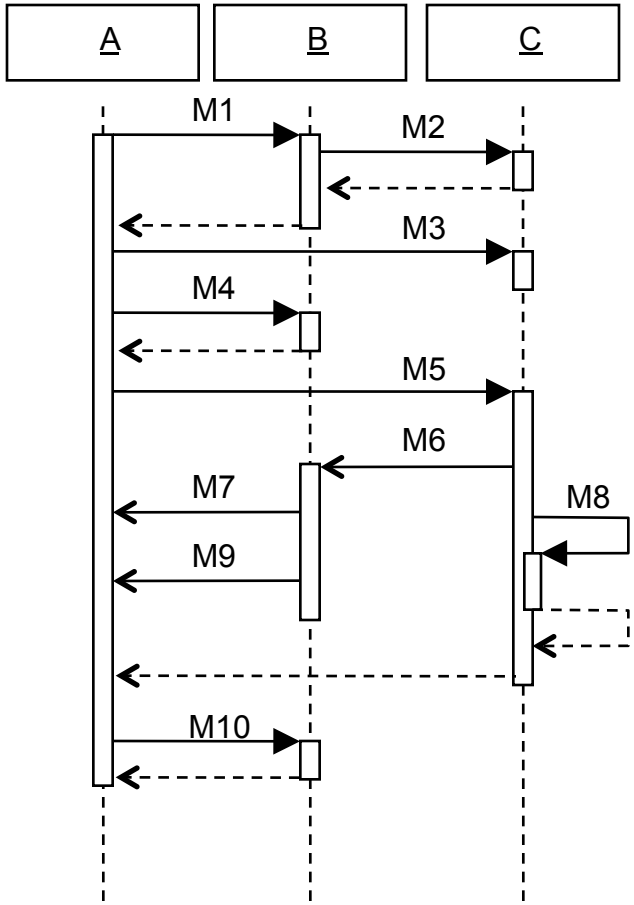
Diagramme de séquence

- Comme les DCO, comporte :
  - des objets dans une situation donnée (instances)
  - les messages échangés entre les objets

- A la différence des DCO :
  - l'accent est mis sur la communication, au détriment de la structure spatiale
  - chaque objet est représenté par une barre verticale (sa ligne de vie)
  - le temps s'écoule de haut en bas, de sorte que la numérotation des messages est optionnelle

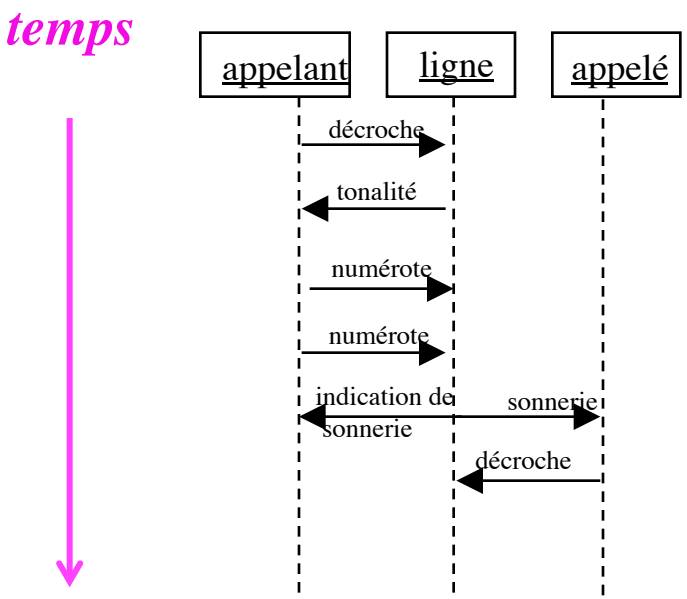
Comparaison DCO/DSE



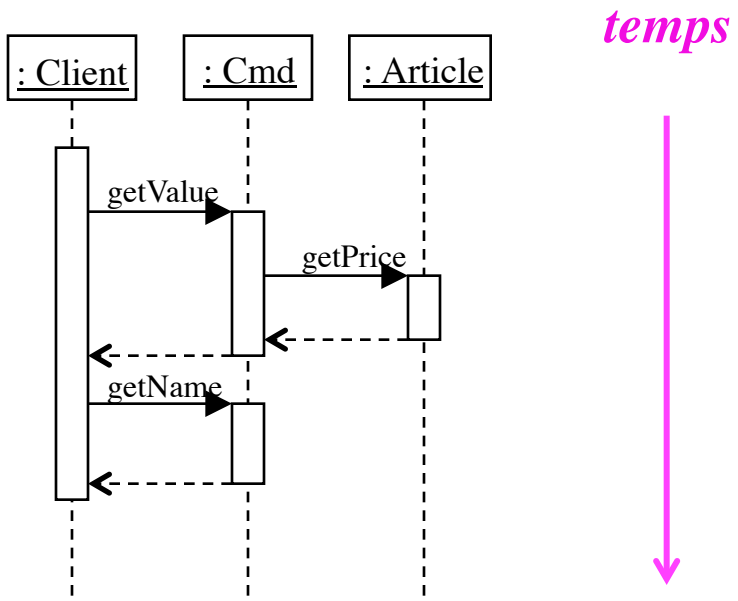


Notation : *niveaux de détails*

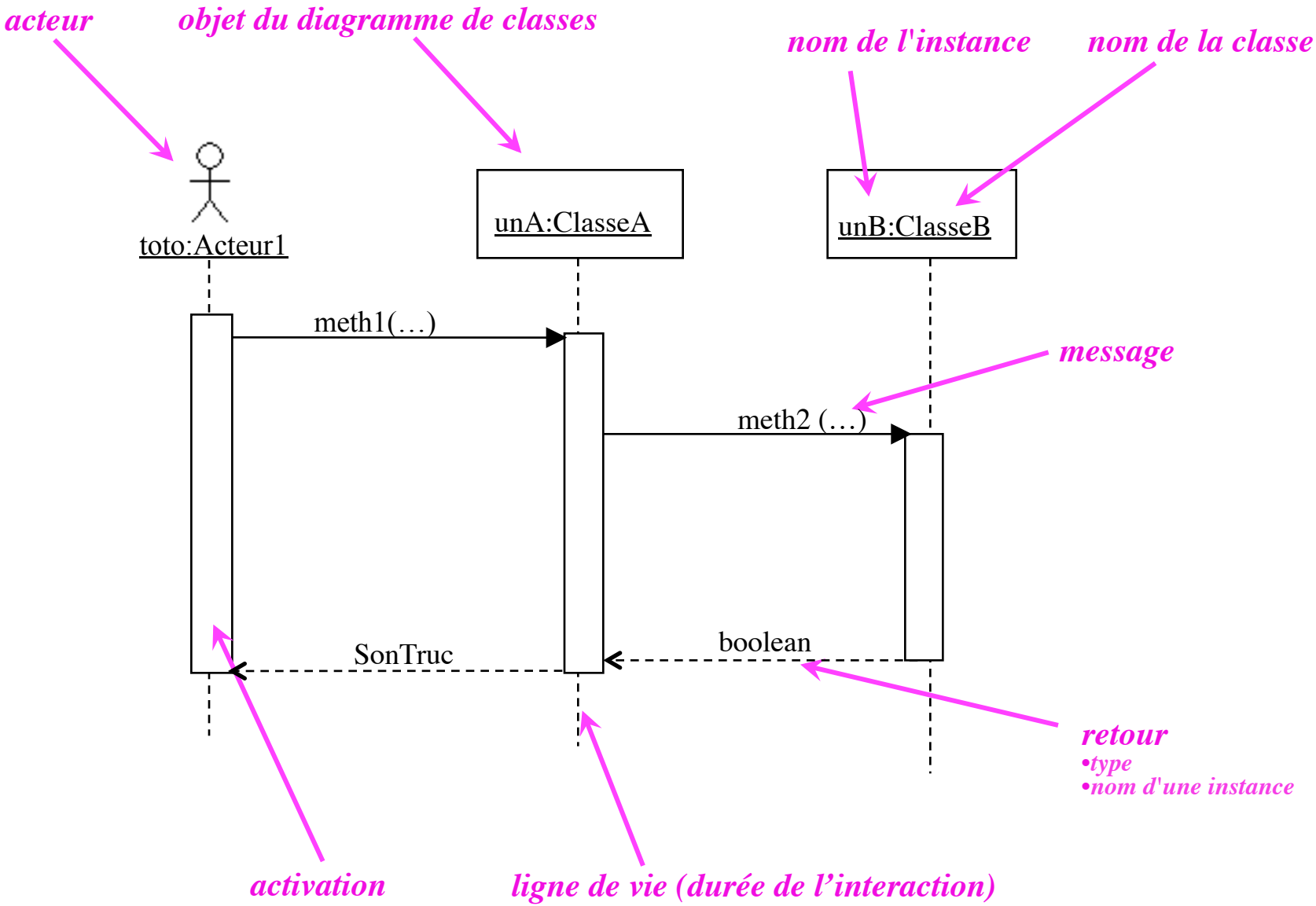
1er niveau :
documentation
du cas d'utilisation



