

# **PARTIE 1 : PRINCIPES DU GENIE LOGICIEL**

- **Qu'est-ce que le Génie logiciel ?**
- **Etat des connaissances en Génie logiciel**
- **Démarche du développement et cycle de vie du logiciel**
- **Qualité du logiciel**
- **Sûreté de fonctionnement**

# 1 – Qu'est-ce que le Génie logiciel ?

- **Génie logiciel ou Ingénierie du logiciel :**

Théories , méthodes , outils et ressources humaines permettant de construire des systèmes logiciels et d'en assurer l'évolution .

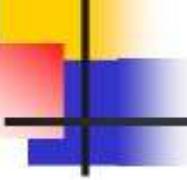
- **Programmation vs. Génie logiciel**

- Programmation = Activité personnelle
- Génie logiciel = Activité d'équipe structurée organisée autour d'un projet

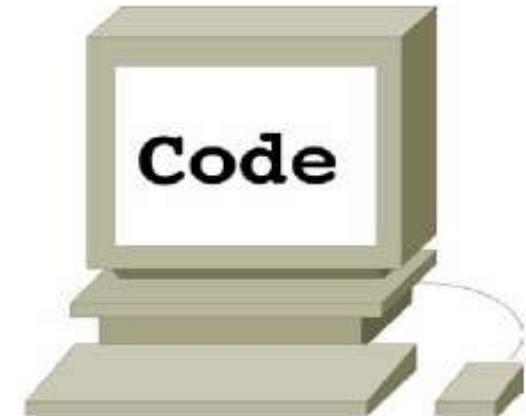
- **Quelques faits sur l'industrie du logiciel**

- L'économie de toutes les nations développées dépend du logiciel ( part du PNB très importante )
- Le coût du logiciel domine maintenant celui du matériel
- Le logiciel coûte plus à maintenir qu'à développer
- Le coût de production du logiciel est très variable et difficile à évaluer
- Le logiciel a une importance croissante dans les systèmes critiques ( centrales nucléaires , systèmes de transport : avion , train , auto , données bancaires ,etc... ) => il doit être fiable et sûr
- Les utilisateurs sont de plus en plus exigeants : certains critères de qualité doivent être satisfaits : fiabilité , efficacité , possibilité d'évolution , ergonomie , etc...
- La concurrence est énorme dans chaque secteur de l'industrie du logiciel .

- Systèmes informatiques
  - 80 % de logiciel
  - 20 % de matériel
- Depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement
  - Le matériel est relativement fiable
  - Le marché est standardisé
- Les problèmes liés à l'informatique sont essentiellement des problèmes de **Logiciel**



Besoins



Quelle méthode pour passer de l'expression des besoins au code de l'application ?

Méthode ?

UML ?

## 2 – Etat des connaissances en Génie logiciel

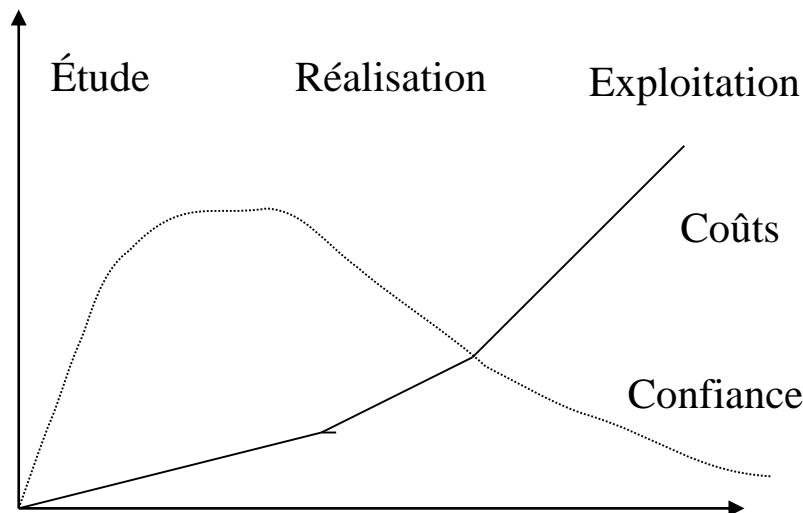
- Discipline informatique jeune et en pleine évolution . ( a débuté dans les années 60 )
- **Beaucoup de techniques informelles** ( pas de consensus sur ce que doit faire un système logiciel )

Remarque : D'autres disciplines de l'ingénieur telles que l'électricité , la mécanique possèdent des méthodes précises pour décrire leurs besoins . En informatique , les méthodes mathématiques commencent seulement à se développer .

