Introduction à UML

Jean-Philippe Domenger

Université Bordeaux I & LaBRI

Objectifs de ce cours

- Présenter la notation UML
  - Historique
  - Les motivations
  - Les différents diagrammes

Historique

La floraison des méthodes objet pénalise la communication et la réutilisation des solutions.

Constat sur l 'objet

- L 'objet à 30 ans (Simula 1967)
  - atteint sa maturité
  - augmentation des logiciels objets
  - stabilité des concepts
- Concepts complexes
  - Approche objet plus délicate que l 'approche fonctionnelle
  - confusion entre langage et concepts

Constat sur l 'objet

- Prolifération des méthodes objets
  - une cinquantaine dans les 10 dernières années
  - Aucune ne s 'est imposée
    - ⇨ frein à l 'essor des technologies objets
  - Difficultés d 'échanger et de comparer des découpes objets
  - Apparition de modèles complémentaires

Que faut il !

- Un langage
  - définitions non ambiguë des concepts
  - extensible et générique
  - Couverture complète des aspects du développement allant de l ’analyse jusqu’au déploiement.
    - Stabilisation par convergence
  - Indépendant des langages de programmation objet

Que faudrait-il ?

- Un processus: formalisation et attribution des phases du développement
  - Description des phases
  - Enchaînement des phases
  - Attribution des phases
- Difficile à standardiser
  - Freins: Domaine, Entreprise, Personne

Convergence des notations

- Des notations ont tout de même émergées
  - 1980 - OMT (Rumbaugh) maturité en 1991
    - Modèle statique, Modèle dynamique, Modèle fonctionnel
  - 1980 - OOD (Booch) maturité en 1991
    - Modèle statique, Modèle dynamique
  - 1990 - OOSE (Jacobson) maturité en 1992
    - Modèle des besoins, Modèle d 'analyse, Modèle de conception, Modèle d'implémentation, Modèle de test

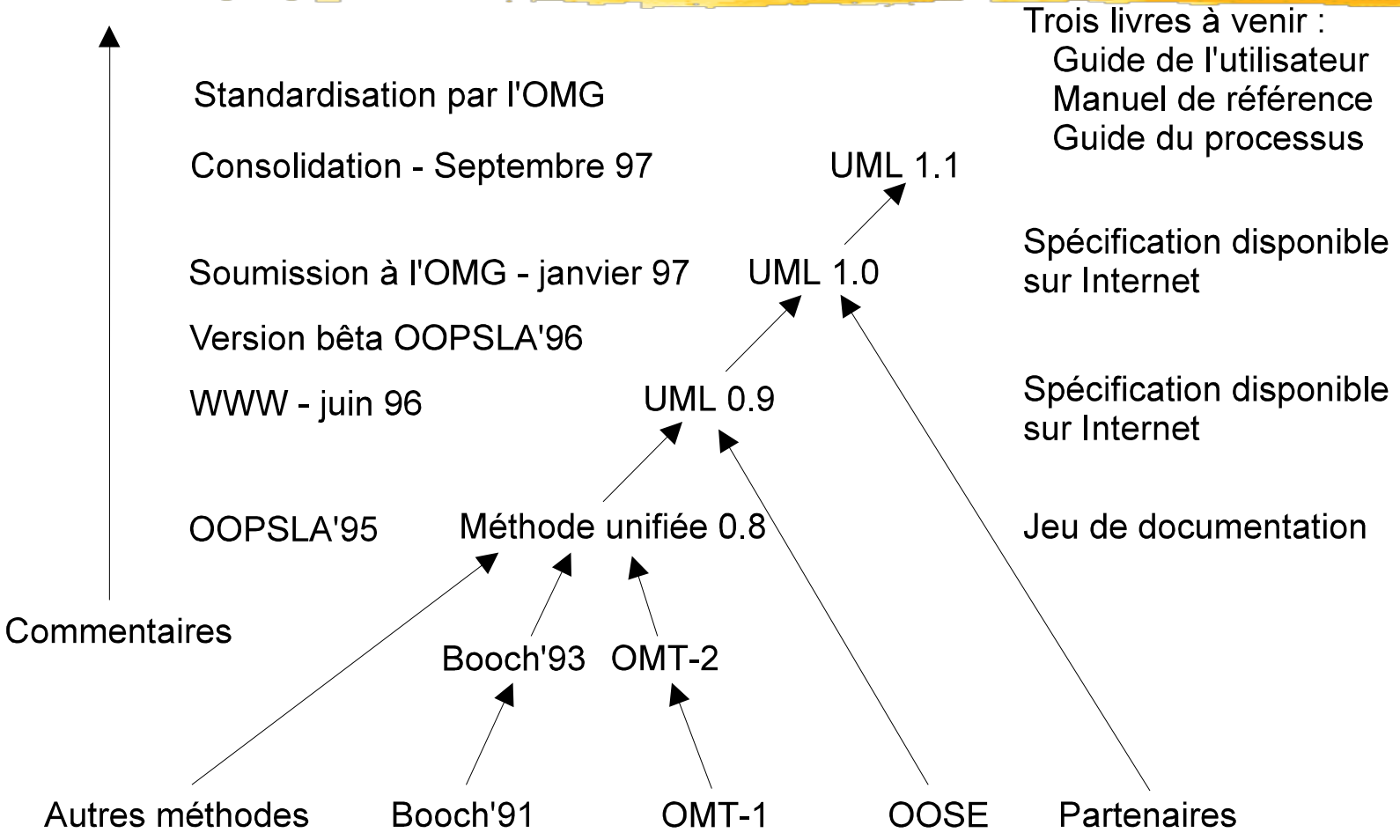
Convergence des notations

- Des notations ont tout de même émergées
  - 1994- FUSION (Coleman) intégration de certains modèles provenant d ’autre notations.
    - Un premier pas vers UML

L’ Emergence d’ UML

- 1995 *Unified Method* V0.8
- 1996 UML V0.91 (*The Unified Modeling Language for Object-Oriented Development*)
- 1997 UML 1.0 est soumise à l’OMG
- 1997 UML 1.1 est acceptée par l’OMG
- 1999 UML 1.3

Emergence d'UML



UML: une convergence

Les contributions

Booch	Catégories et sous-systèmes
Embley	Classes singletons et objets composites
Fusion	Description des opérations, numérotation des messages
Gamma, et al.	*Frameworks*, *patterns*, et notes
Harel	Automates (*Statecharts*)
Jacobson	Cas d'utilisation (*use cases*)
Meyer	Pré- et post-conditions
Odell	Classification dynamique, éclairage sur les événements
OMT	Associations
Shlaer-Mellor	Cycle de vie des objets
Wirfs-Brock	Responsabilités (CRC)

Le consortium UML

- Adhésion des plus grands
  - Rational, Microsoft , HP, Oracle, Sterling, MCI Systemhouse, Unisys, ICON, i-Logix, Intellicorp, IBM, ObjectTime, Platinum, Ptech, Taskon, Reich Technologies, Softeam.
- UML: un enjeu stratégique
  - Les trois fondateurs chez Rational

Motivation

Une documentation pour représenter l ’approche objet.

UML: un langage

- UML: un langage visuel et universel
  - UML est dans le domaine public
  - Notation graphique expressive
  - Basé sur un méta-modèle
    - Formalisation de la sémantique des concepts
      - ⇒ non ambiguë
    - Développement d'outil basé sur le méta-modèle
  - XMI format d 'échange standard de modèle UML.

UML: fédère l 'approche objet

- UML permet de représenter des modèles, pas de les définir.
  - Modèle: Description abstraite d 'un système
    - compréhension du système
    - abstraction du système
    - simulation du système

UML:fédère l 'approche objet

- UML permet de représenter les vues d 'un système.
  - Vue utilisateur:
    - Spécifie les besoins de l 'utilisateur vis à vis du système.
  - Vue logique:
    - identification des entités du domaine,
    - abstraction de ces entités,
    - définition des relations et interactions entre entités
    - répartition des entités

UML:fédère l ’approche objet

- Vue des composants:
  - implémentation de la vue logique
  - contraintes logicielle (langage, bibliothèque)
- Vue des processus:
  - décomposition en processus
  - synchronisation et communications des processus
- Vue de déploiement
  - répartition des processus sur architecture parallèle.

UML:fédère l 'approche objet

- UML: un outil de documentation de l 'approche objet
  - Représentation des phases de développement
    - Analyse, Conception, Implémentation
  - Représentation multi-échelle
    - Raffinement d 'une vue de l 'analyse à l 'implémentation

UML et le processus de développement

- UML est un langage, pas une méthode
  - pas de spécification du processus de développement
- mais UML favorise un processus
  - guidé par les besoins des utilisateurs
  - focalisé sur l ’architecture logicielle
  - itératif et incrémental

En résumé

- Langage visuel basé sur un méta modèle
- Langage utilisable en largeur et en profondeur
- Langage standardisé et consensuel
- Indépendant des langages de programmation
- Utilisable par tous les processus de développement

UML

Les différents diagrammes d ’UML et leur signification.

Les diagrammes d ’UML

- Diagrammes capturant l ’aspect statique du système
  - Diagramme des cas d ’utilisation
  - Diagramme de classe et de paquetage
  - Diagramme d ’objet
  - Diagramme de composants
  - Diagramme de déploiement

Les diagrammes d ’UML

- Diagrammes capturant l ’aspect dynamique du système
  - Diagramme de collaboration
  - Diagramme de séquence
  - Diagramme d ’activité
  - Diagramme d ’état-transition

Les Vues d'UML

- Une vue décrit un aspect statique ou dynamique du système
- A chaque vue sont associés un ou deux types de diagrammes
- Les éléments utilisés dans les vues sont les classes, les composants, les interfaces, les acteurs,…
- Les constructions standard des diagrammes peuvent être étendues par les stéreotypes, les notes et les contraintes

Les Vues d'UML.

structure statique	vue statique	diagramme de classes
		diagramme d'objets
	vue de cas d'utilisation	diagramme de cas d'utilisation
	vue de composants	diagramme de composants
	vue de déploiement	diagramme de déploiement
comportement dynamique	vue d'états	diagramme d'états
	vue d'activités	diagramme d'activités
	vue d'interaction	diagramme de séquence
		diagramme de collaboration

Les cas d ’utilisation (1)

- Les cas d ’utilisation permettent de définir un système en fonction des besoins exprimés par les utilisateurs.
  - Recenser et clarifier les besoins des clients
    - Définition des acteurs (Rôle).
    - Description de leur actions vis à vis du système
    - Délimitation de la frontière du système
    - Définition des actions/réactions du système

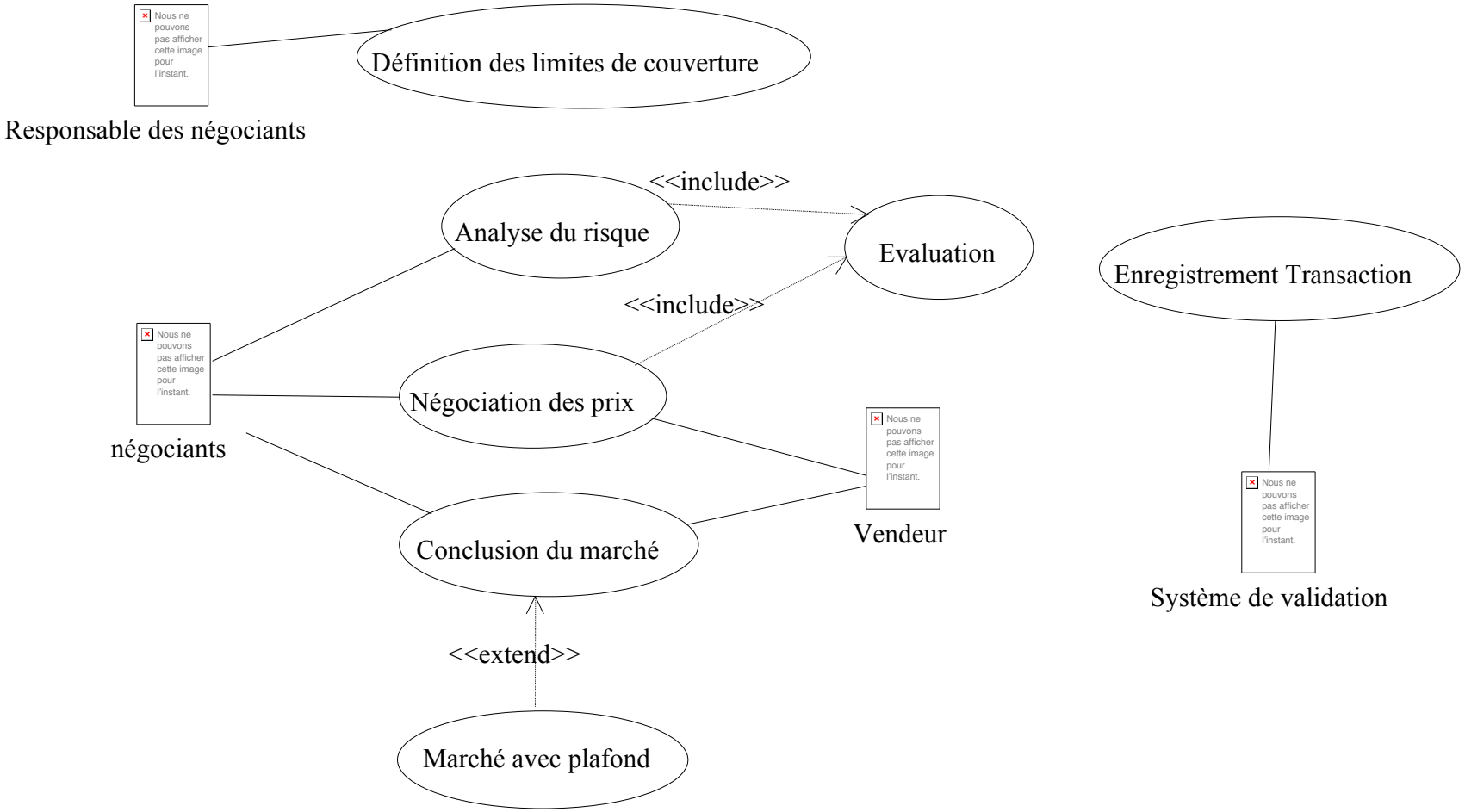
Les cas d ’utilisation (2)

- Un **acteur** est un **rôle** jouer par un utilisateur du système.
  - Plusieurs utilisateurs peuvent tenir le même rôle.
    - Pour une salle des marché, il y a plusieurs personnes qui joue le rôle de négociant
  - Un utilisateur peut avoir plusieurs rôles.
    - Un négociant peut acheter pour son propre compte.
  - Un acteur n ’est pas nécessairement une personne physique, il peut être un système externe

Les cas d ’utilisation (3)

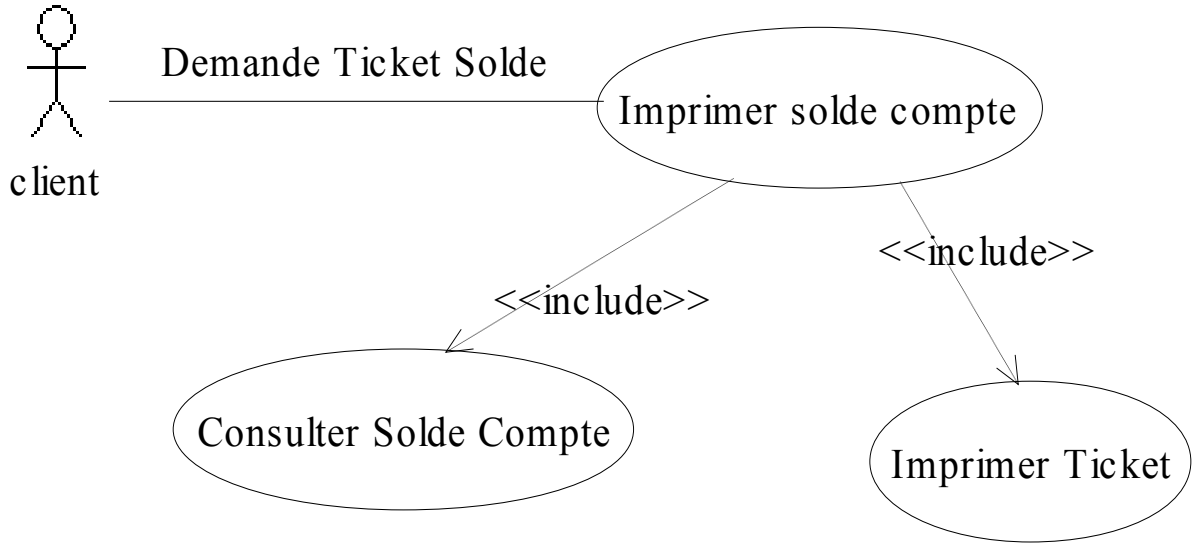
- L ’acteur et le système:
  - L'acteur consulte ou modifie l'état du système.
    - Quelles informations communiquées au système.
  - Le système répond à une action de l ’acteur.
    - Quelles sont les actions de l ’acteur
    - Quelles informations sont communiquées par le système.