2ème niveau :
représentation précise
des interactions entre objets



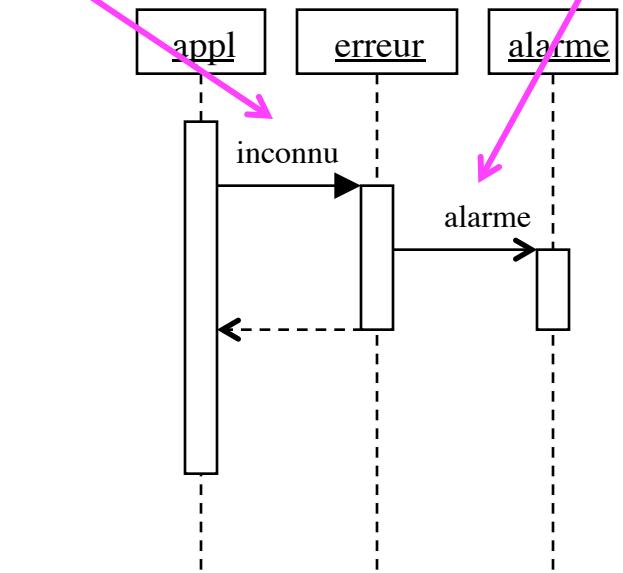
Notation : *principes généraux*



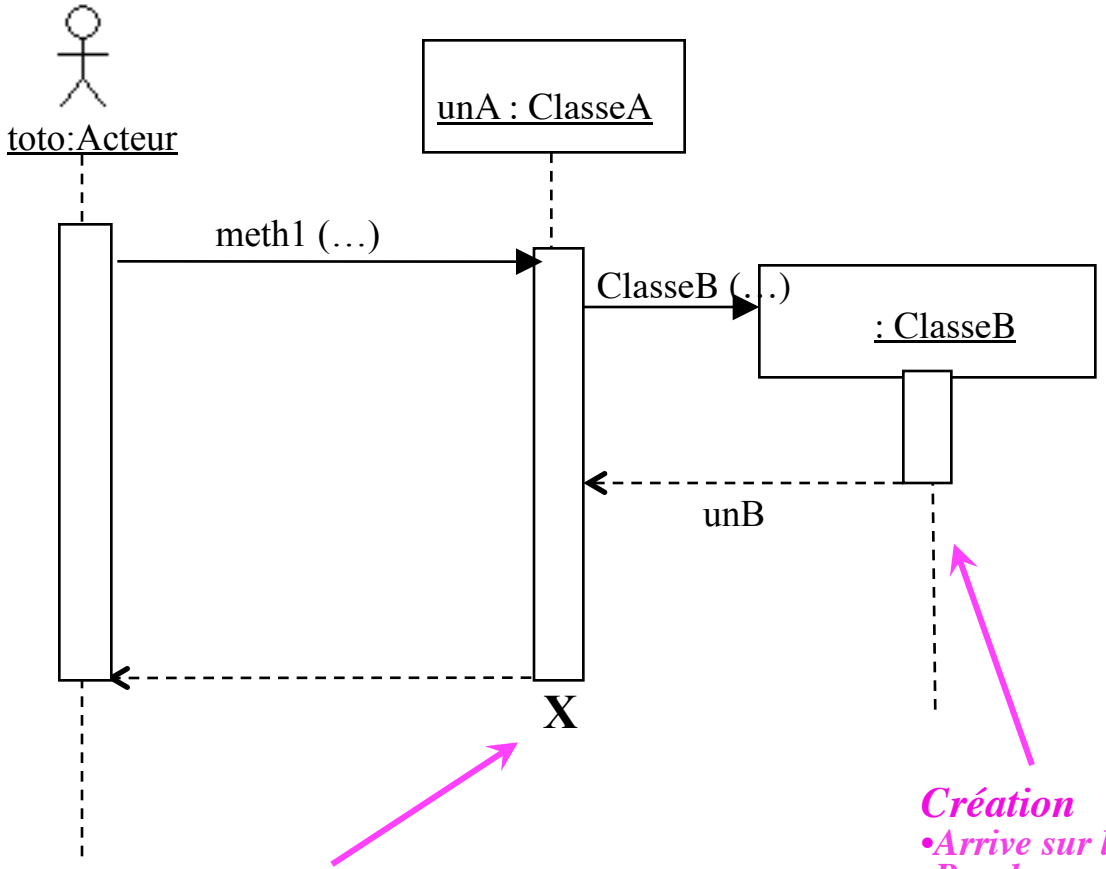
Notation : *messages synchrone/asynchrone*