- **Conséquence : Beaucoup d'incertitudes associées au processus de développement :**

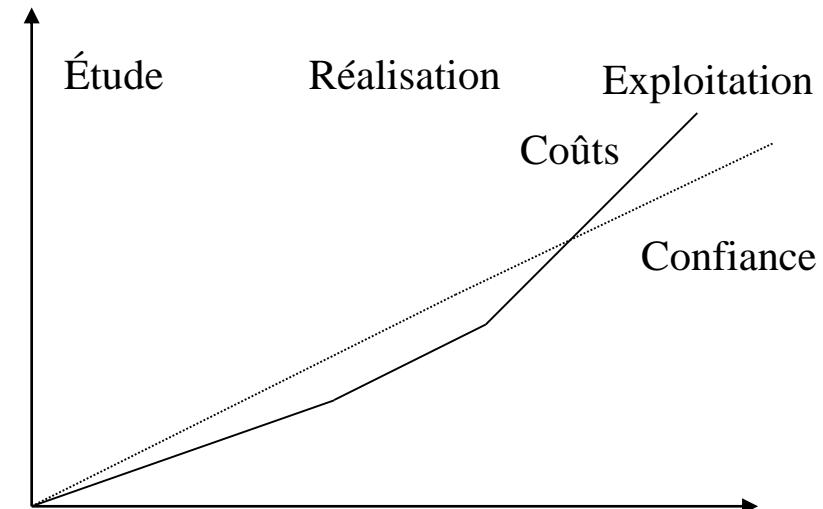
Evolution des coûts et de la confiance en l'issue du développement

**Méthodes non structurées**



La 1ère version est très éloignée du produit final

**Méthodes structurées**



Est-on sûr d'être sur la bonne voie ?

## 2 – Etat des connaissances en Génie logiciel ( Suite )

### Evolution vers une science de l'ingénieur

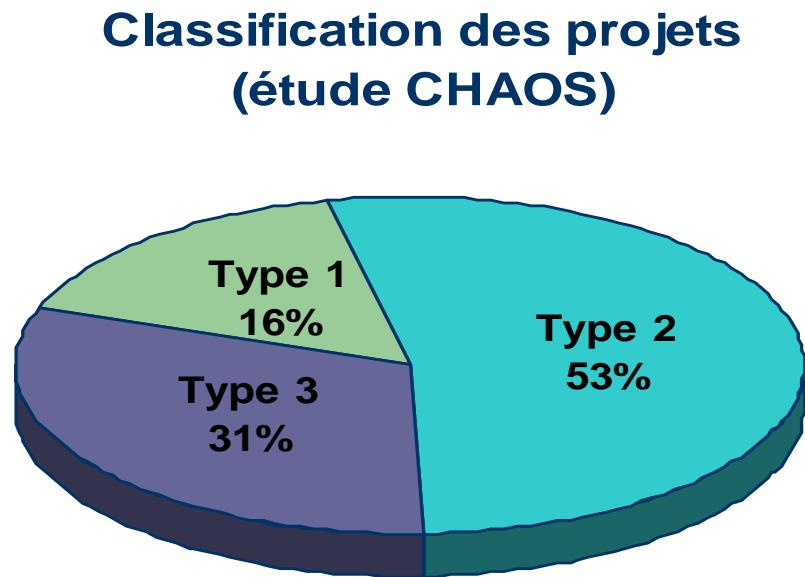
- **Facteurs positifs :**

- Economie en pleine évolution : 1985 : 140 Mrd \$ - 1995 : 450 Mrd \$ -2000 : 900 Mrd \$
- Nombreuses collaborations académie – industrie ( projets de recherche mixtes )
- Concurrence très importante pour la domination du marché du logiciel ( Microsoft , Sun Microsystems , Oracle , Borland , Netscape , etc... )
- Effet d'entraînement dans la compétition pour les petits éditeurs de logiciel