Les cas d ’utilisation (4)



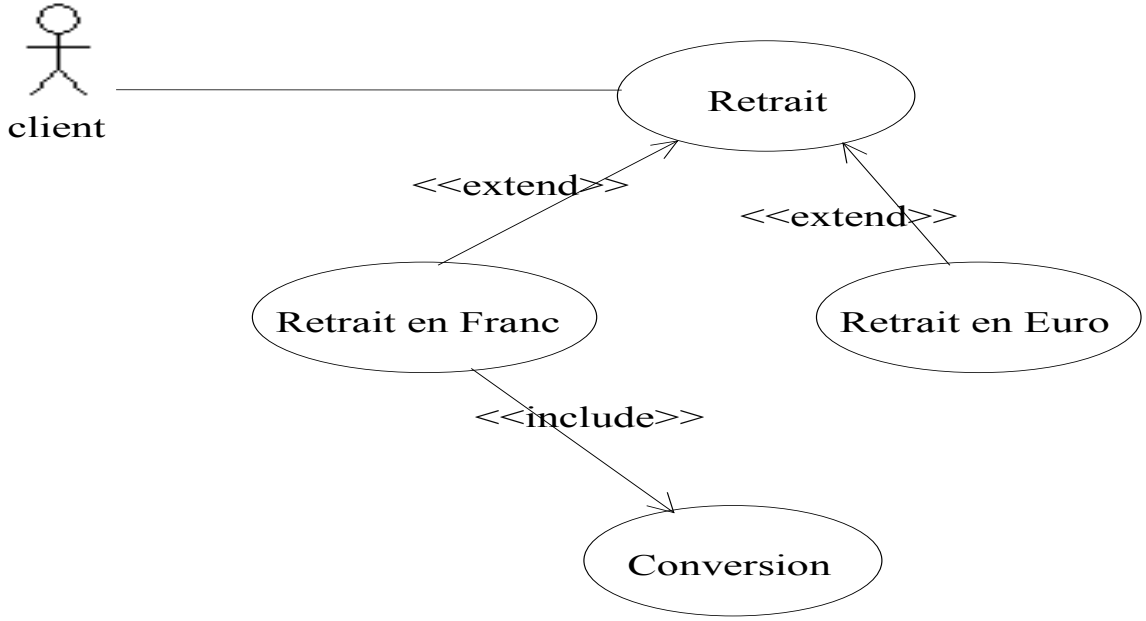
Les cas d 'utilisation (5)

- Relation d 'utilisation: Un cas d'utilisation contient le comportement d'un autre (sous fonction).

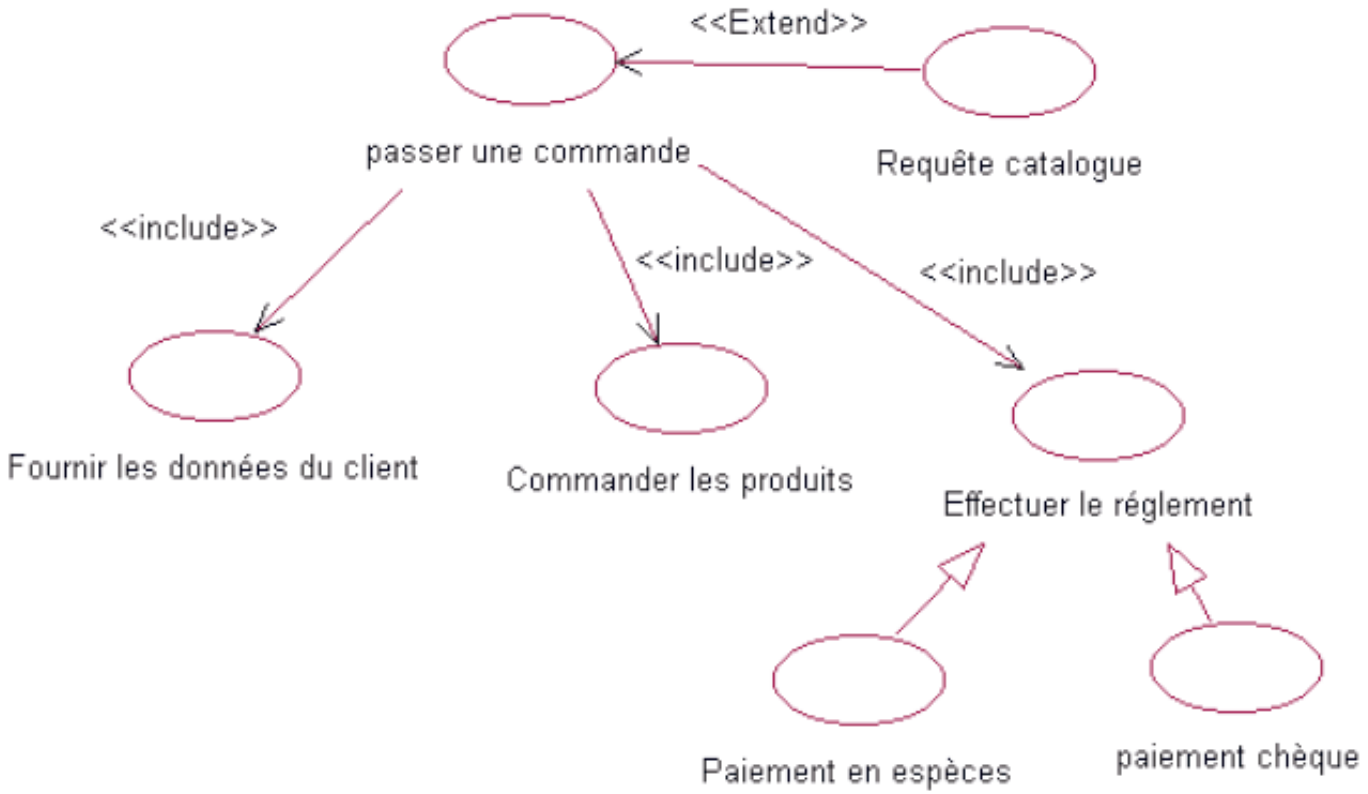


Les cas d ’utilisation (6)

- Relation d ’extension: Un cas d’utilisation précise un autre cas.



Les cas d'utilisations (7)



Les cas d'utilisations(8)

- Le nom du cas d'utilisation
- L'acteur primaire du cas d'utilisation
- Le système concerné par le cas d'utilisation
  - Le niveau du cas d'utilisation
  - Objectif utilisateur
  - Sous-fonction (pour les cas d'utilisation étendant un autre ou destinés à être inclus)
- Les opérations
- Les points d'extension (afin de programmer les conditions)

Les cas d'utilisations (9)

Cas d'utilisation		Retrait d'espèces
Acteur primaire		Agent du guichet de la banque
Système		Système informatique de la banque
Niveau		Objectif utilisateur
Opérations		
	1	Saisir le numéro de compte du client
	2	Vérifier le solde
	3	Saisir le montant du retrait
	4	Attendre l'argent
	5	Terminer la transaction
Extensions		
	2.A	Le solde est insuffisant et le client a droit à un découvert
	2.A.1	Continuer

Démarche guidée par les besoins

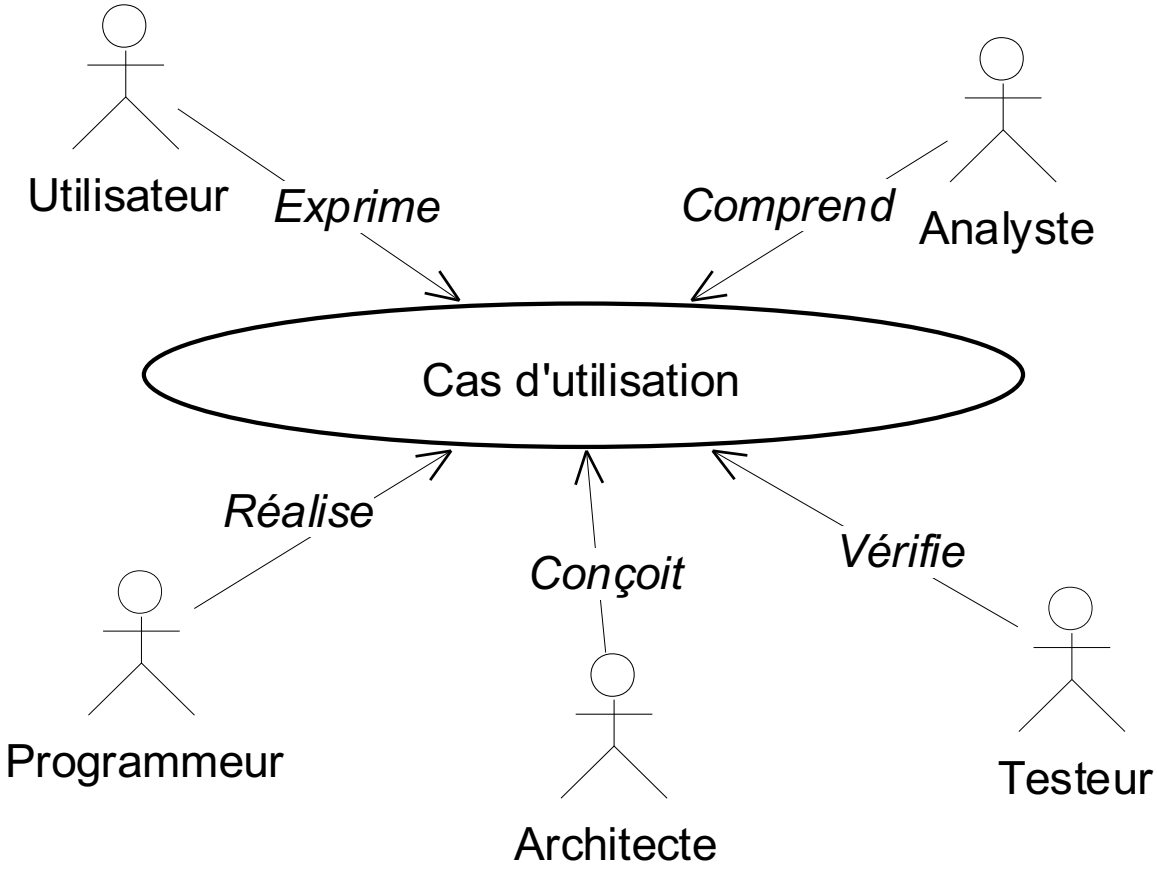


Diagramme d 'objet (1)

- Ce diagramme présente
  - l 'identité des objets
  - leur classe (qui peut être indéfinie)
  - leur état (pas nécessairement la valeur de leur variable d 'instance)
  - les liens entre les objets
  - une qualification des liens