message synchrone **:**
émetteur bloqué, attend retour

message asynchrone **:**
émetteur non bloqué,
continue ses traitements

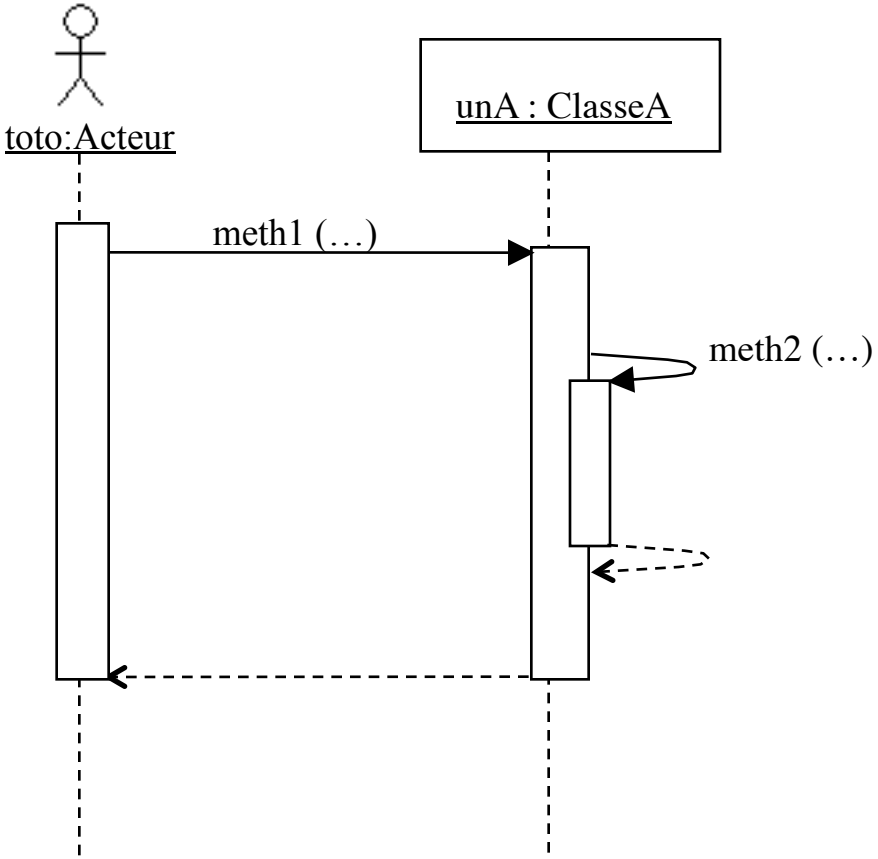


Notation : *création, destruction*

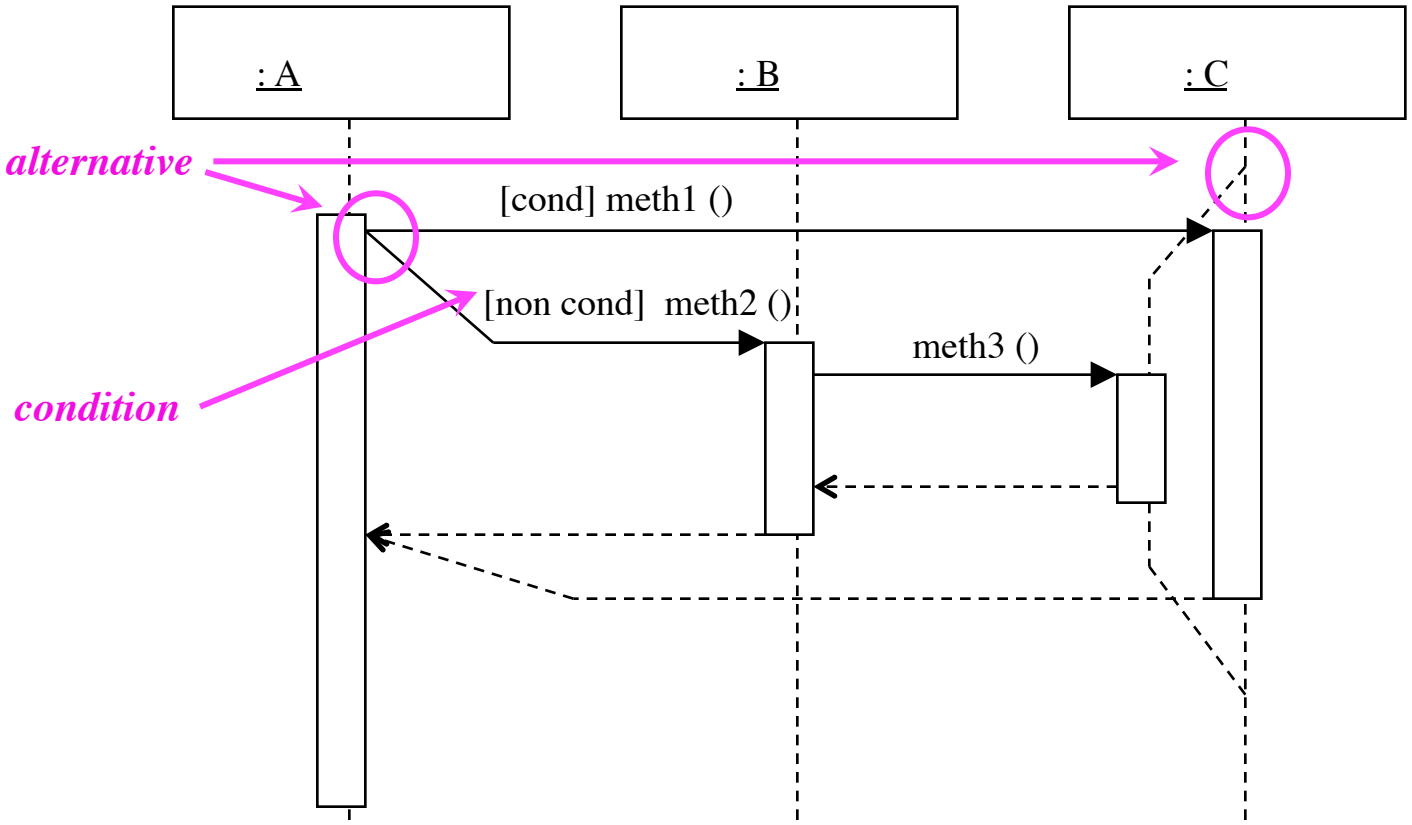


Notation : *message réflexif*

Un objet peut s'envoyer un message à lui-même



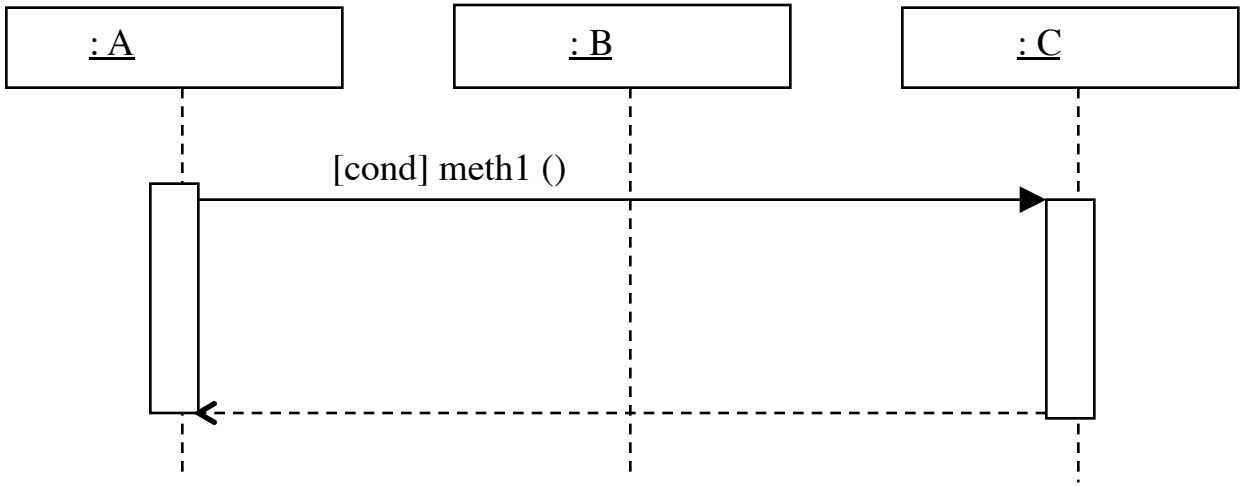
Notation : *alternative 1*



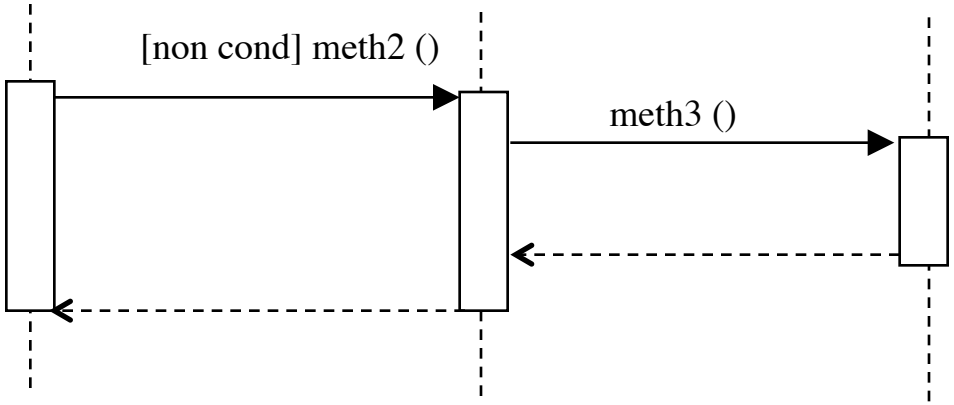
On peut aussi faire deux diagrammes de séquences correspondant aux deux scénarios
→ meilleure lisibilité

Notation : *alternative 2*

Cas cond :

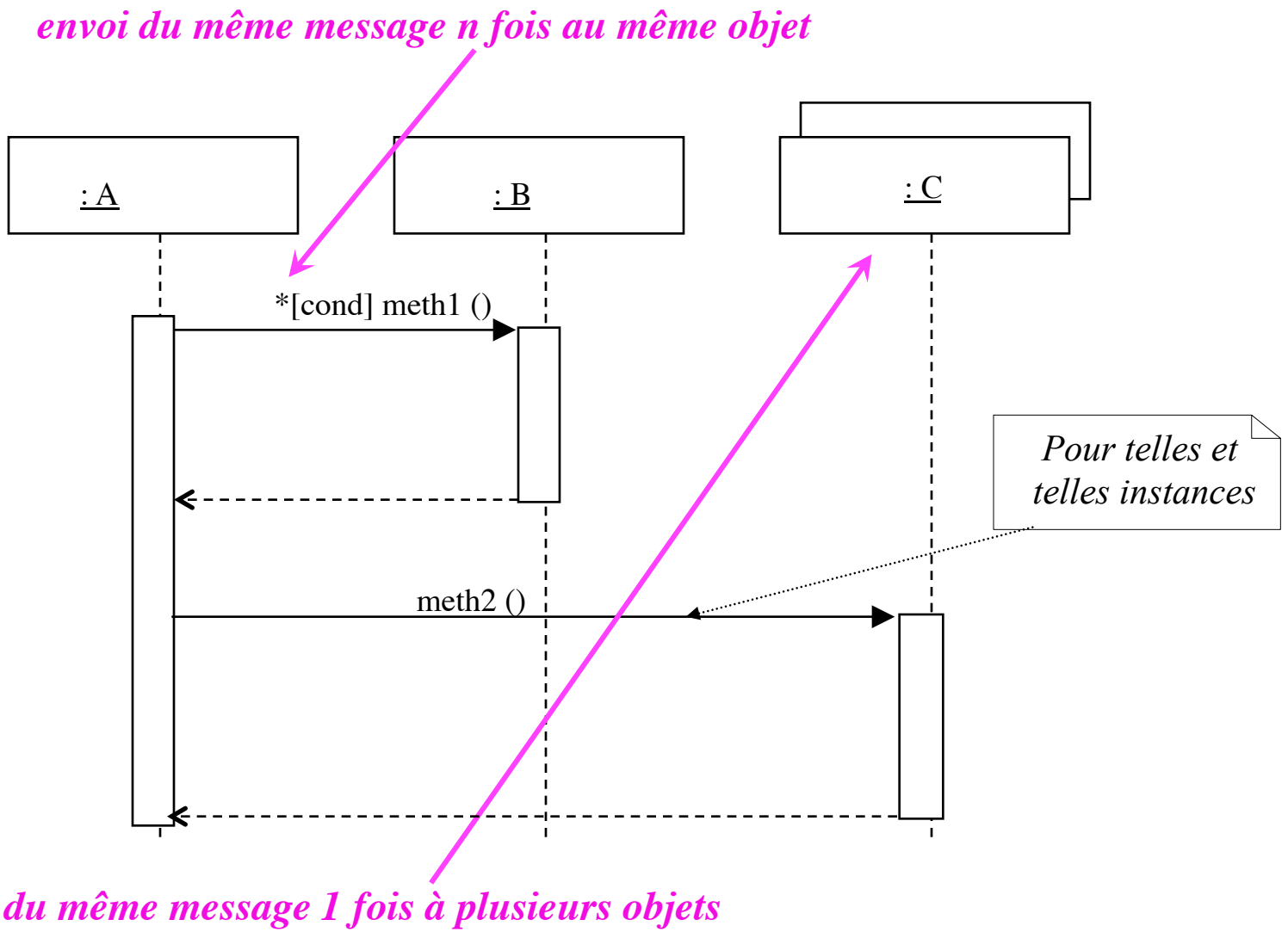


Cas non cond :