- **Facteurs négatifs :**

- Monopole économique
- Plusieurs faillites et débâcles économiques
- Enseignements inadaptés dans beaucoup d'universités
- Pas encore de véritable démarche scientifique dans la production de logiciel ( théories et méthodologies jeunes et en cours de développement )

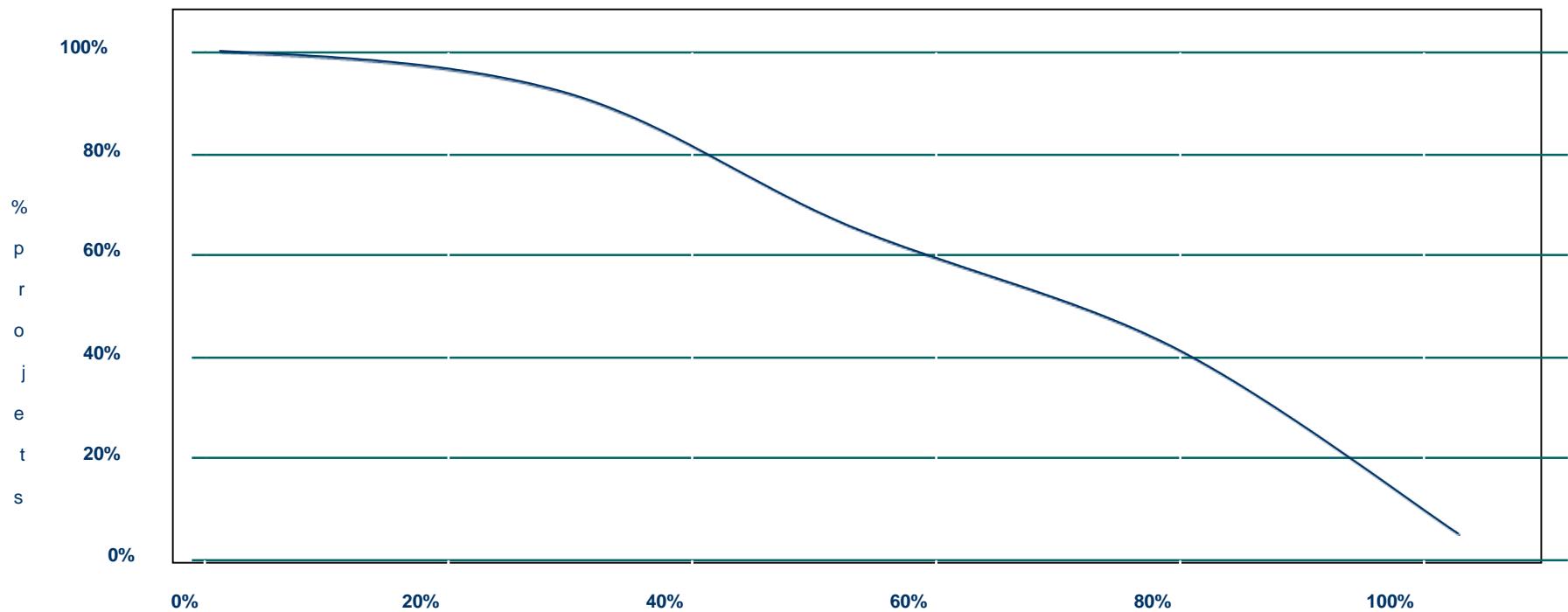
## La crise du logiciel en chiffre :



- **Type 1:** Projets réussis à temps dans le budget prévu et réalisant les fonctionnalités demandées
- **Type 2:** Projets terminés mais hors budget, hors planning et n'offrant pas toutes les fonctionnalités requises
- **Type 3:** Projets abandonnés en cours de réalisation

# La crise du logiciel en chiffre :

## Fonctionnalités réellement disponibles



Source :The Standish Group International, Inc.