Diagramme d 'objet (2): exemple

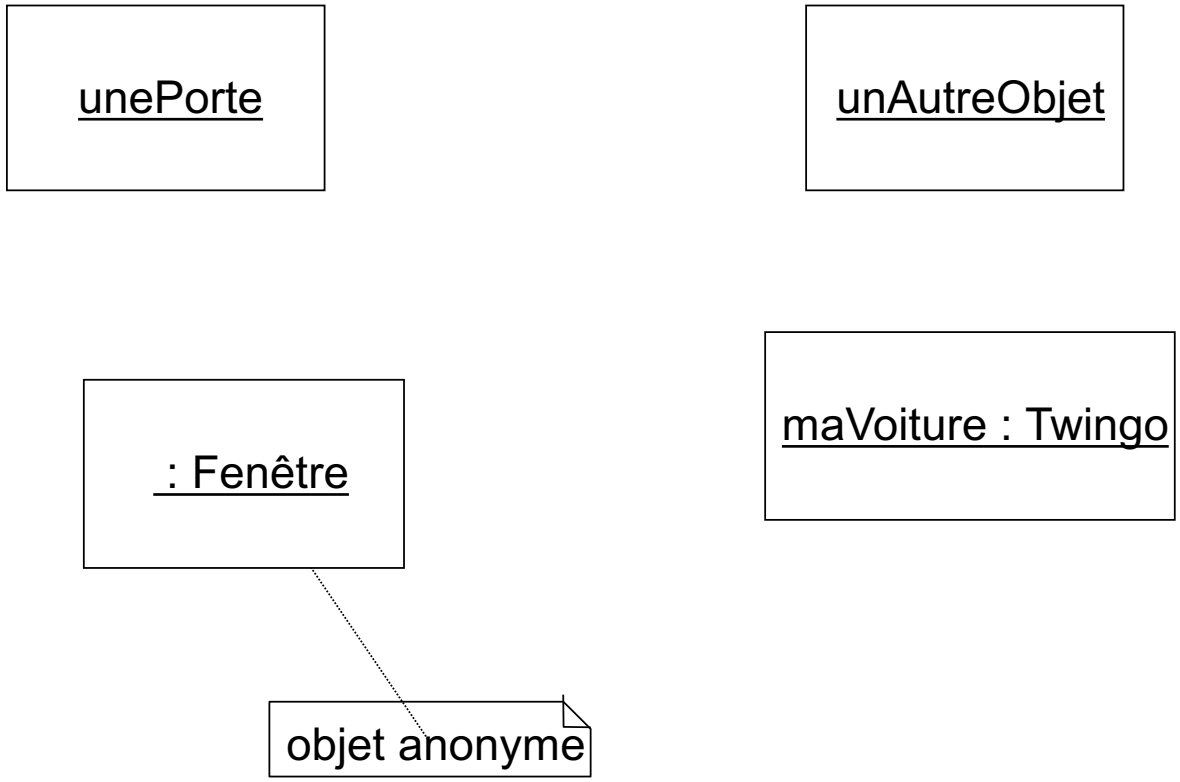


Diagramme d 'objet (3): exemple

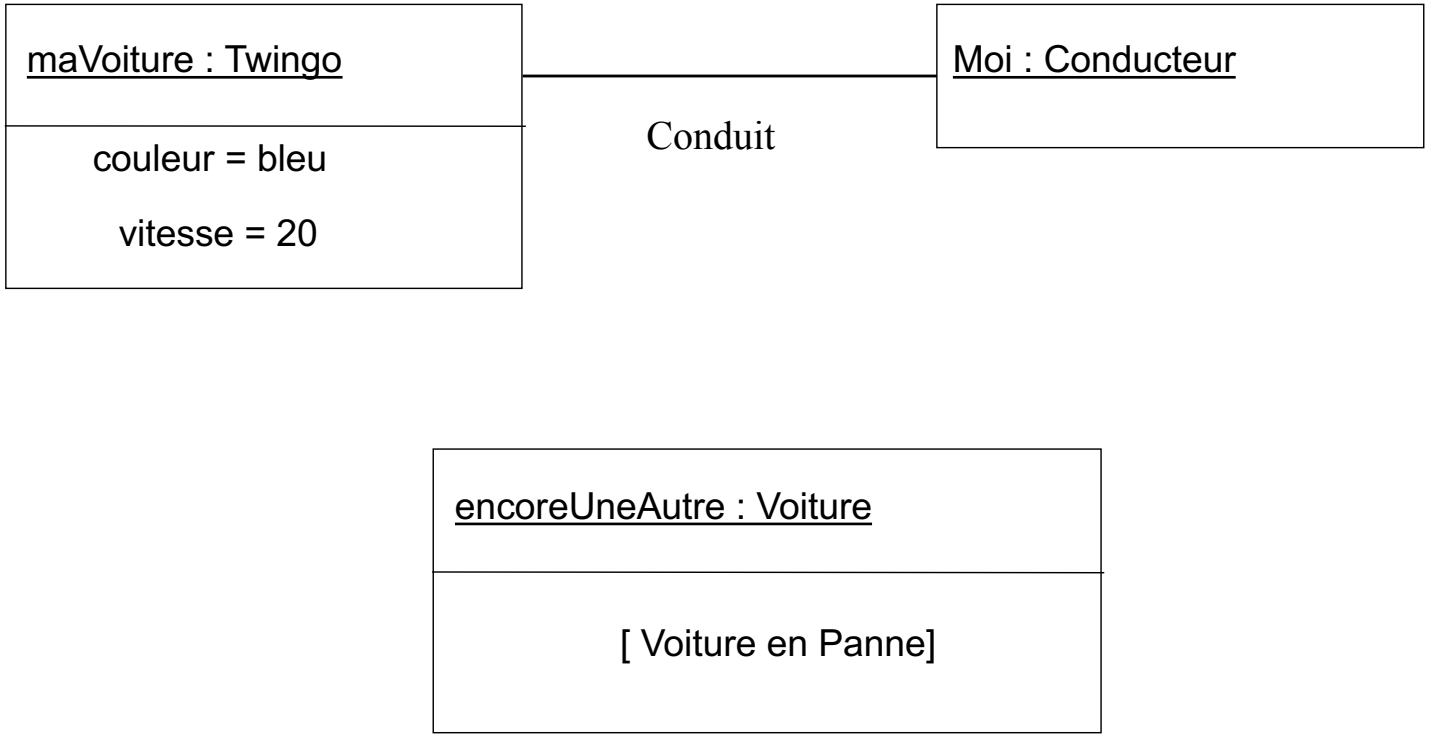


Diagramme d'objet (4)

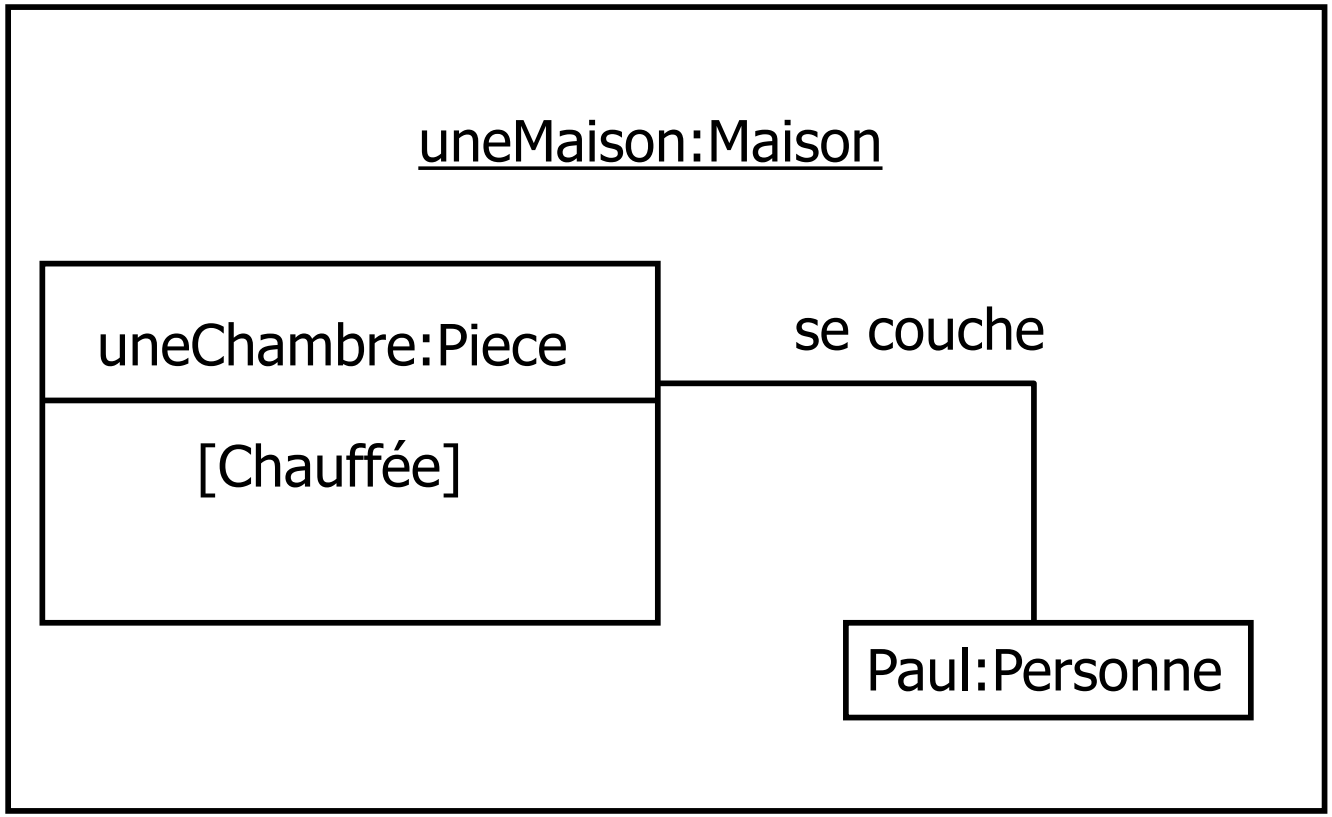
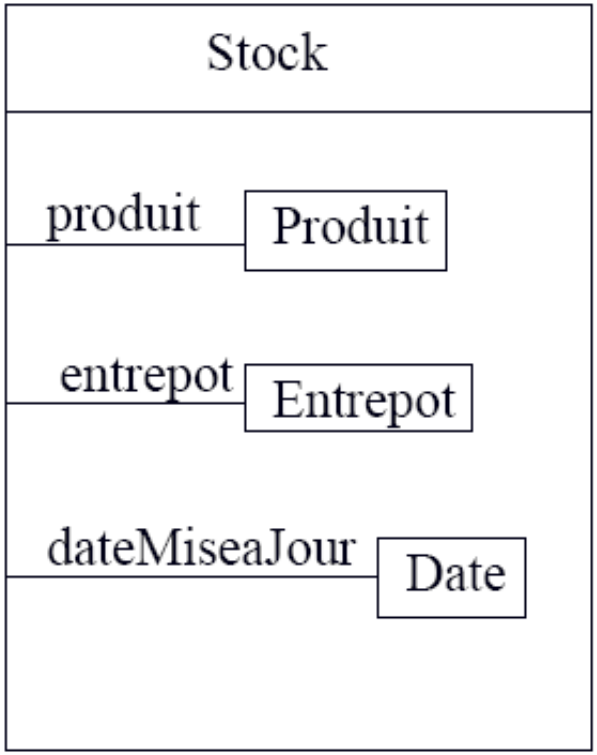
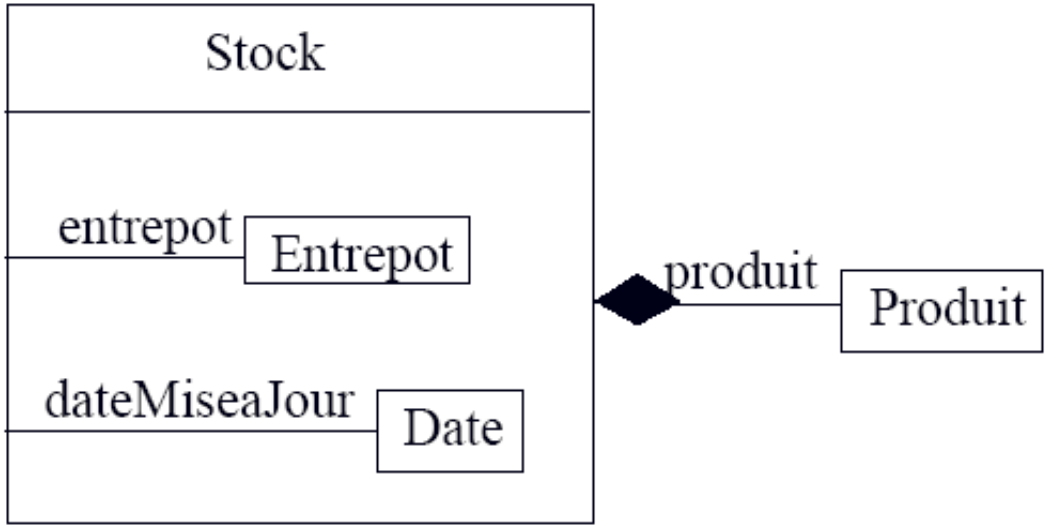


Diagramme d'objets (5)





De l ’objet à la classe

- Une classe représente un ensemble d ’objets.
- Un objet sera obligatoirement attaché à une classe.
- Les objets similaires sont regroupés dans la même classe.

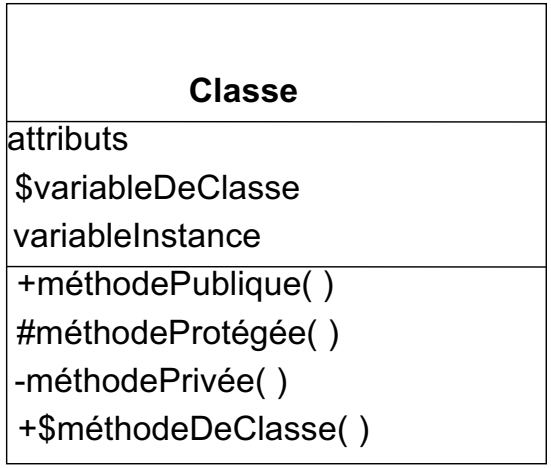
Diagramme de classe (1)

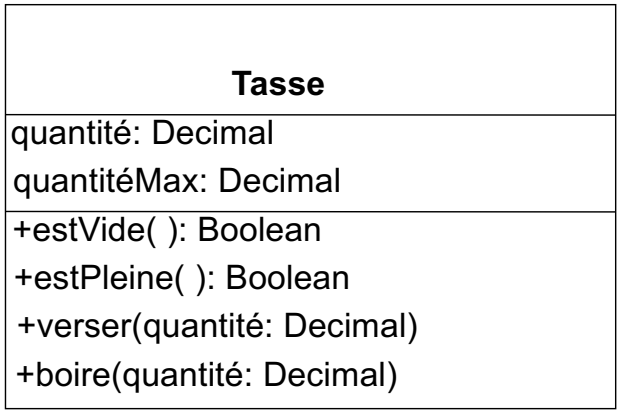
- Une classe est la déclaration statique
  - Attributs:
    - Définition des variables d 'instances
    - Définition des variables de classes
  - Comportement:
    - Définition des méthodes des objets
    - Définition des méthodes de classes
  - Visibilté (public, private, protected)

Diagramme de classe(2)

- **Publique** : un attribut ou une méthode publique est spécifée avec le signe +.
- **Privée** : un attribut ou une méthode privée est spécifée avec le signe –.
- **Protected** : un attribut ou une méthode protégée est spécifée avec le signe #.

Diagramme de classe (3)





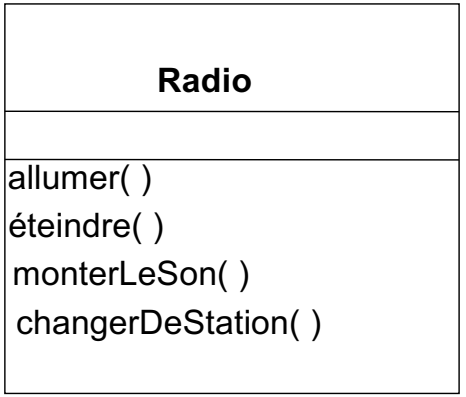
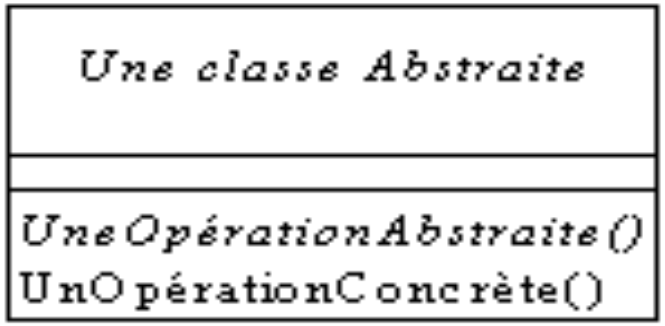


Diagramme de classe (4)