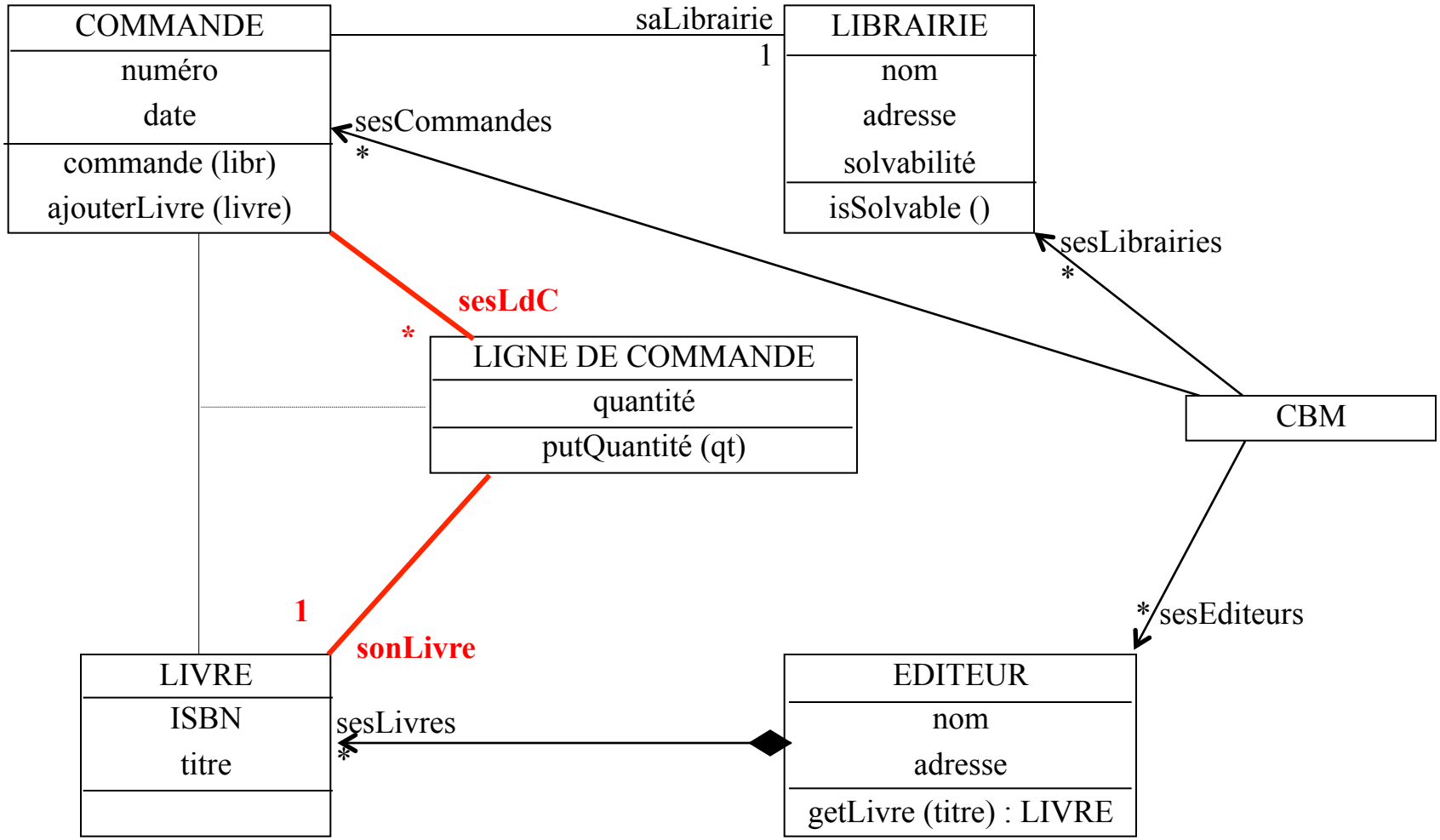


Deux diagrammes de séquences correspondant aux deux scénarios

Notation : *répétition*



Exemple de la CBM : *Le Diag. de Classes*

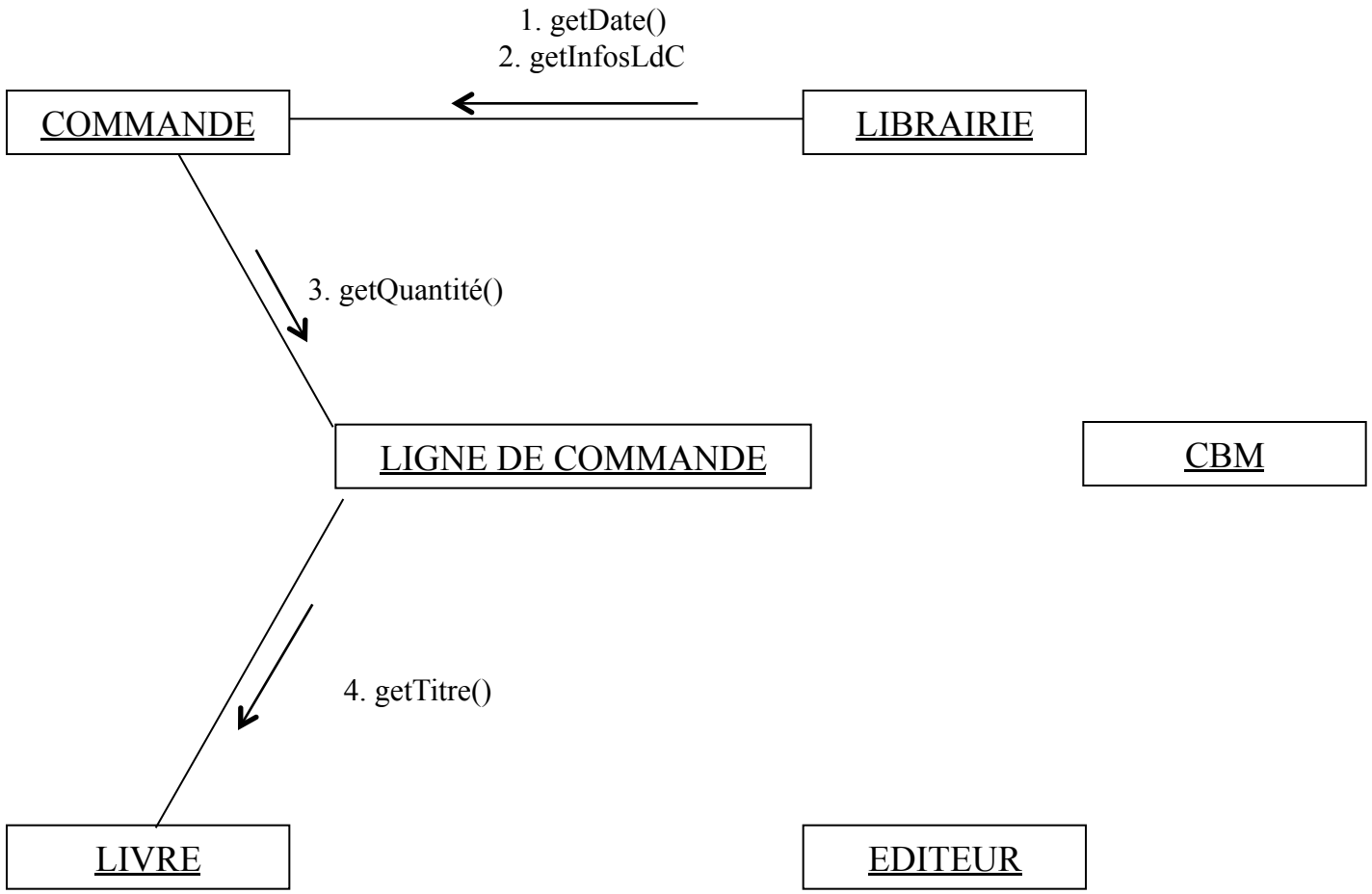


Un cas d'utilisation

- *"Communiquer les détails de toutes les commandes d'une librairie donnée"*

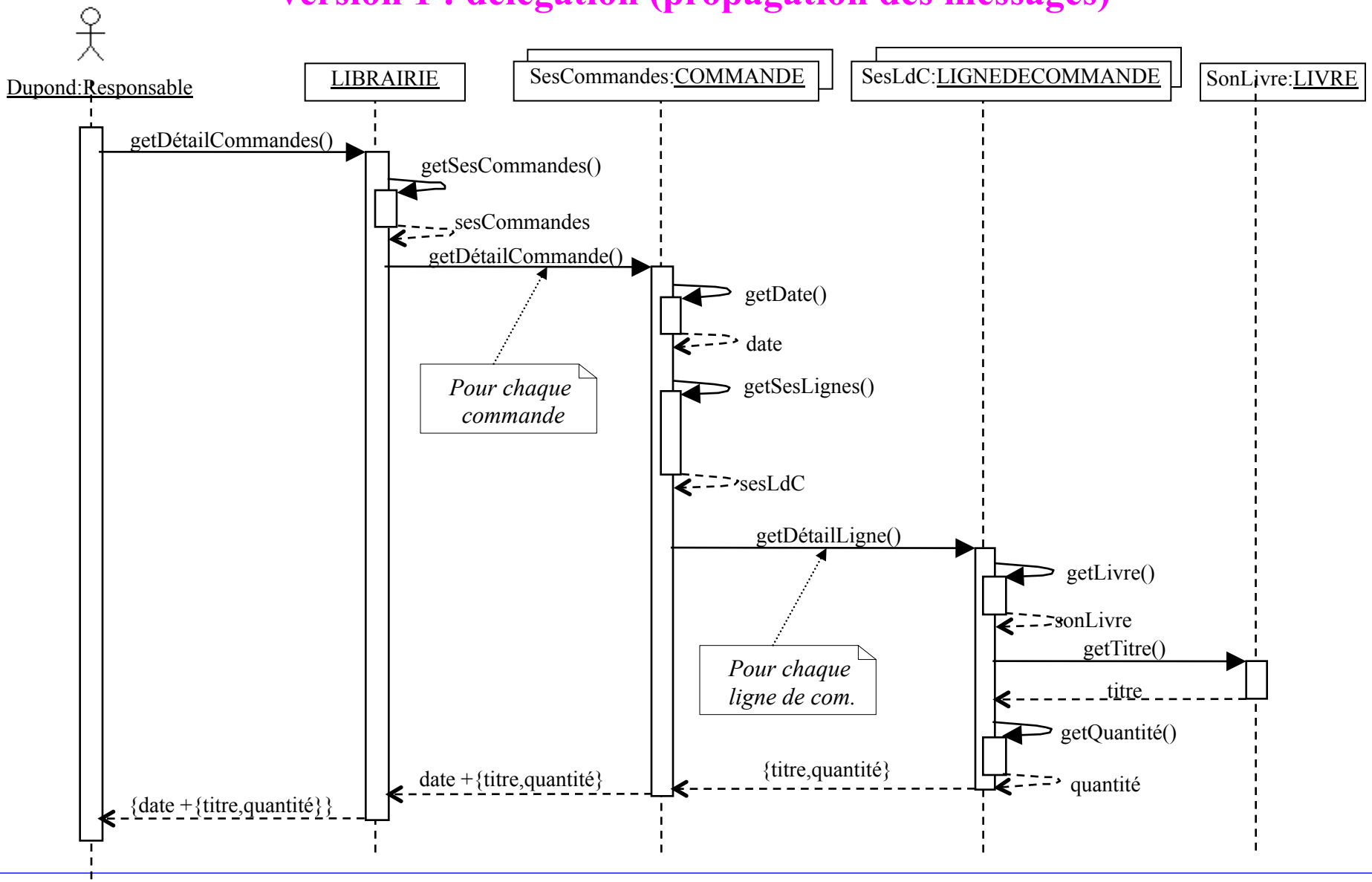
- Méthode ***getDétailCommandes()*** de la classe LIBRAIRIE