USA, 2014

365 organisations, 8380 projets

Rapport disponible sur internet:

<http://www.standishgroup.com/chaos.html>

# Cycle de vie de développement logiciel

## DÉMARCHE DE DÉVELOPPEMENT

« Un système complexe qui fonctionne a toujours évolué à partir d'un système simple qui a fonctionné...

Un système complexe conçu à partir de zéro ne fonctionne jamais et ne peut être rapiécé pour qu'il fonctionne.

Il faut tout recommencer, à partir d'un système simple qui fonctionne. » - John Gall

# **DÉMARCHE DE DÉVELOPPEMENT**

Une démarche de développement repose sur :

- **Un formalisme ( exemple : orientation objet )**
- **Une méthode (SADT , FUSION, Merise ,  
OMT , RUP / UML , ... )**
- **Un processus et Un cycle de vie**

# Qu'est ce qu'une méthode ?

- Définition (**Démarche**)
  - guide plus ou moins formalisé, démarche reproductible permettant d'obtenir des solutions fiables à un problème
  - capitalise l'expérience de projets antérieurs et les règles dans le domaine du problème
- Une méthode définit (**Des raisonnements et des techniques**)
  - des **concept**s de modélisation (obtenir des modèles à partir d'éléments de modélisation, sous un angle particulier, représenter les modèles de façon graphique)
  - une **chronologie** des activités (→ construction de modèles)
  - un ensemble de **règles et de conseils** pour tous les participants
- Description d'une méthode
  - des gens, des activités, des résultats

## 1ère génération

### Modélisation par les fonctions

- Approche dite « cartésienne »
- Décomposition d'un problème en sous-problèmes
- Analyse fonctionnelle hiérarchique : fonctions et sous-fonctions
  - avec fonctions entrées, sorties, contrôles (proche du fonctionnement de la machine)
  - les fonctions contrôlent la structure : si la fonction bouge, tout bouge
  - données non centralisées
- Méthodes de programmation structurée
  - IDEF0 puis SADT
- Points faibles
  - focus sur fonctions en oubliant les données, règles de décomposition non explicitées, réutilisation hasardeuse

## 2ème génération

# Modélisation par les données

- Approches dites « systémiques »
- SI = structure + comportement
- Modélisation des données et des traitements
  - privilégie les flots de données et les relations entre structures de données (apparition des SGBD)
  - traitements = transformations de données dans un flux (notion de processus)
- Exemple : MERISE
  - plusieurs niveaux d'abstraction
  - plusieurs modèles
- Points forts
  - cohérence des données, niveaux d'abstraction bien définis.
- Points faibles
  - manque de cohérence entre données et traitements, faiblesse de la modélisation de traitement (mélange de contraintes et de contrôles), cycles de développement trop figés (cascade)

## génération actuelle

### Modélisation orientée-objet

- Mutation due au changement de la nature des logiciels
  - gestion > bureautique, télécommunications
- Approche « systémique » avec grande cohérence données/traitements
- Système
  - ensemble d'objets qui collaborent
  - considérés de façon statique (ce que le système est : données) et dynamique (ce que le système fait : fonctions)
  - évolution fonctionnelle possible sans remise en cause de la structure statique du logiciel
- Démarche
  - passer du monde des objets (du discours) à celui de l'application en complétant des modèles (pas de transfert d'un modèle à l'autre)
  - à la fois ascendante et descendante, récursive, encapsulation
  - abstraction forte
  - orientée vers la réutilisation : notion de composants, modularité, extensibilité, adaptabilité (objets du monde), souples
- Exemples : nombreux à partir de la fin des années 80

# processus

## Notion de processus de développement :

Un processus de développement représente un ensemble d'étapes ou activités successives permettant la production d'un système logiciel dans les délais ( et le budget ) fixés par le client et répondant aux besoins des utilisateurs .