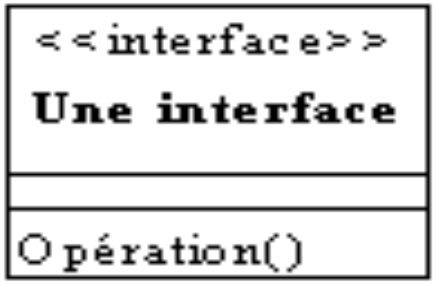


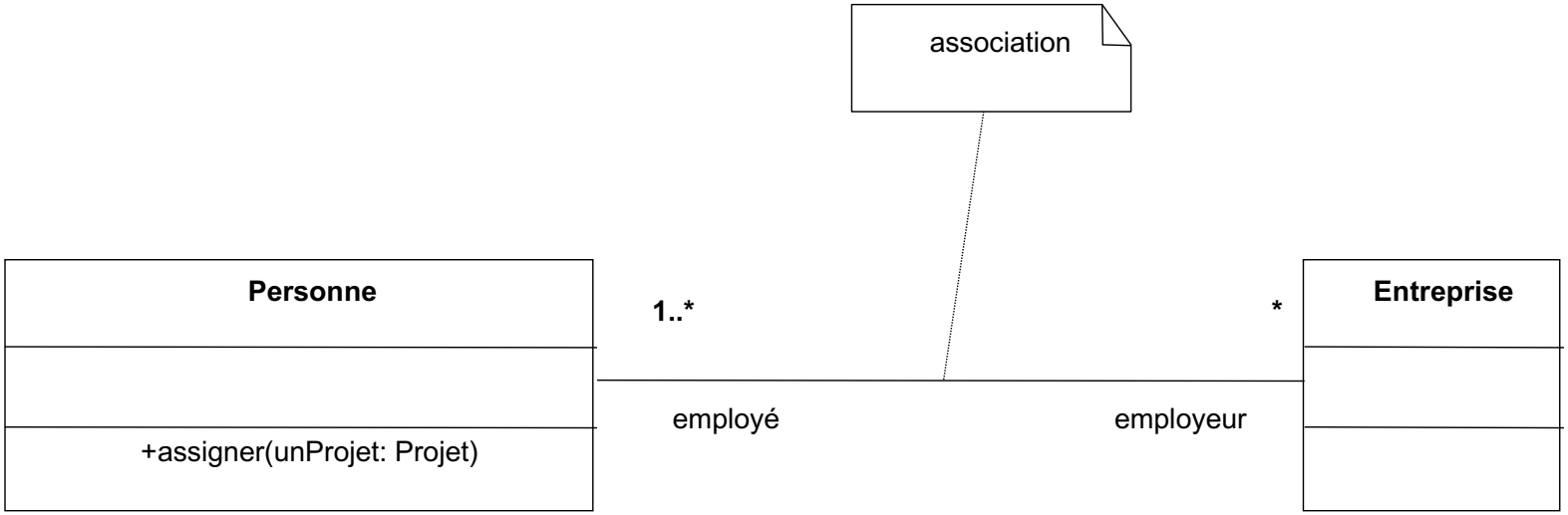
Diagramme de classe (5)

- Relation entre classes
  - Associations sont déclinées en
    - Association simple
    - Agrégation
    - Composition
  - Généralisation/Spécialisation
  - Réalisation

Diagramme de classe (6)

- Une association représente les liens qui existent entre les instances des classes associées. Plusieurs liens entre objet peuvent se projeter sur une même association
- L ’association exprime une connexion sémantique bidirectionnelle entre classe

Diagramme de classe (7)



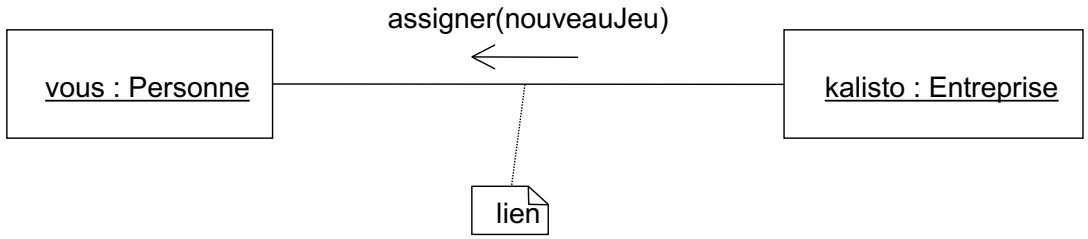


Diagramme de classe (7)

- Le sens de lecture d ’une association peut être précisé

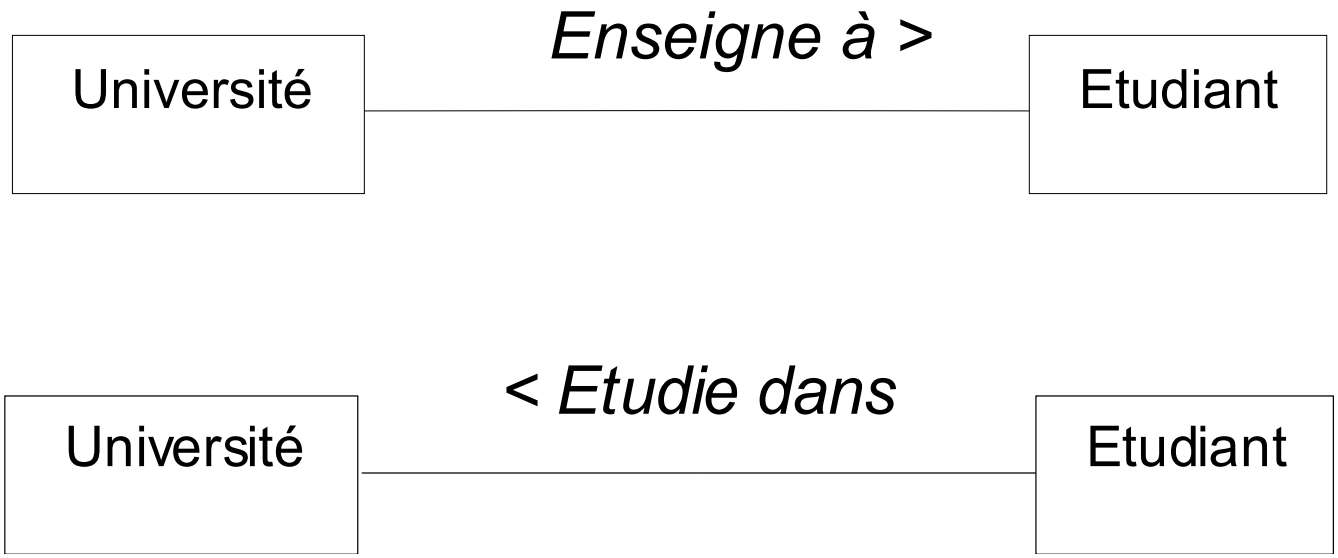


Diagramme de classe (8)

- Le rôle d'une classe pour une association peut être défini.

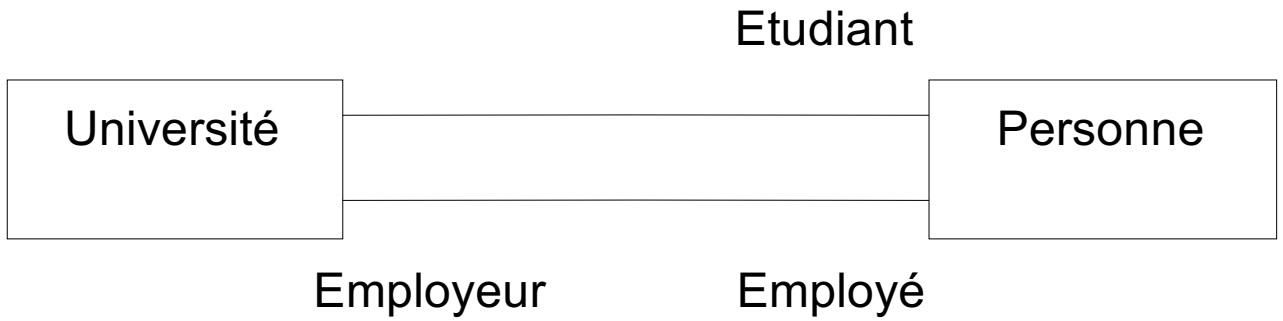


Diagramme de classe (9)

- Une association peut être précisée par les cardinalités

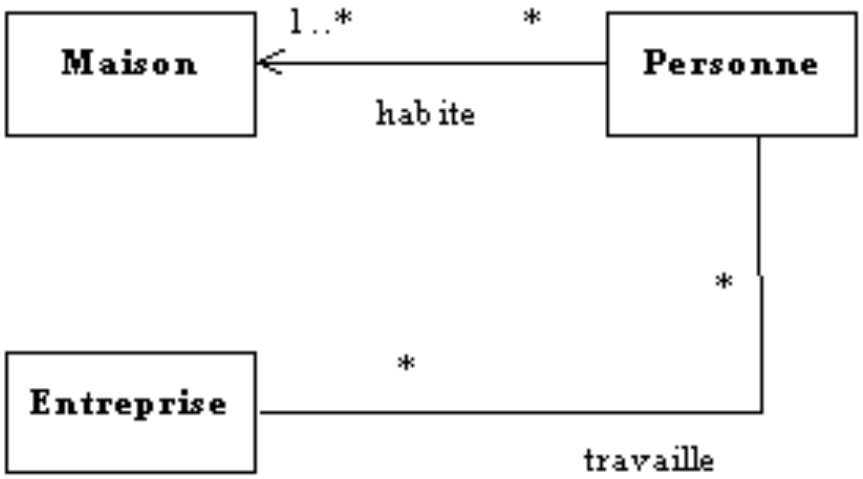


Diagramme de classe (10)

- La représentation des cardinalités

1	Un et un seul
0..1	Zéro ou un
M .. N	De M à N (entiers naturels)
*	Plusieurs
0 .. *	De zéro à plusieurs
1 .. *	D'un à plusieurs

Diagramme de classe (11)

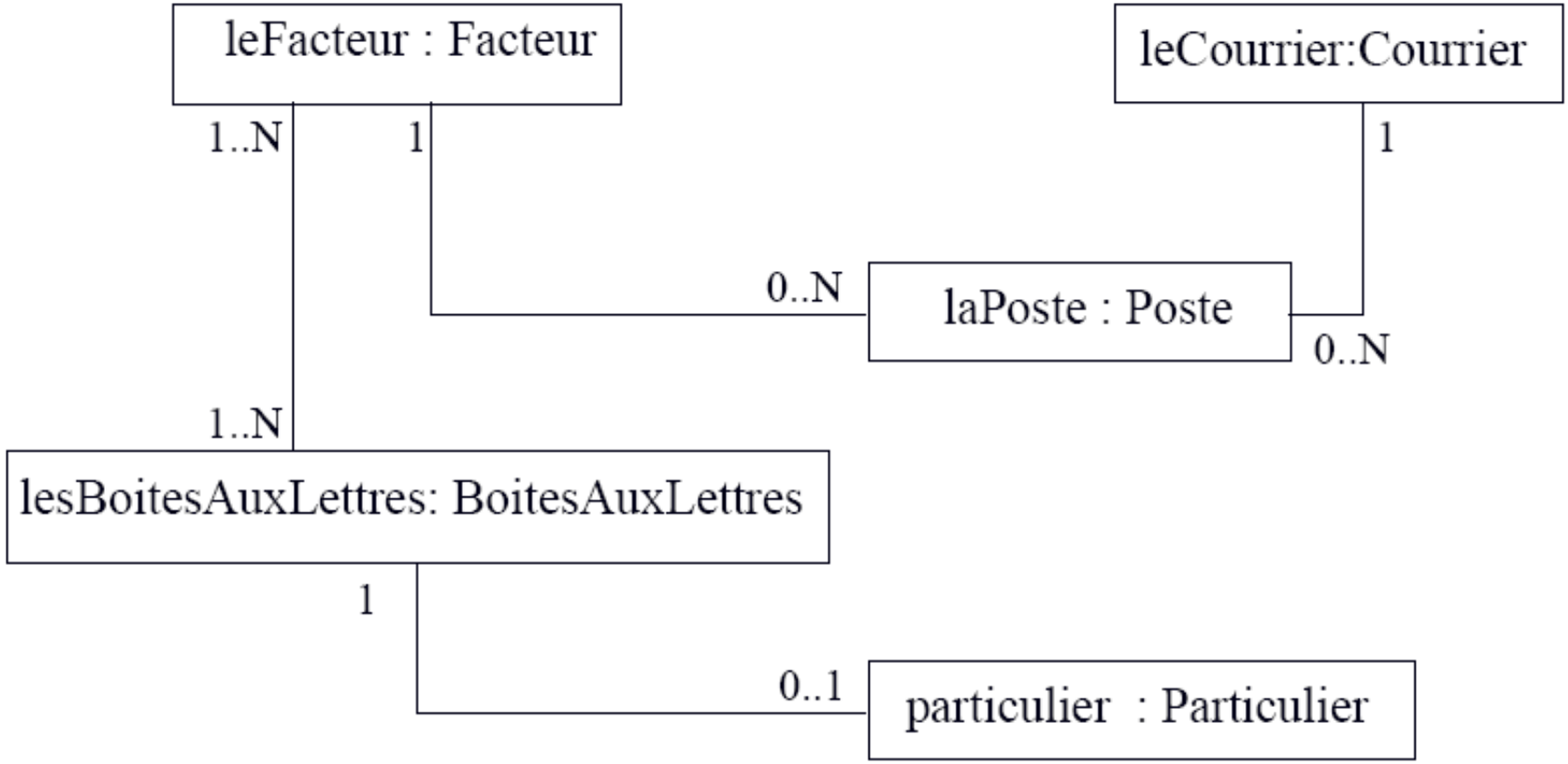


Diagramme de classe (12) Dépendance

- On peut raffiner une association en lui donnant une direction. On exprime alors une relation de dépendance.
- *Le mobile ne connaît pas le labyrinthe*

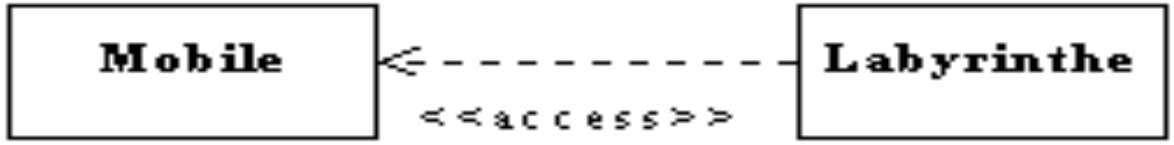


Diagramme de classe (13) Aggrégation

- Raffinement de l ’association
- Correspond à une relation : *partie-de* ou *composant-composé*
  - La classe A fait partie de la classe B
  - L’état d’une instance de la classe B contient temporairement une instance de la classe A
  - *une voiture a 4 roues*
- La relation est orientée (navigation)

Diagramme de classe (14) Composition

- Relation partie-de plus stricte : pas de partage
- Le composé encapsule ses composants
- Durée de vie du composant est égale à celle du composé.