Le DCO correspondant

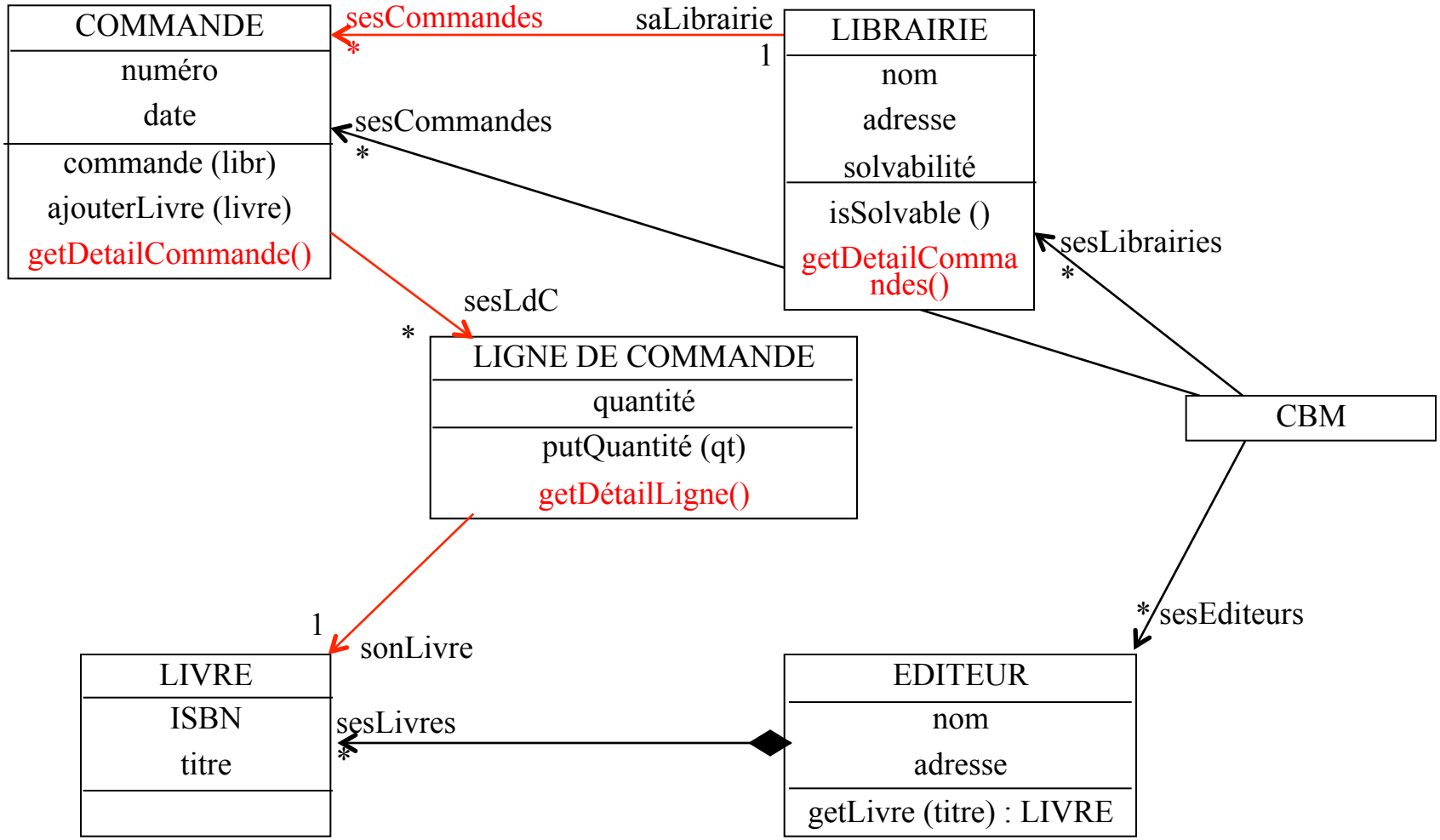


Un DSE : *getDetailsCommandes V1*

version 1 : délégation (propagation des messages)