## Étapes d'un processus de développement

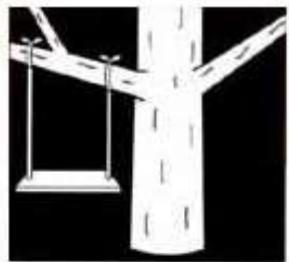
- **Spécification** : établissement du cahier des charges et des contraintes du système ( capture des besoins )
- **Analyse** : Détermination des éléments constituant le système
- **Conception** : Production d'un modèle du système final tel qu'il doit fonctionner
- **Implémentation** : réalisation du système ( codage des composants et assemblage )
- **Test** : Vérification de l'adéquation entre les fonctionnalités du système et la description des besoins
- **Installation** : livraison du système au client et vérification de son fonctionnement .
- **Maintenance** : réparation des fautes dans le système au fur et à mesure de leur découverte .

## cycle de vie

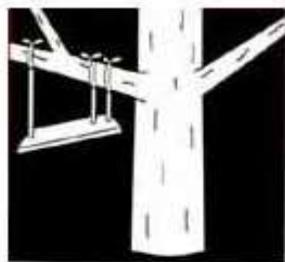
- On entend, par un cycle de vie, toutes les phases de développement du logiciel, de l'établissement des besoins du client jusqu'à l'achèvement du logiciel en tant que produit commercial
- La notion de cycle de vie n'est pas spécifique au logiciel, mais le cas du logiciel est assez particulier

# De main en main

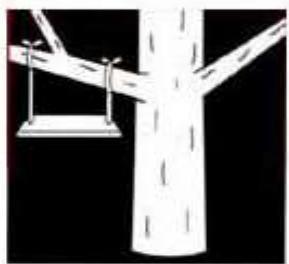
- Ce qu'il fallait
- Ce que l'utilisateur demande
- Ce qui est écrit dans le cahier de charge
- Ce que l'analyste a compris
- Ce que le programmeur a réalisé
- Après la mise au point