Diagramme de classe (15)

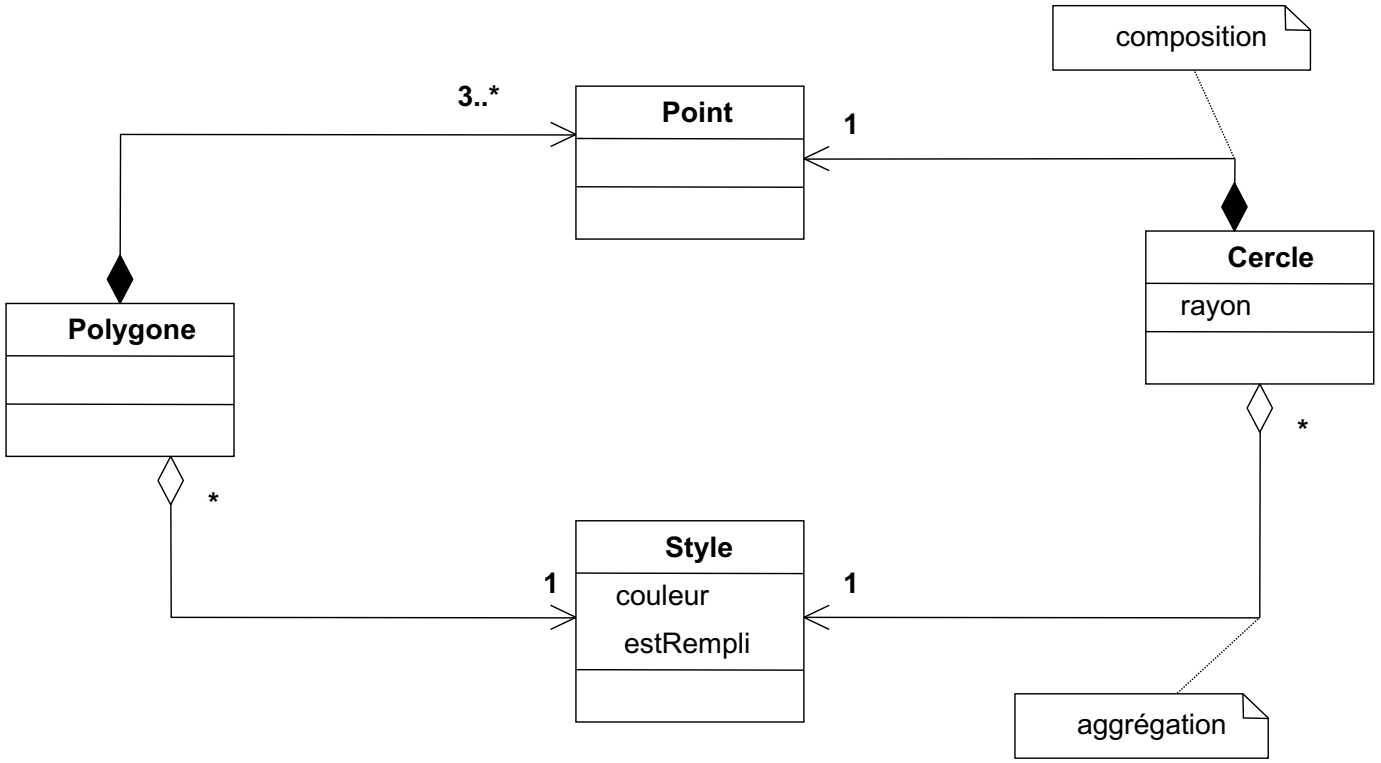


Diagramme de classe (16) Généralisation/Spécialisation

- La relation Généralisation/spécialisation se traduit par « *est une sorte de* ». Elle implique le partage de code et le sous-typage

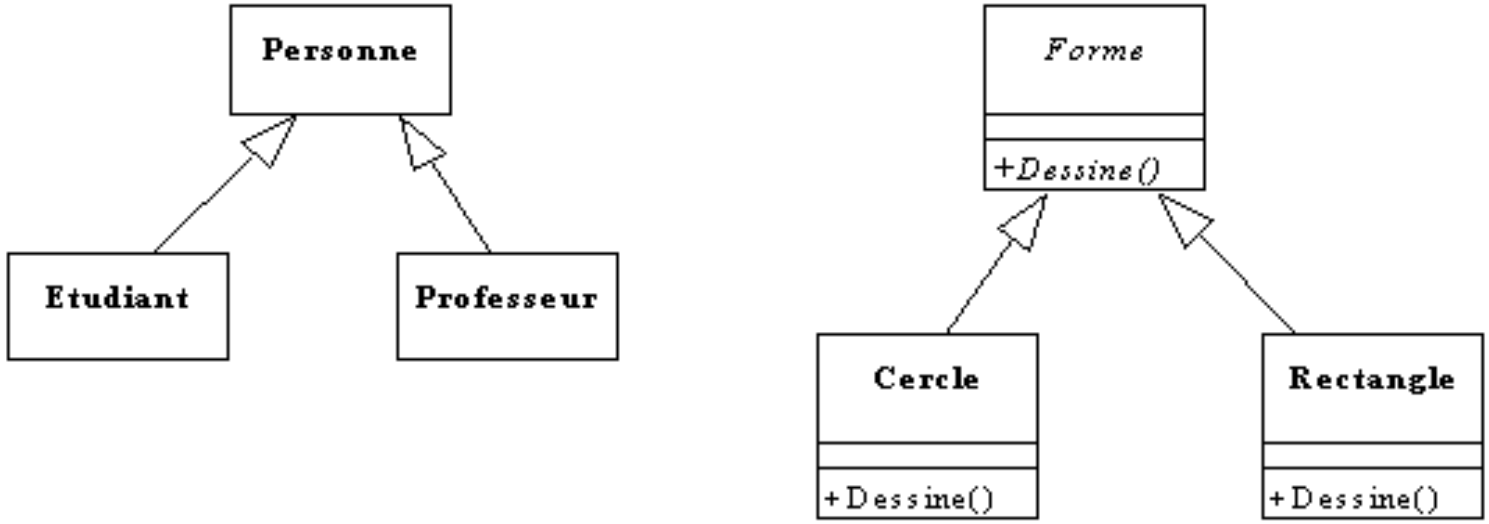
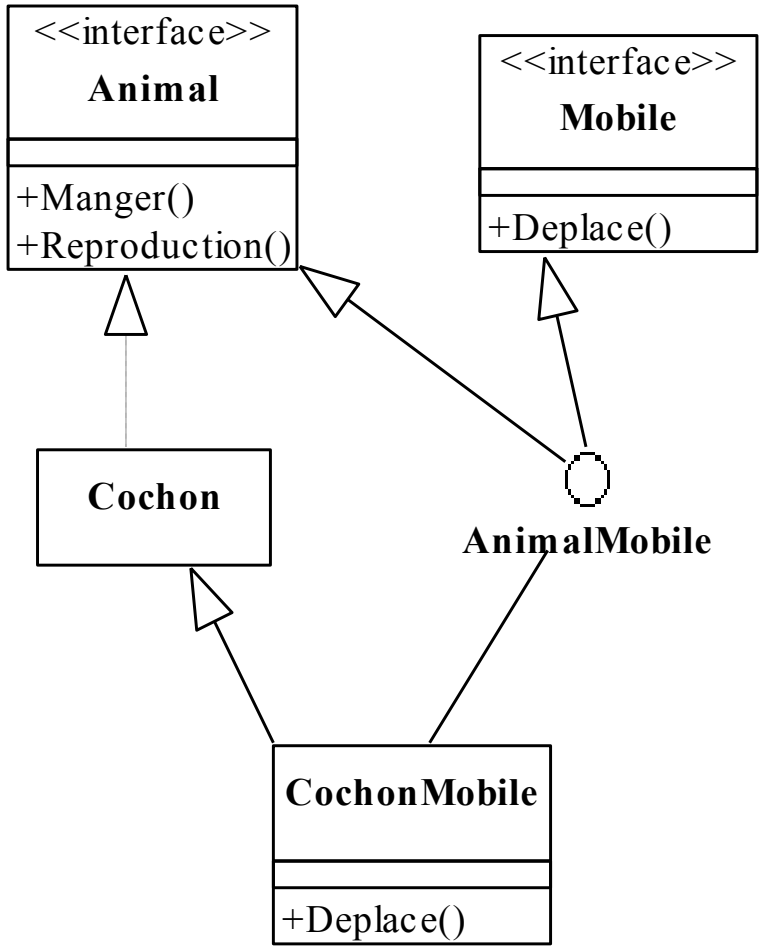


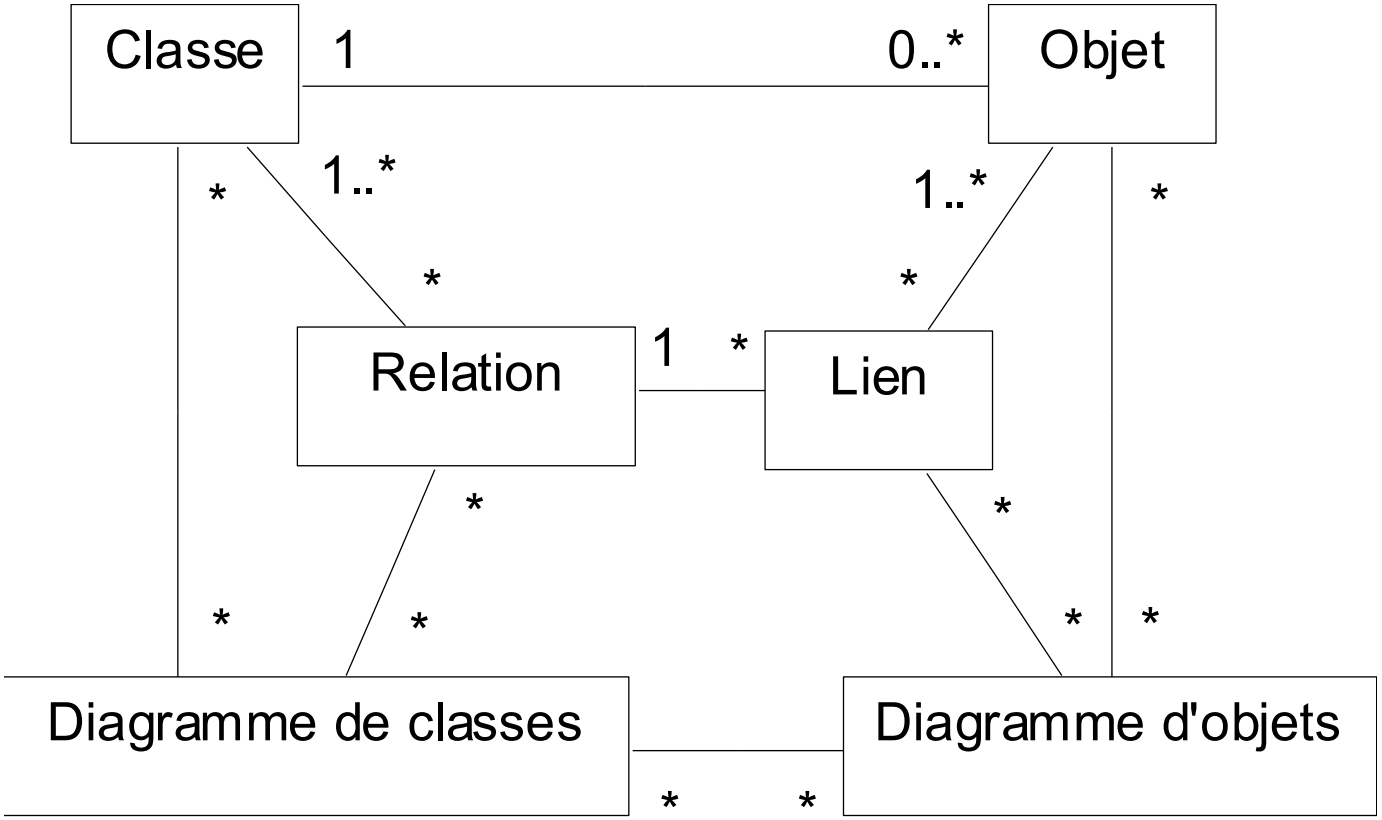
Diagramme de classe (17)
Interface

- L’interface est la définition d’un rôle (type abstrait). Elle matérialise une partie du comportement d ’un objet.Elle définit un ensemble de signature.

Diagramme de classe (18) Interface



Correspondances entre diagrammes



Comportement

- Le comportement dépend de l'état de l'objet
- Les objets interagissent entre eux par envoie de messages
  - Un objet réalise une opération lorsqu'il reçoit un message d'un autre objet (client)
  - L'envoie de message un concept très général pouvant être mis en oeuvre de différentes manières (par exemple par des évènements).

Diagramme de séquence

- Les diagrammes de séquences permettent de représenter des collaborations entre objets selon un point de vue temporel
- Les objets sont représentés par une droite verticale en pointillé (axe du temps)
- Le temps s'écoule de haut en bas
- Un rectangle indique quand un objet est actif

Diagramme de séquence

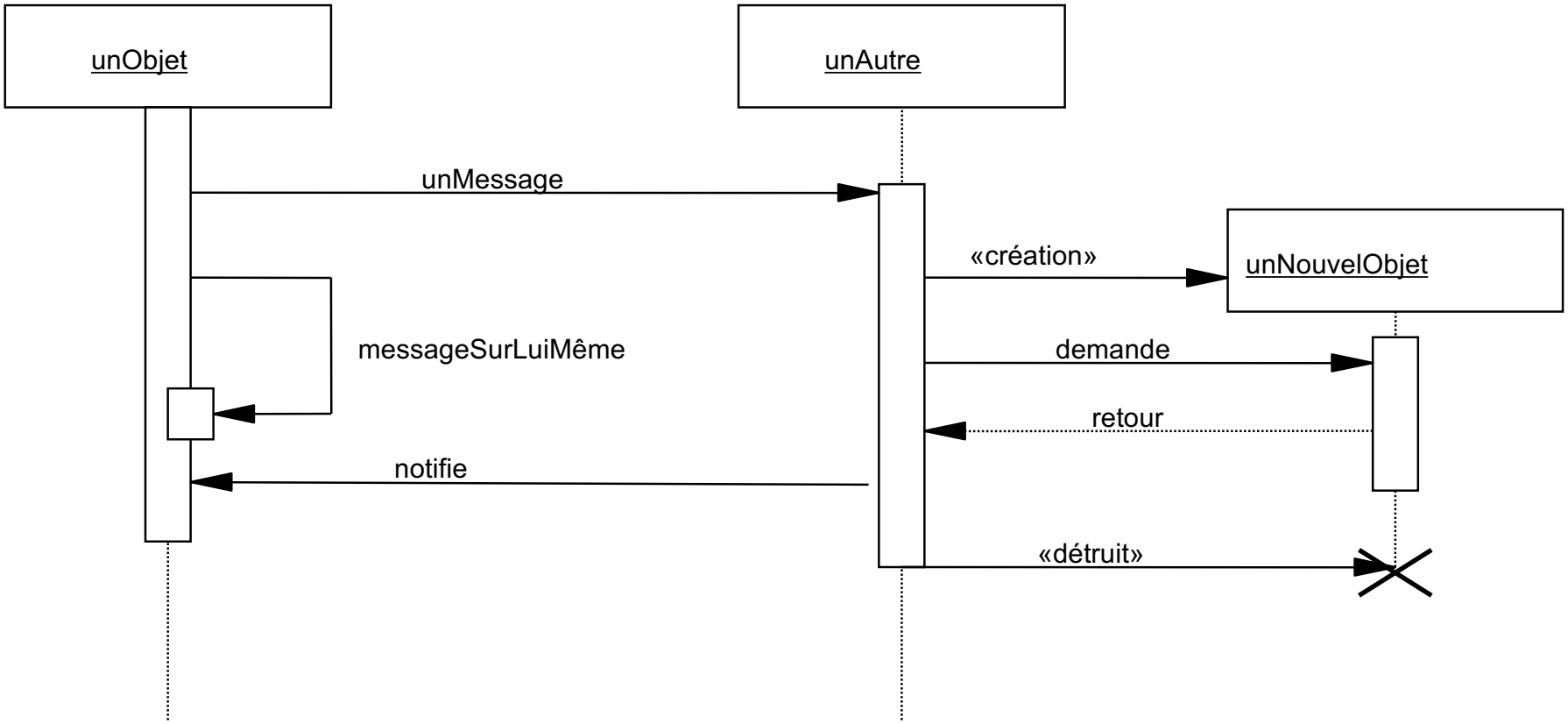


Diagramme de séquence

- L'ordre est matérialisée par la position sur l'axe vertical.