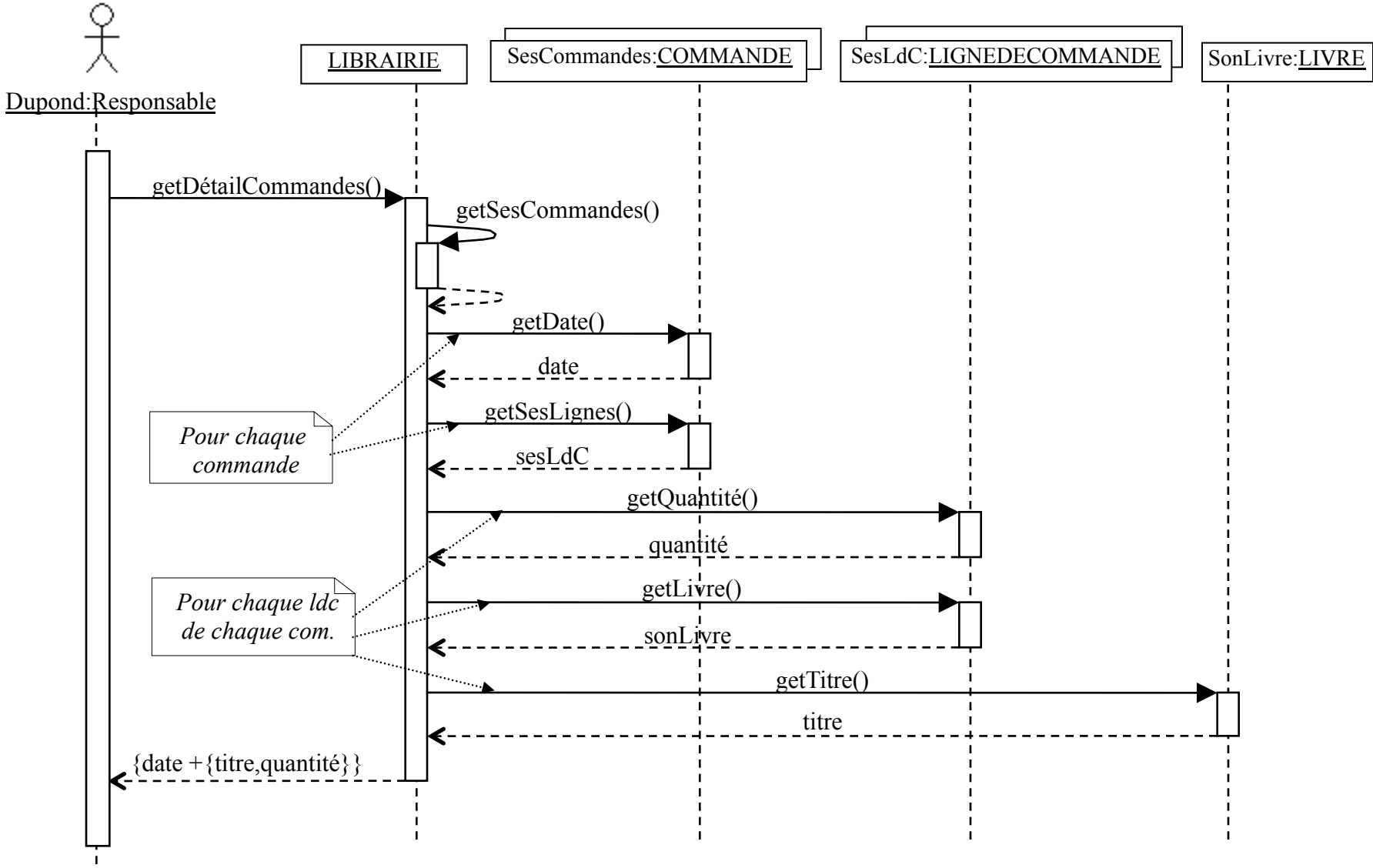


Le DCL : *nouvelle version (délégation)*

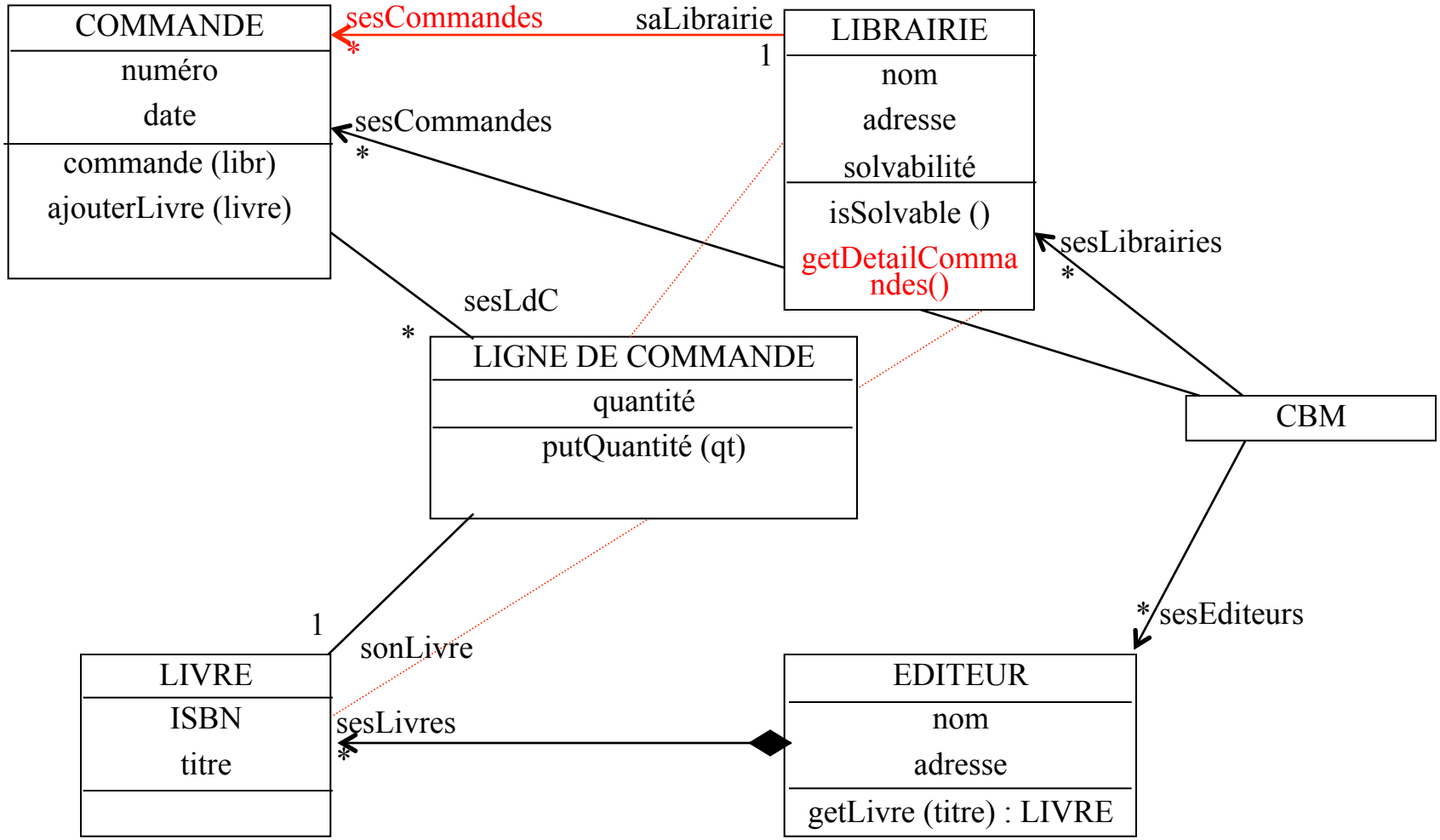


Un DSE : *getDetailsCommandes V2*

version 2 : supervision (un objet envoie tous les messages)



Le DCL : *nouvelle version (supervision)*



4. Vue dynamique

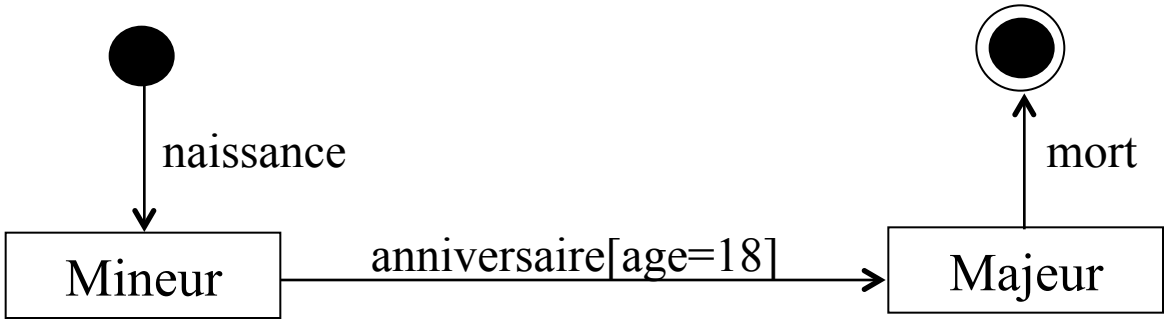
Diagramme état/transition (DET)

Diagramme d'activités

Diagramme états/transitions

- Rôle :
  - Vue synthétique du fonctionnement dynamique d'un objet
  - Décrit le comportement d'un objet tout au long de son cycle de vie
  - Décrit tous les états possibles d'un **unique** objet à travers l'**ensemble** des cas d'utilisation dans lequel il est impliqué
  - On ne s'intéresse qu'aux objets qui ont un comportement complexe

- Exemple - DET d'un citoyen :



Composition

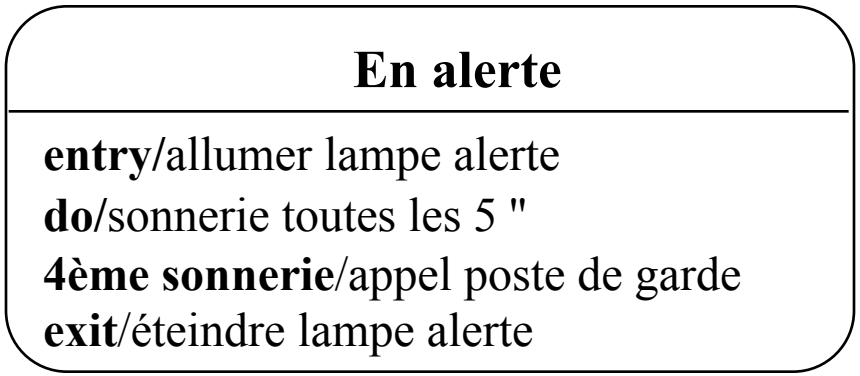
Comporte 2 types d'informations :

- des états :
  - initial
  - final
  - intermédiaires - ex : mineur et majeur
- des transitions
  - induisant un changement d'état
  - ex : naissance, anniversaire…

État (1)

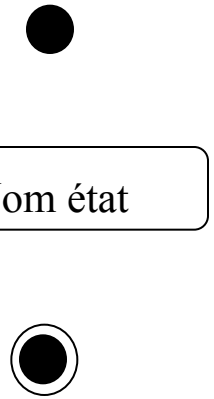
- Un état correspond à la manière d'être d'un objet pendant un intervalle de temps plus ou moins long.
- Un état se compose de plusieurs parties :
  - le nom
  - l'activité attachée à cet état
  - les actions réalisées pendant cet état

- Exemple :



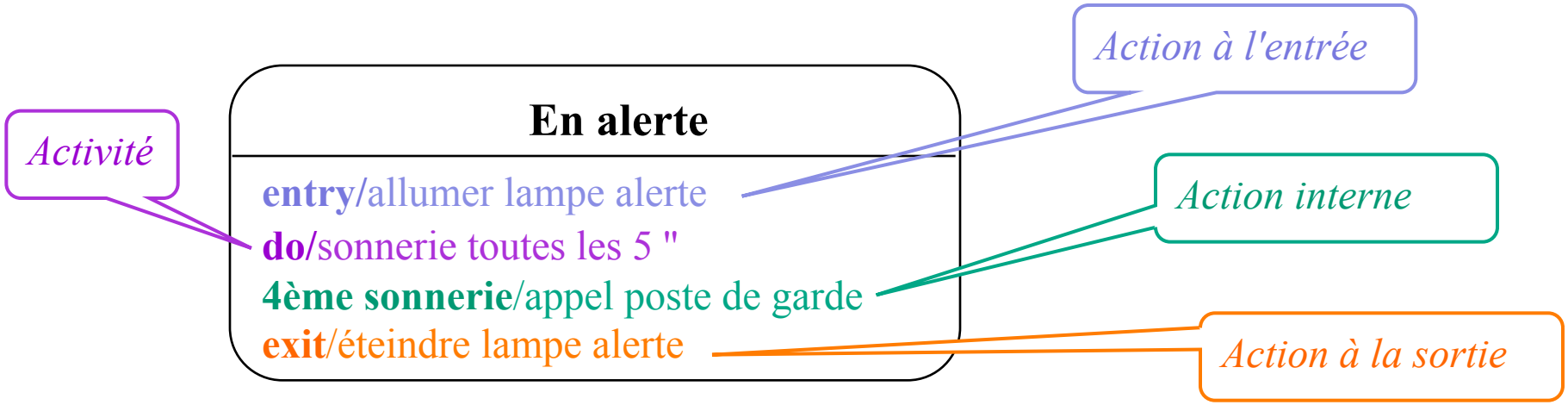
État (2)

- DET : Automate **déterministe** → 3 types d'états
  - état initial (1 et 1 seul)
  - état intermédiaire (plusieurs possibles)
  - état final (aucun ou plusieurs possibles)



Action

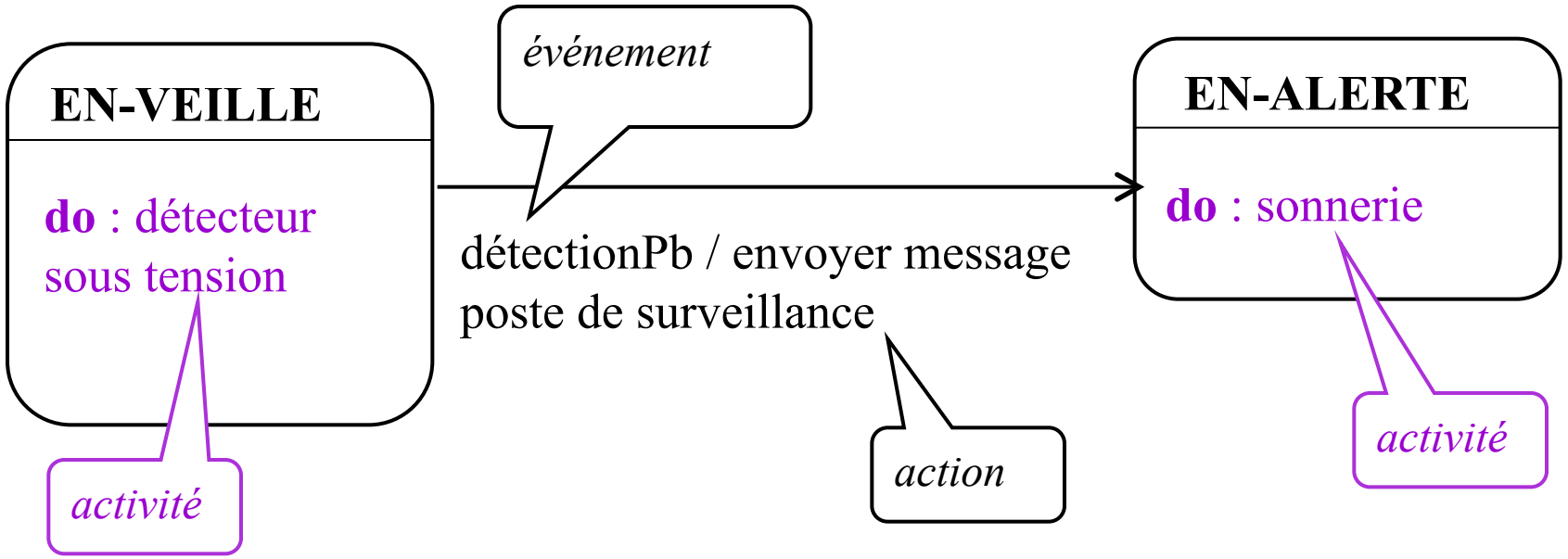
- opération instantanée (durée négligeable) toujours intégralement réalisée → méthode de la classe
- exécutée :
  - lors d'une transition — $event_i$ / $action_i$
  - à l'entrée dans un état — **entry** / $action_i$
  - à la sortie d'un état — **exit** / $action_i$
  - interne, sans changer d'état — $event_i$ / $action_i$