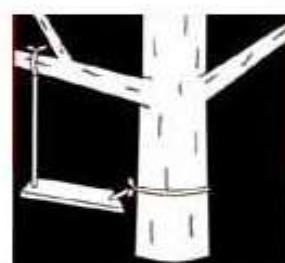
*Ce que voulait  
le client...*



*Ce qu'a compris  
le chef de projet...*



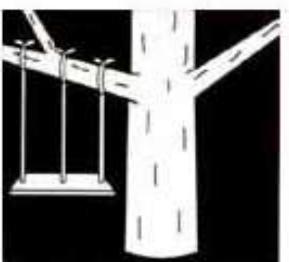
*Ce qu'a spécifié  
le client...*



*Ce qu'a compris  
le concepteur...*



*Ce qui fonctionne  
actuellement :  
Version 1.2.5 + patch*



*Ce qu'a promis  
l'ingénieur commercial*

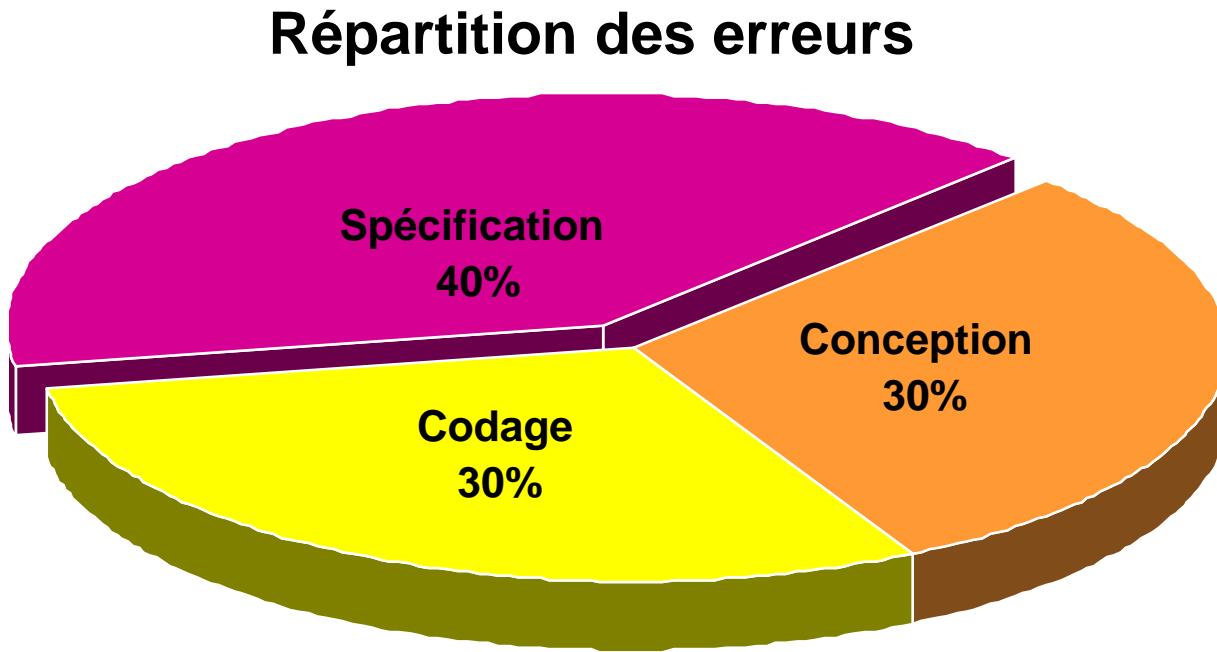


*Ce qui a été livré :  
Version 1.0*

# Le grand risque

- Les défauts apparaissent lors de l'exploitation du logiciel
- la découverte d'une erreur lors des phases avancées de développement
- Le logiciel, certes bien, fait quelque chose, mais peut-être pas, ce qu'il fallait !!!!!
- régression devient très coûteuse
- coût de correction élevé

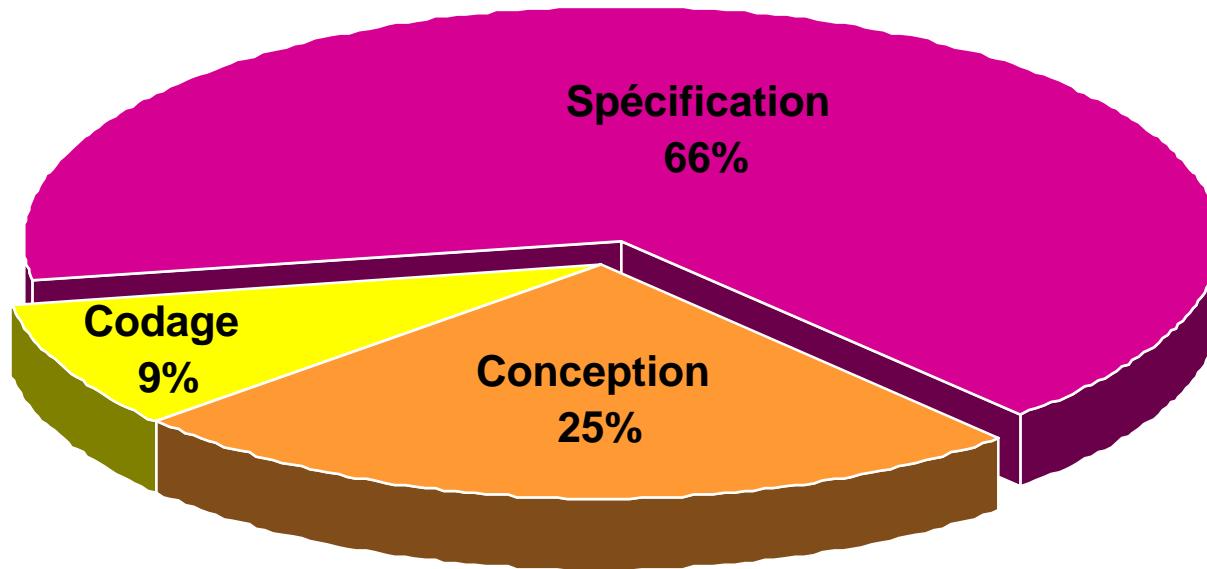
## La crise du logiciel en chiffre :