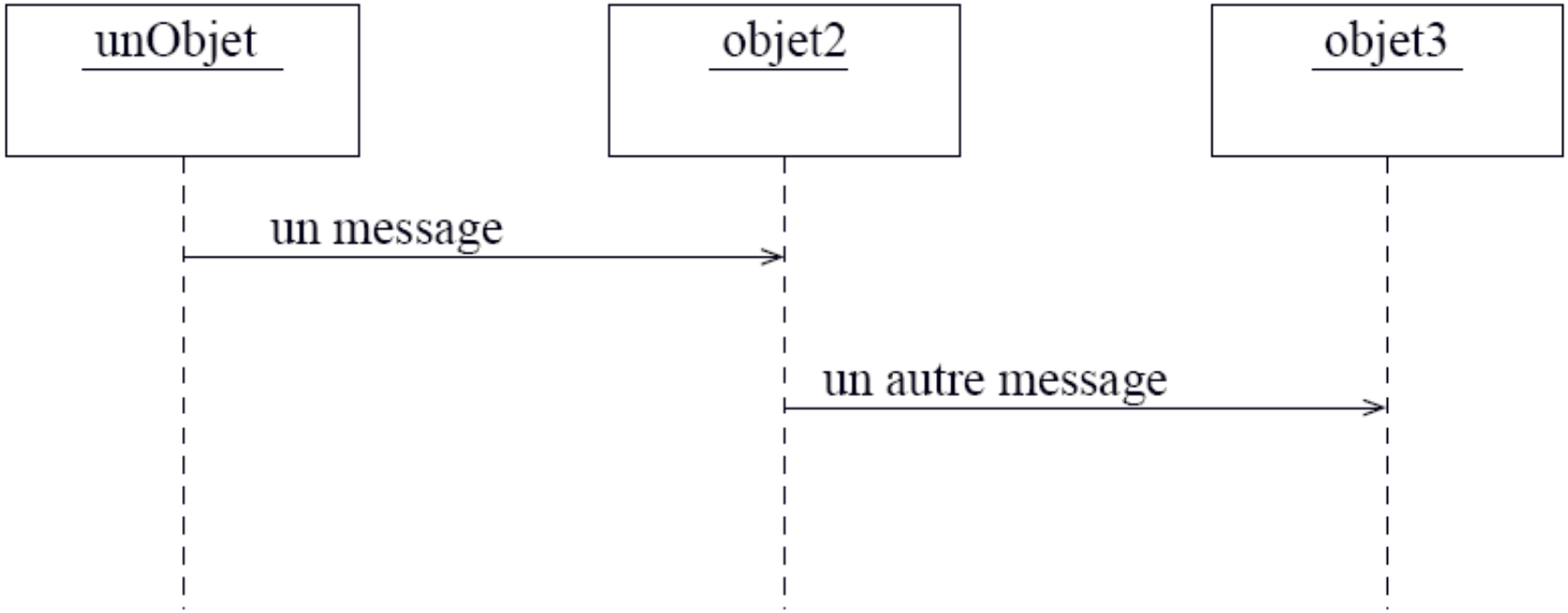


Diagramme de séquence

- Détail des interactions

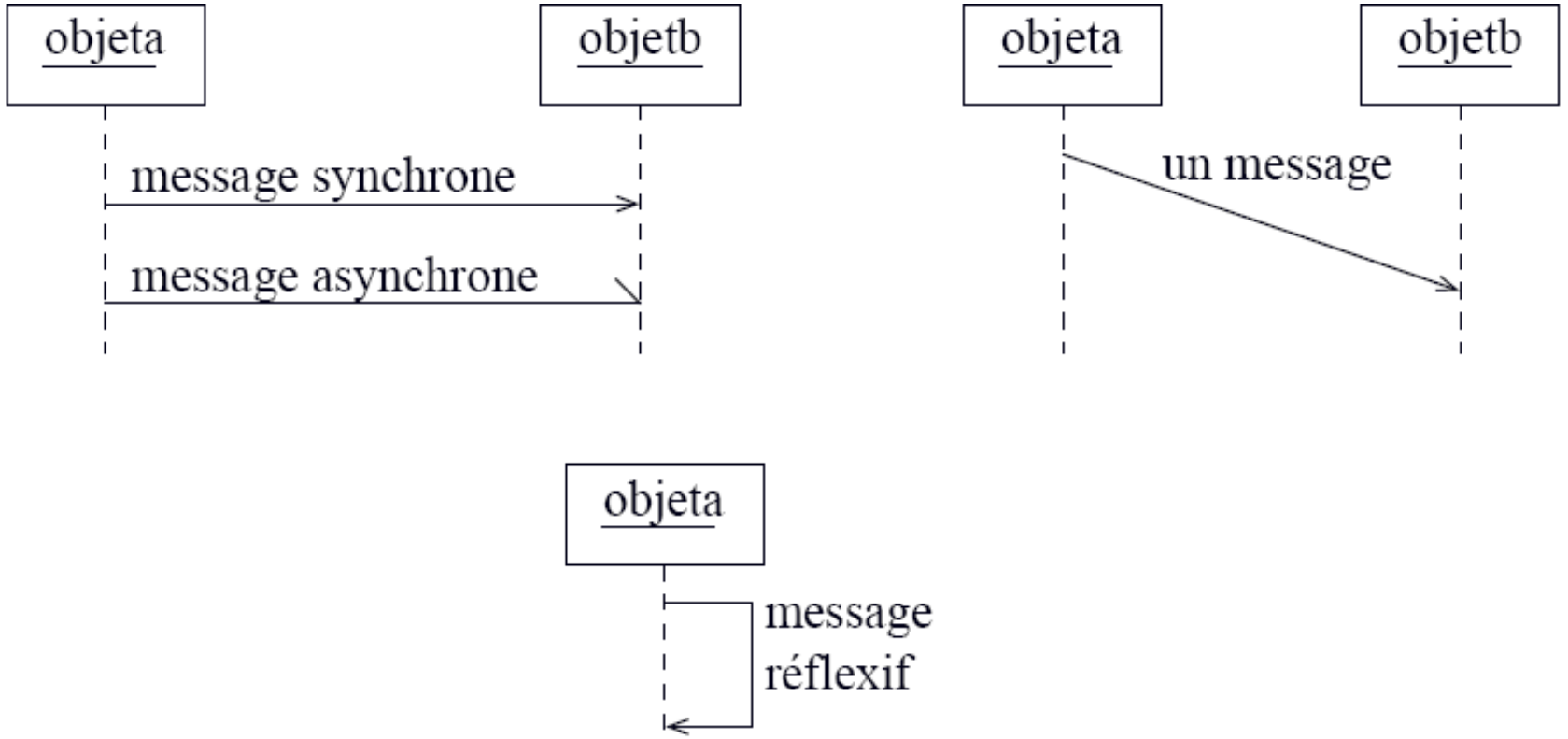


Diagramme de séquence

- Création destruction d'objets

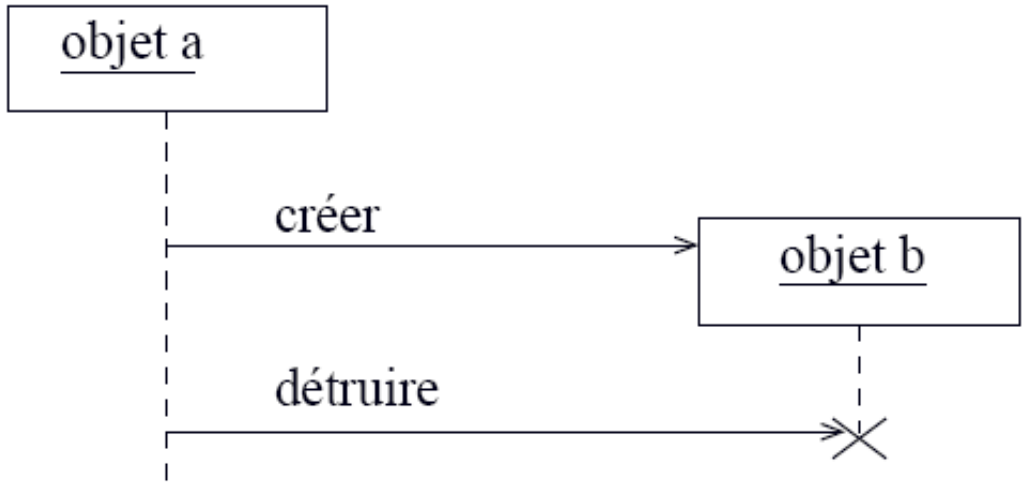


Diagramme de séquence

- Matérialisation des période d'activité des objets.En mode synchrone retour implicite.

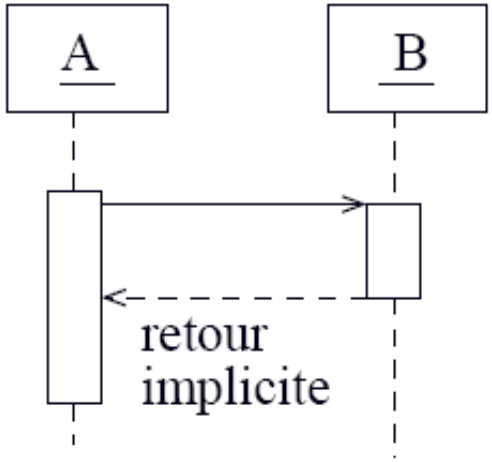


Diagramme de séquence

- En mode asynchrone le retour doit être explicité.

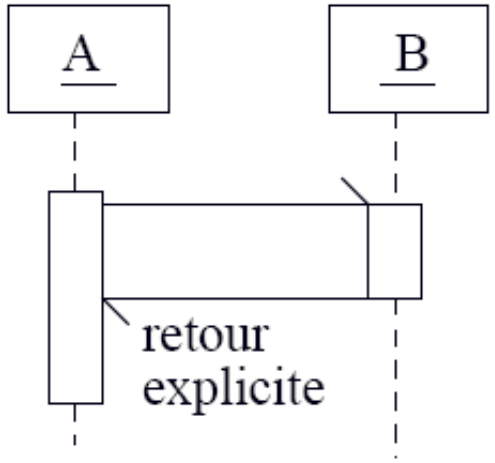


Diagramme de séquence

- Appels récursifs

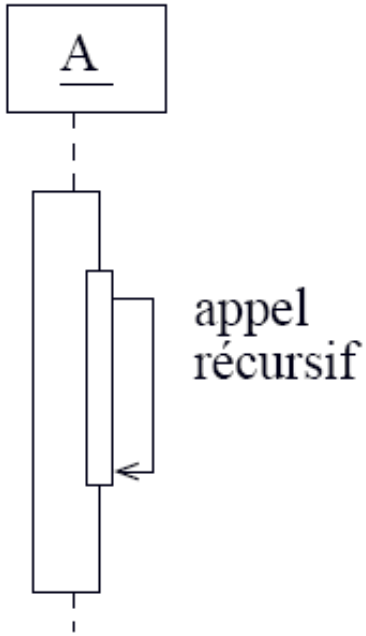


Diagramme de séquence

- Boucles et branchements

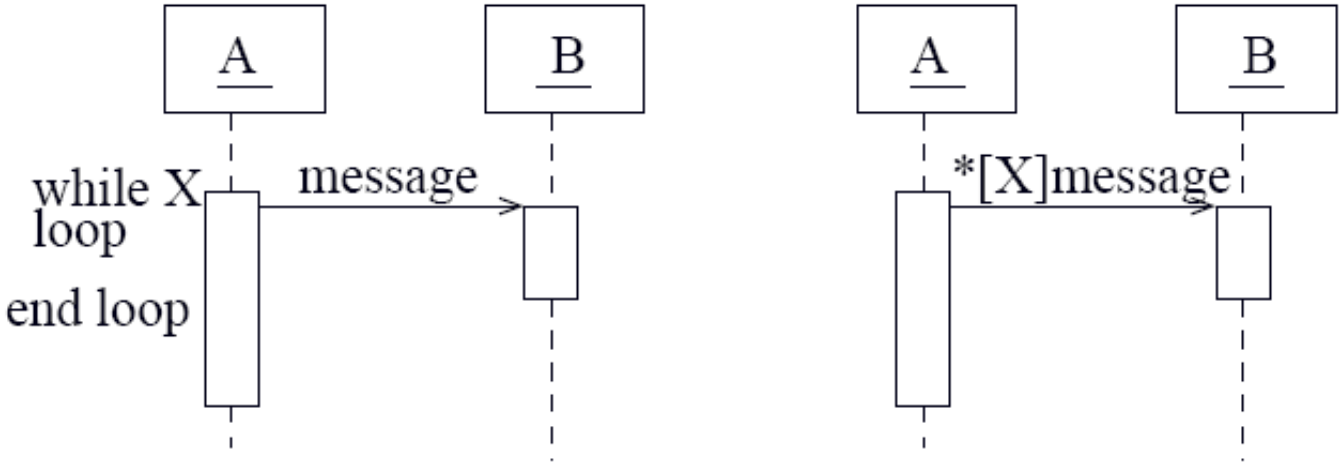
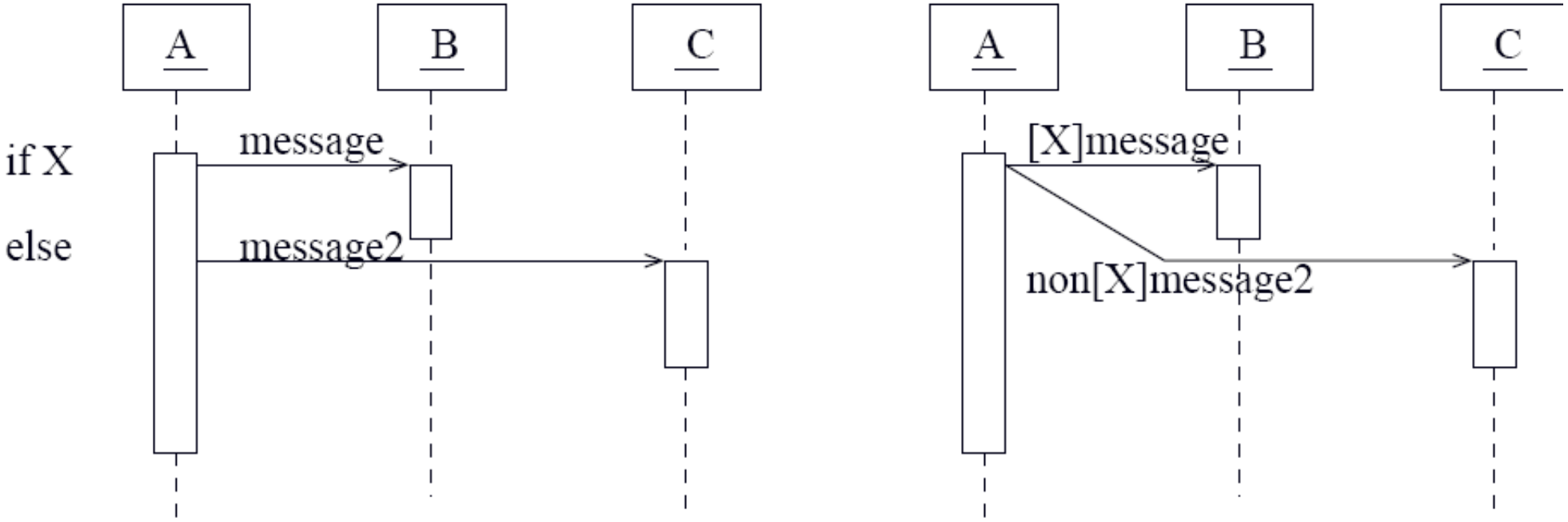
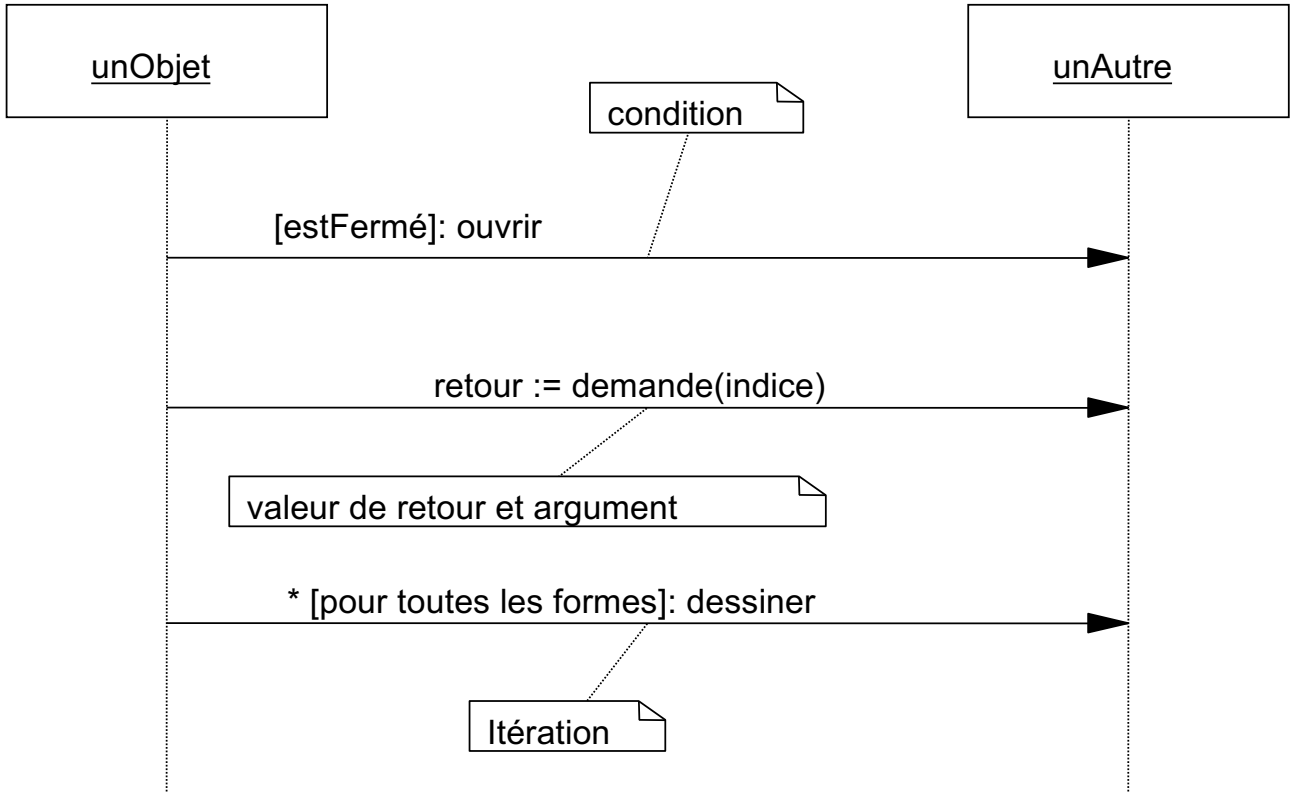