Activité

- opération qui nécessite un certain temps d'exécution
- peut être interrompue à chaque instant
- associée à un état ($état_i$ ≡ "en train de faire ceci")
- exécutée :

  entre l'entrée et la sortie de l'état **do** : $activité_i$



Transition (1)

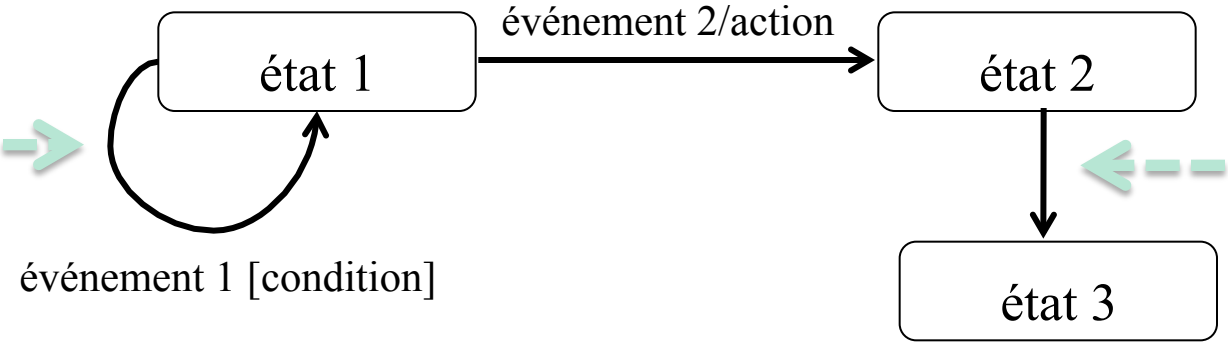
- Transition : passage **unidirectionnel** et instantané d'un état (source) dans un autre état (cible)
- Déclenchée :
  - par un **événement**
  - automatiquement à la fin d'une ***activité*** (transition automatique)
- Syntaxe :

  <NomÉvénement> [<Garde (ou contrainte)>]/<NomAction>
  - Garde : condition booléenne qui valide ou non le déclenchement d'une transition lors de l'occurrence d'un évènement
  - Événement : provoque le changement d'état (il existe aussi des événements internes à un état, dont l'action associée ne provoque pas de changement d'état)
  - Action : opération réalisée lors du changement d'état

Transition (2)

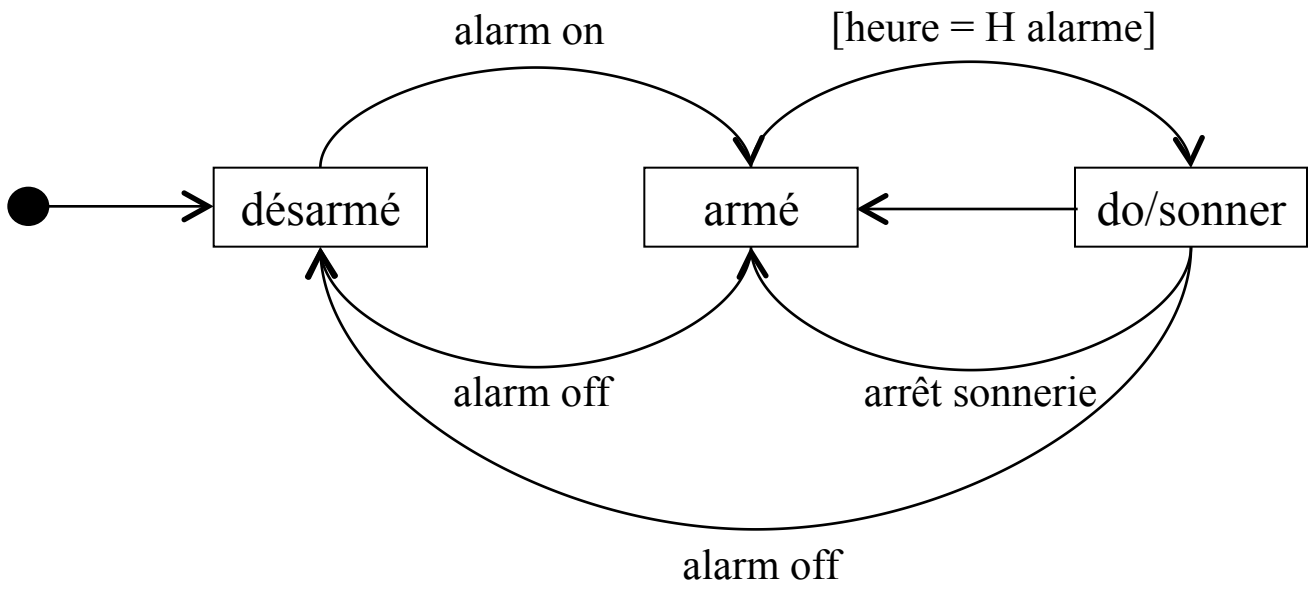
- Transition automatique
  - lorsque qu'il n'y a pas de nom d'événement sur une transition, il est sous-entendu que la transition aura lieu dès la fin de l'activité.

- Auto-transition : transition d'un état vers lui-même

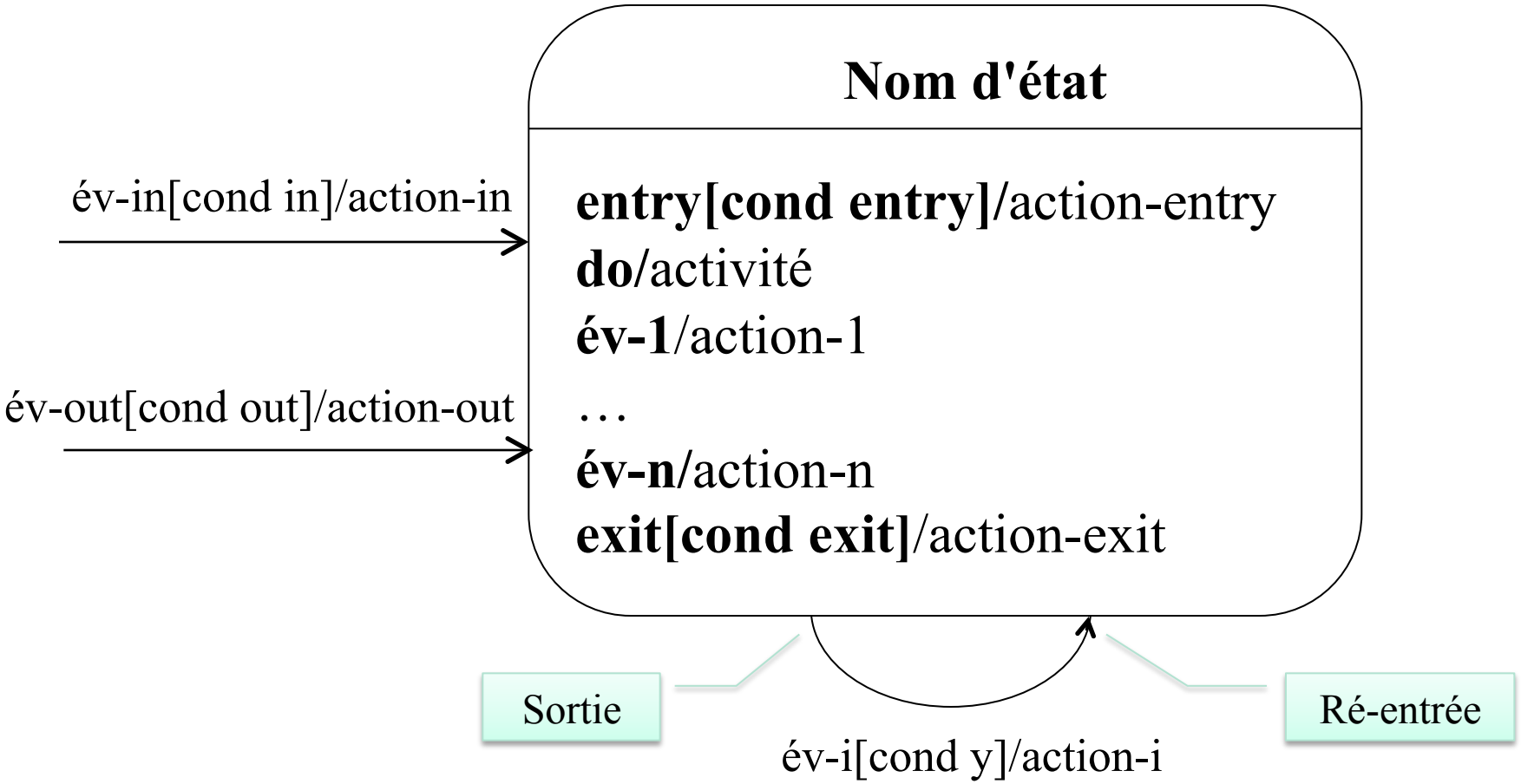


Exemple : *Le réveil matin*

- Réveil
  - Heure d'alarme : on suppose que l'heure est fixée quand on met l'alarme sur on
  - Trois boutons : alarm on/off, arrêt sonnerie, réglage alarme (événement interne)
- Événements : Appui sur un bouton