**Source** : L'assistant à la maîtrise d'ouvrage dans la démarche de sélection logicielle  
Présentation du Marc.Krystkowiak de CRP Henri Tudor Luxer  
19 mai 2004  
Chambre de Commerce de Luxembourg

## La crise du logiciel en chiffre :

### Coût de correction des erreurs



**Source** : L'assistant à la maîtrise d'ouvrage dans la démarche de sélection logicielle  
Présentation du Marc.Krystkowiak de CRP Henri Tudor Luxer  
19 mai 2004  
Chambre de Commerce de Luxembourg

## Le grand souci

- Réduire la distance entre les premières phases de développement et la phase de validation (vérification externe) qui fait intervenir le client/utilisateur

# Facteurs transcendants

## Facteur humain

- Quelque soit le cycle de développement choisi, La réussite d'un projet dépend en grande partie de la qualité des personnes qui travaillent.
- Laisser l'équipe se constituer : des gens qui se connaissent et s'apprécient travaillent mieux ensemble
- Mettre en place des primes ou des compensations
- L'architecte logiciel doit programmer : pour garder sa crédibilité
- Faite tourner la responsabilité : augmente la cohésion, améliore la communication, équilibre les charges.

# Facteurs transcendants

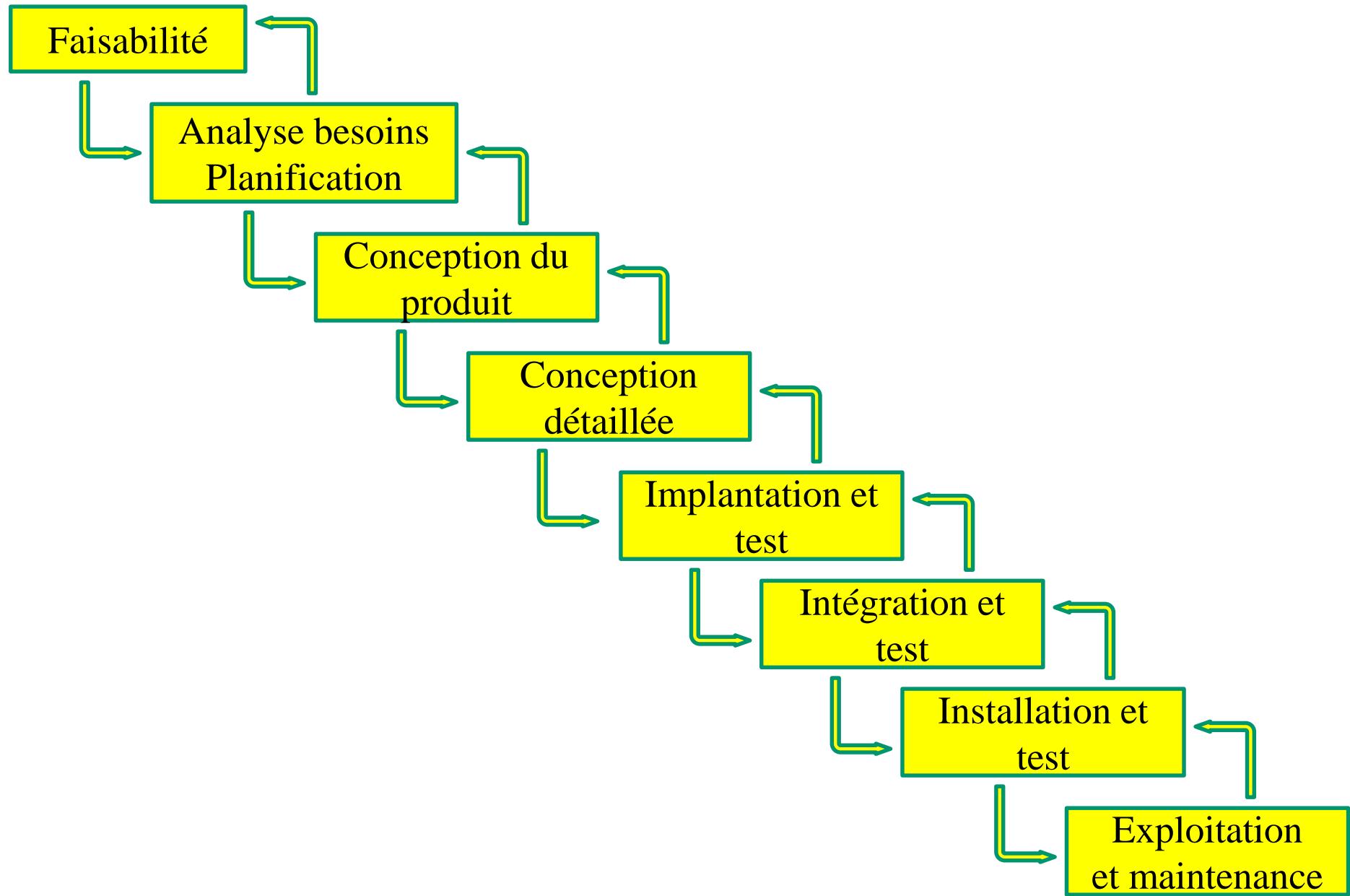
## Estimation et planification

- PERT : converger sur des dates au plus tôt et au plus tard
- Oracle : estimer par consensus, s'adresser à des experts
- Analogie : comparer avec des projets antérieurs
- Planning pocker : tous les membres d'équipe y participent

# Les principaux modèles de cycle de vie

## Le modèle en cascade

- Le modèle en cascade définit des étapes (ou phases) durant lesquelles les activités de développement se déroulent
- Une étape doit se terminer à une date donnée par la production de certains logiciels ou documents
- L'étape suivante n'est abordée que si les résultats sont jugés satisfaisants
- Les résultats de l'étape sont soumis à une étude approfondie