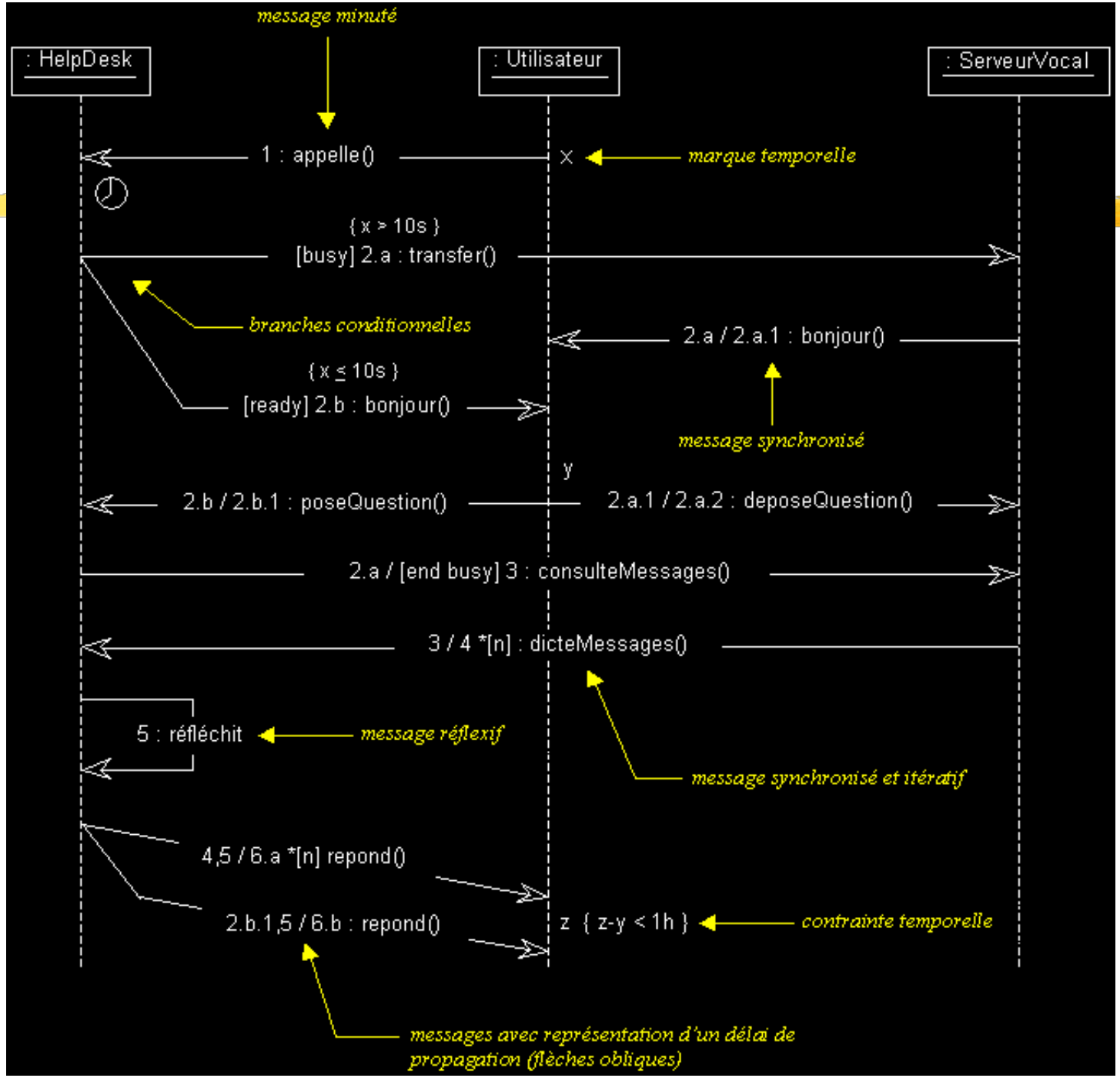
Diagramme de séquence

- Branchement conditionnel



Un exemple


unObjet
unAutre
condition
[estFermé]: ouvrir
retour := demande(indice)
valeur de retour et argument
* [pour toutes les formes]: dessiner
Itération


message minuté
: HelpDesk
: Utilisateur
: ServeurVocal
1 : appelle()
x
marque temporelle
{ x > 10s }
[busy] 2.a : transfer()
branches conditionnelles
2.a / 2.a.1 : bonjour()
{ x ≤ 10s }
[ready] 2.b : bonjour()
message synchronisé
y
2.b / 2.b.1 : poseQuestion()
2.a.1 / 2.a.2 : deposeQuestion()
2.a / [end busy] 3 : consulteMessages()
3 / 4 *[n] : dicteMessages()
5 : réfléchit
message réflexif
message synchronisé et itératif
4,5 / 6.a *[n] repond()
2.b.1,5 / 6.b : repond()
z { z-y < 1h }
contrainte temporelle
messages avec représentation d'un délai de propagation (flèches obliques)

Le diagramme état/transition

- La représentation de ce diagramme est un automate dont les nœuds sont les états d'un objet et dont les transitions sont des évènements, des signaux ou des actions qui produisent un changement d'état.
- Les états sont un état stable de l'objet, il peut être caractérisé par une succession de transition (état conceptuel) ou par la valeur des variables d'un objet (état implémenté).

Le diagramme état/transition

- L'état *initial* d'un objet est spécifié avec un rond noir plein et les états *finaux* avec des ronds noirs encerclés.

Action et activité

- Une **action** est supposée avoir une durée nulle à l'échelle d'évolution des objets.
- Une **activité** par contre a une durée non nulle à l'échelle d'évolution des objets.
- Une action est effectuée à la suite de la réception d'un événement (notion instantanée) alors qu'une activité est effectuée pendant que l'objet est dans un état (notion durable).

Action et activité

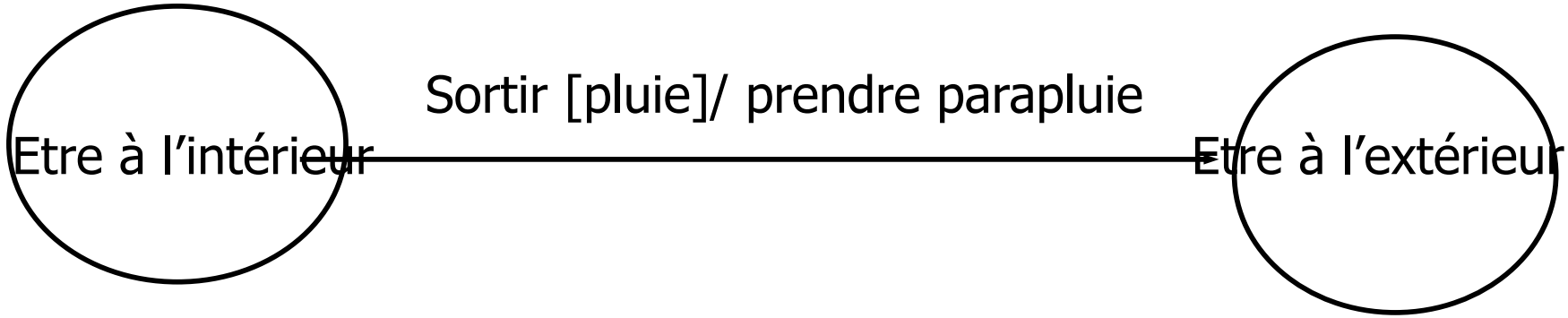
- Une activité est spécifié par le mot clef do.
- On distingue 5 type d'actions différentes.
  - Entry/action:Une action d'entrée est effectuée lorsque l'on rentre dans un état.
  - Exit/action: Une action d'entrée est effectuée lorsque l'on rentre dans un état.
  - On/évènement/action: Il s'agit d 'une action interne associé à l'arrivé d'un événement quand un objet est dans un état, cela n'entraine pas de changement d'état.

Action de transition

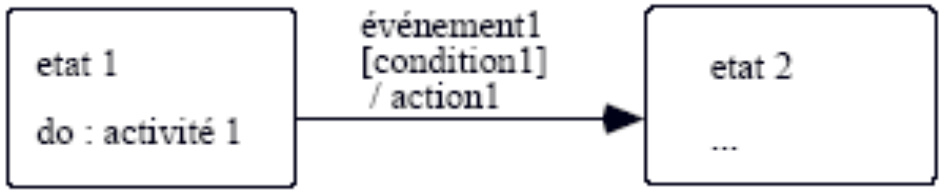
- Le dernier type d'action concerne les actions de transitions qui peuvent conduire à un changement d'état. Une condition peut être attachée à la reception de l'événement attaché à la transition.
  - Quand je me sors, si il pleut, je prends mon parapluie.

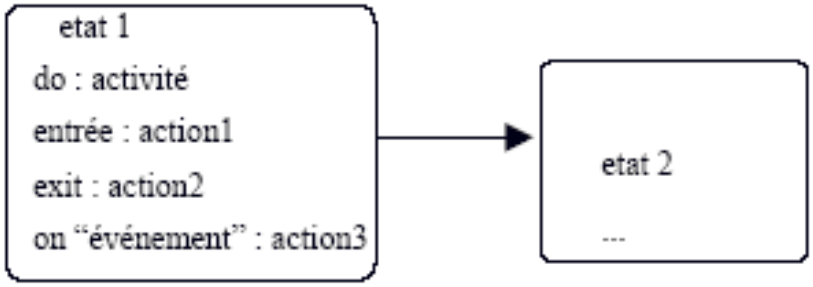
Action de transition

- *Quand je sors* -> événement
- *Si il pleut* -> condition
- *Je prends mon parapluie* -> action

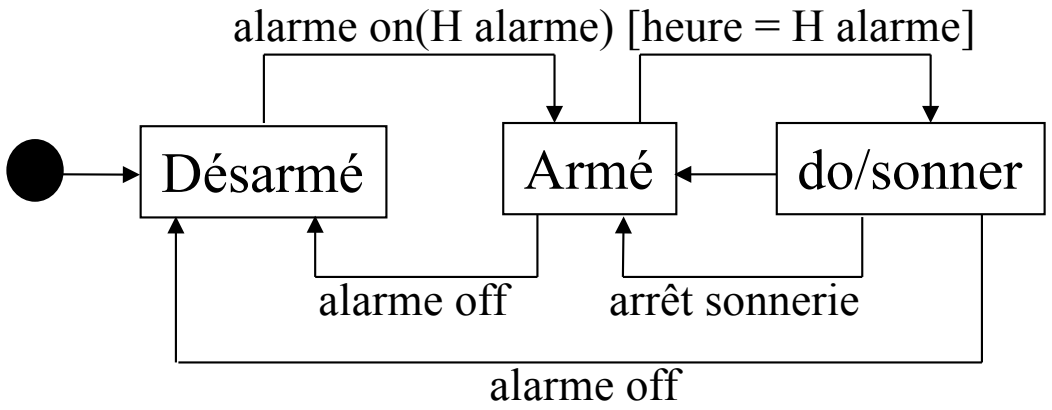


Exemple d'état.