Forme générale d'un état



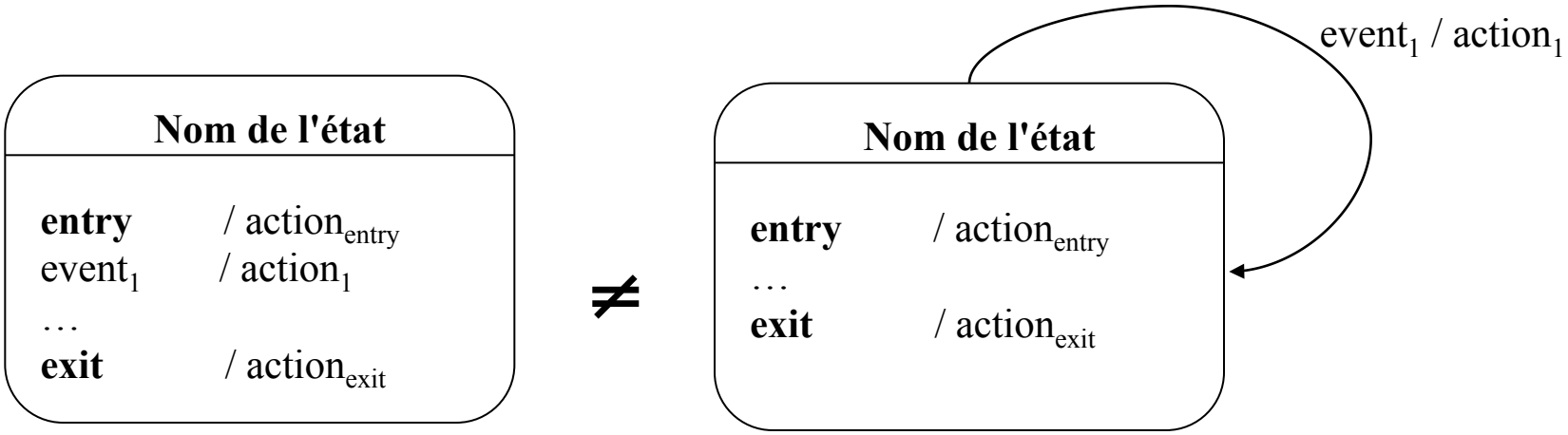
Ordonnancement des actions

- En entrée
  - Action sur la transition d'entrée
  - Action d'entrée
  - Activité associée à l'état
- En interne
  - Interruption de l'activité en cours (contexte sauvé)
  - Action interne
  - Reprise de l'activité
- En sortie
  - Interruption de l'activité en cours (contexte perdu)
  - Action de sortie
  - Action sur la transition de sortie
- Auto-transition
  - Interruption de l'activité en cours (contexte perdu)
  - Action de sortie
  - Action sur l'auto-transition
  - Action d'entrée
  - Activité associée à l'état

Auto-transition / Action interne

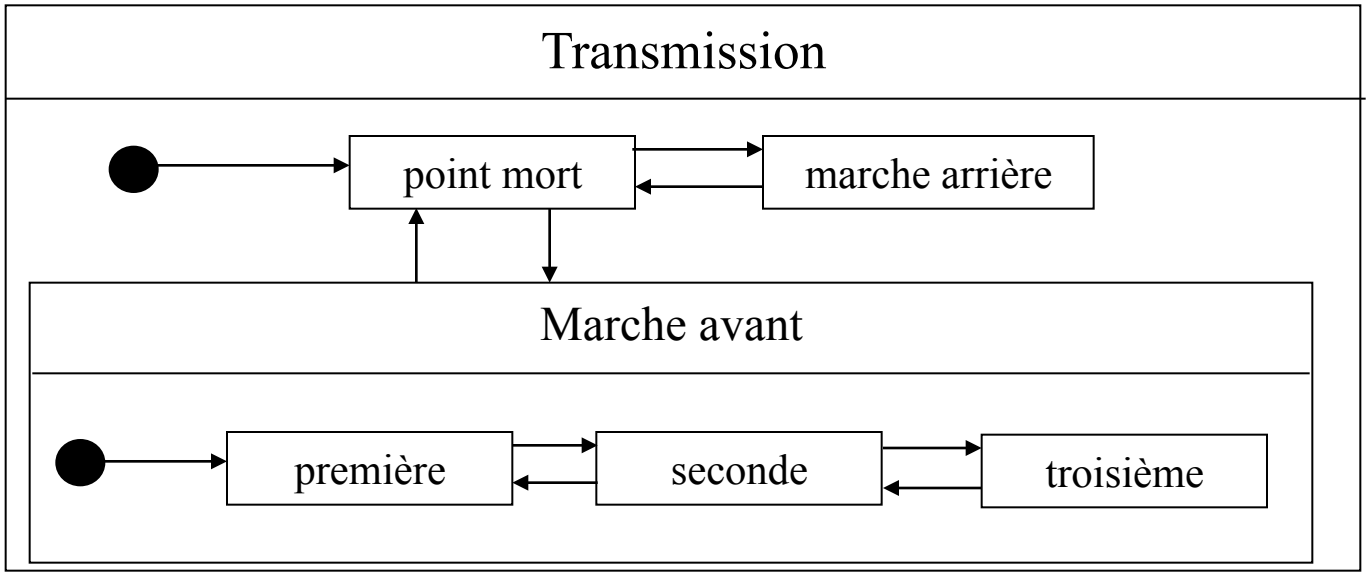
- Action interne, le contexte de l'activité est préservé (on ne sort pas de l'état)

- Auto-transition : le contexte est réinitialisé (on sort et on re-rentre dans l'état)

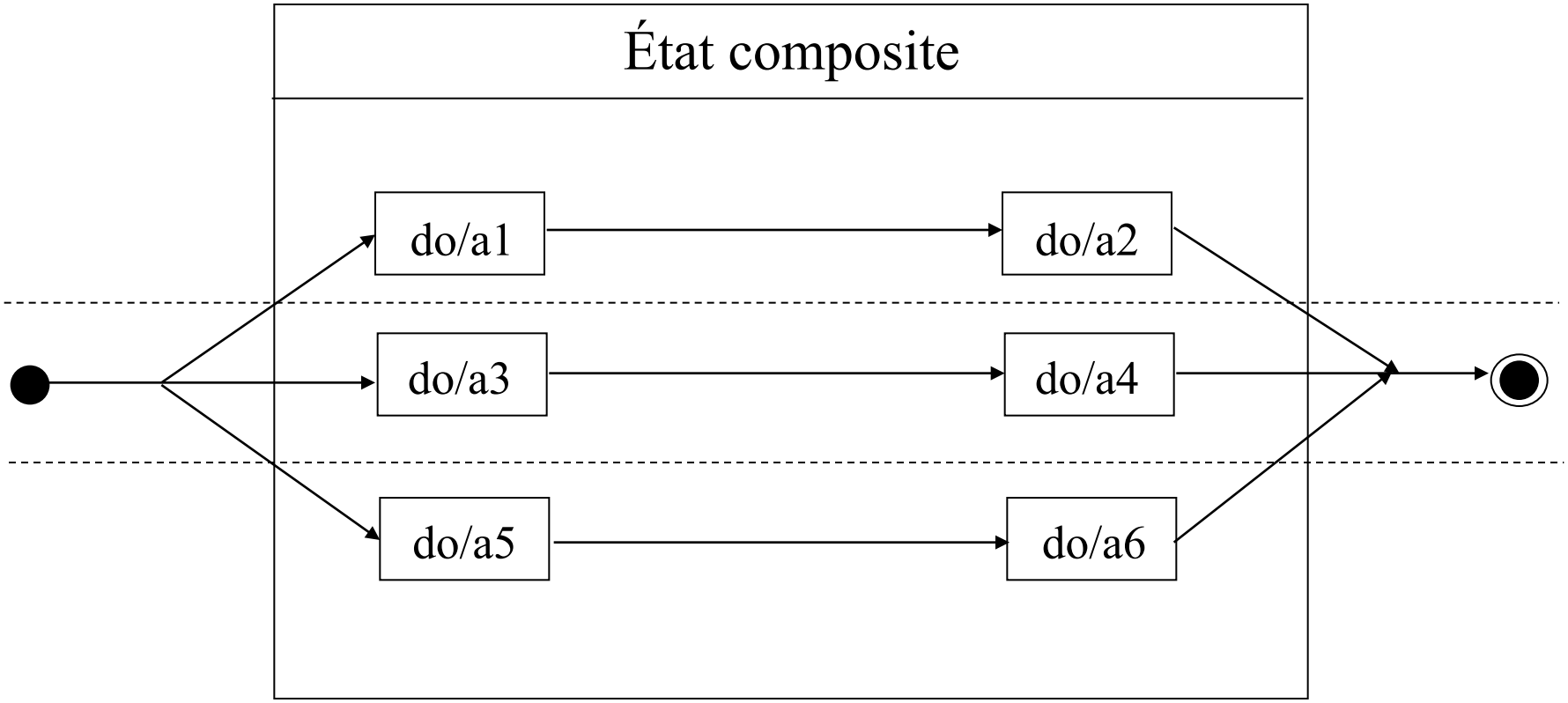


Diagrammes hiérarchisés

- Permettent de structurer les DET complexes
- Factorisation des actions et activités
- Exemple :



Parallélisme et synchronisation



a1//a3//a5 - a2 après a1 - a4 après a3 - a6 après a5 - état final après a2, a4 et a6

Diagramme d'activités

- Cas particulier de DET, dans lequel à chaque état correspond une activité constituant un élément d'une tâche globale à réaliser
- Mise en évidence des contraintes de séquentialité et de parallélisme.
- Exemple - *Faire un café* :