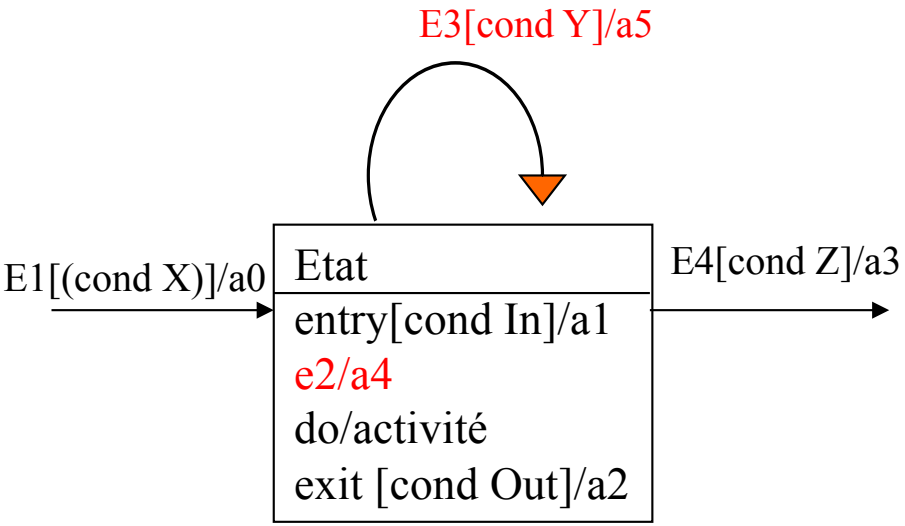




Autres Exemples.

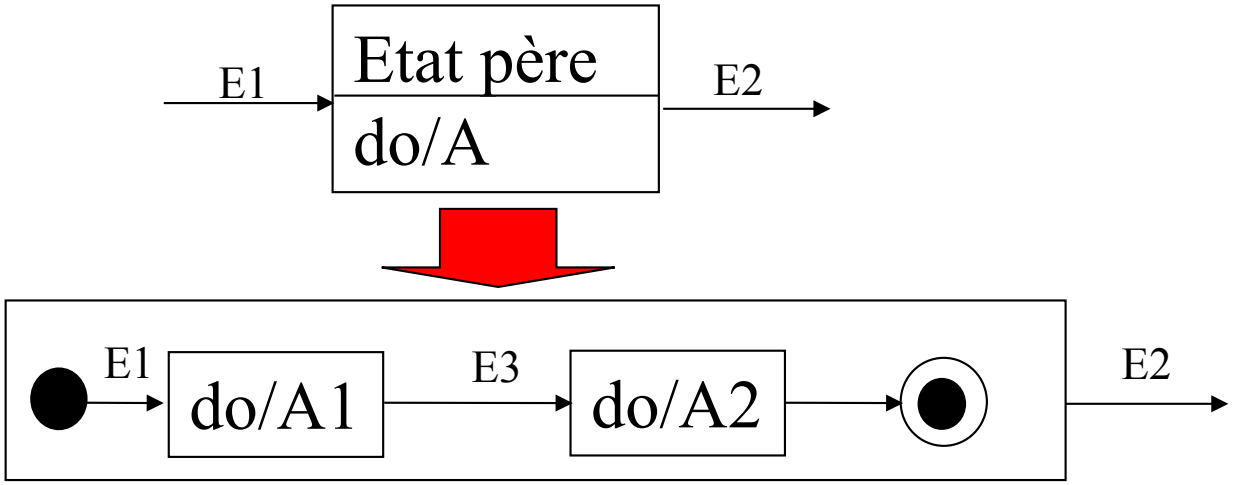


Résumé d'un élément du diagramme état/transition

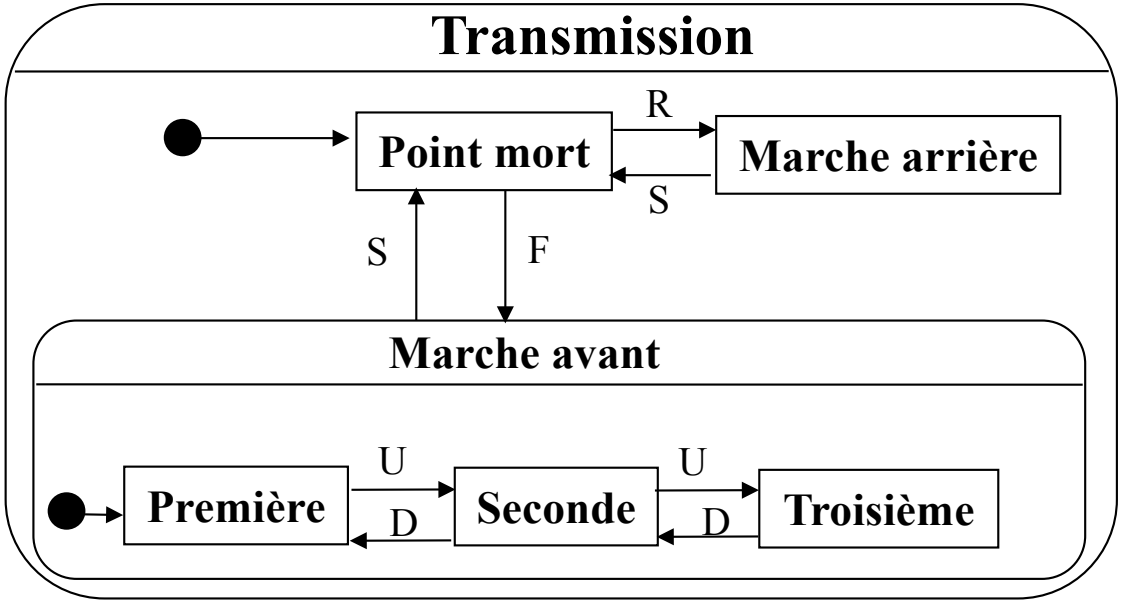


Hiérarchie d'états

- Les diagrammes d'états à plat deviennent rapidement illisibles, il faut structurer hiérarchiquement.
  - Raffiner : décomposer en sous-états
  - Synthétiser : regrouper dans un même état

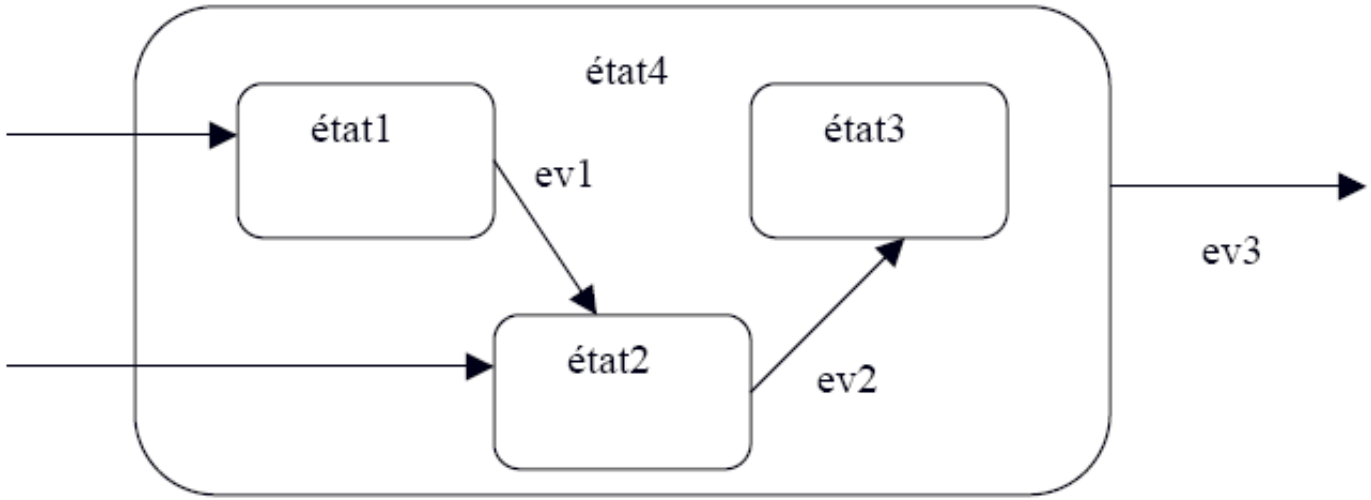


Hiérarchie d'états



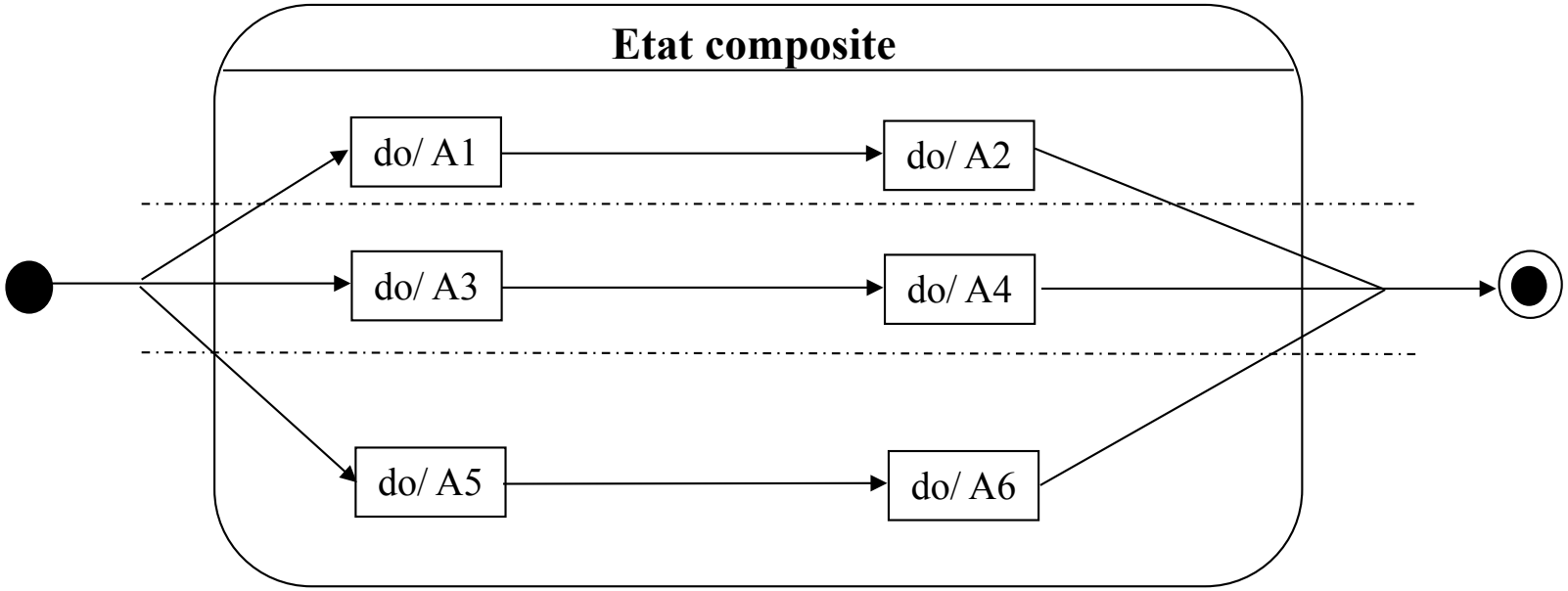
- Un sous-état hérite de son sur-état des actions internes, des transitions de sortie, il n'hérite pas des transitions en entrée ni des activités.

Hiérarchie d'états



Ici, l'état 'état4' est un sur-état des états 'état1', 'état2', 'état3'. Les deux transitions qui rentrent dans l'état 'état4' spécifient dans quel sous-état on arrive. La transition sortante de l'état 'état4' exprime que, quel que soit le sous-état dans lequel l'objet se trouve, si l'événement 'ev3' arrive, alors l'objet sort de l'état 'état4'.

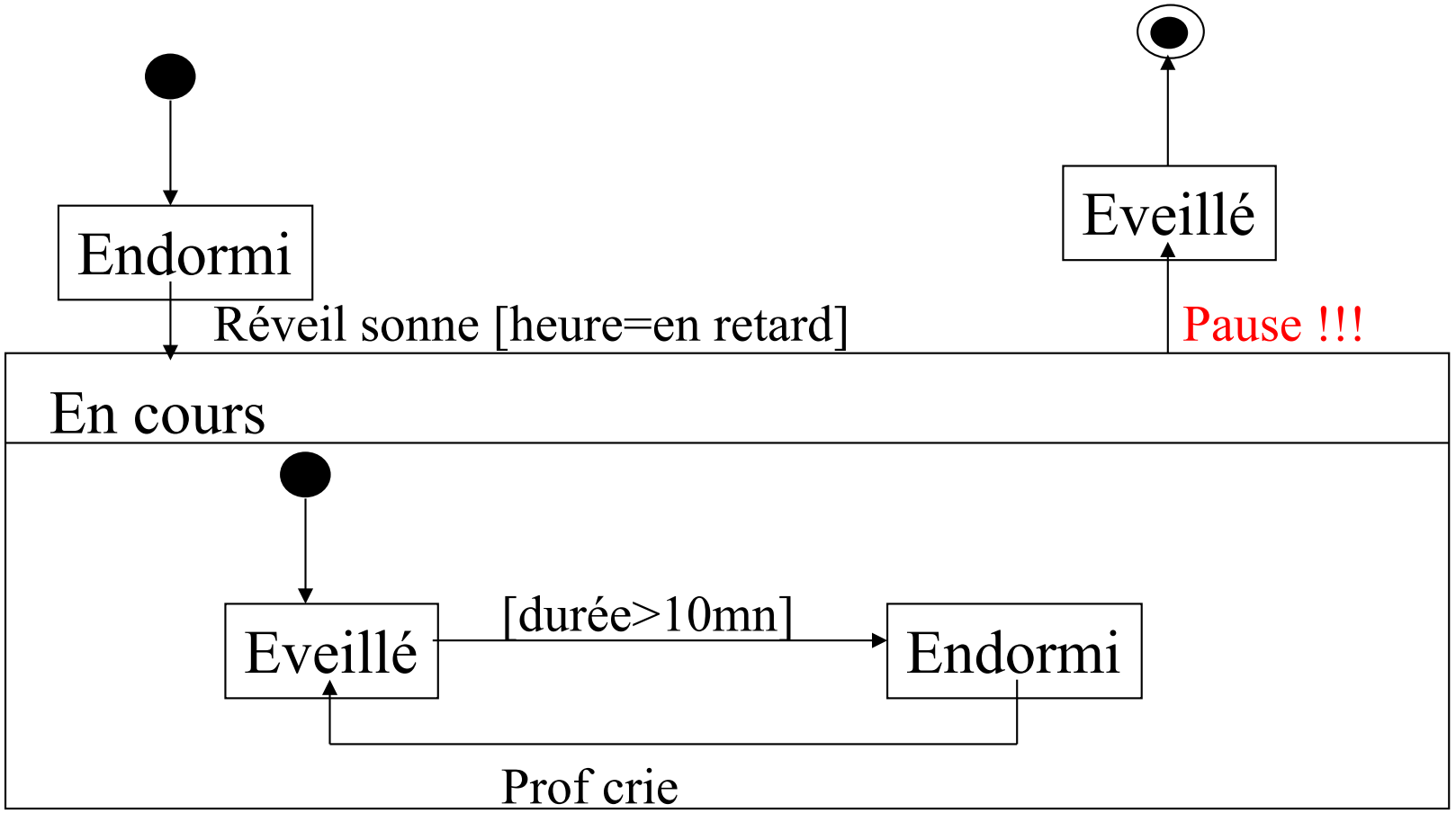
Parallélisme et synchronisation.



Activation de A1, A3 et A5
en même temps

Transition quand A2, A4 et A6
sont toutes finies

Quel est la signification ?



Dernier exemple de uml.free.fr