# Faisabilité (pourquoi ?)

Répondre aux questions

- Pourquoi faut-il réaliser ce logiciel ?
- Y a-t-il de meilleures alternatives ?
- Benchmark, étude orientée vers la comparaison de performances
- Le logiciel sera-t-il satisfaisant pour les utilisateurs ?
- Y a-t-il un marché pour le logiciel ?
- A-t-on le budget, le personnel, le matériel nécessaires ?

# Analyse des besoins (quoi ?)

## Analyse des besoins (quoи ?)

- Définir précisément les fonctions que le logiciels doit réaliser/fournir
- Le résultat de cette phase est le cahier des charges du logiciel

## Conception (comment ?)

- Définir la structure du logiciel
- Les résultats comprennent l'architecture du logiciel (décomposition en modules) et la spécification des interfaces des modules
- La définition des algorithmes de chacune des procédures des modules est appelée la conception détaillée du logiciel

# Implantation et test (comment ?)

- Implantation et test (comment ?)
  - Implanter les procédures des modules
  - Tests unitaires
- Intégration et test
  - Intégrer les différents modules
  - Valider / vérifier l'adéquation de l'implantation, de la conception et de l'architecture avec le cahier des charges (acceptation)

# Installation et test

- Installation et test
  - Déploiement du logiciel chez le client et tests avec un sous ensemble d'usager choisi
- Exploitation et maintenance
  - Utilisation en situation réelle, retour d'information des usagers, des administrateurs, des gestionnaires...
  - Maintenance corrective, perfective et adaptative

# Problèmes du modèle en cascade

- Découpage rigide du projet en étapes distinctes
- difficile de s'adapter aux changements des besoins utilisateurs
- Modèle bien adapté si les spécifications peuvent être précises dès le début
- Toutefois, il est rare d'avoir des spécifications stables
- Les tests sont prévus tardivement

# Quels sont les avantages ?

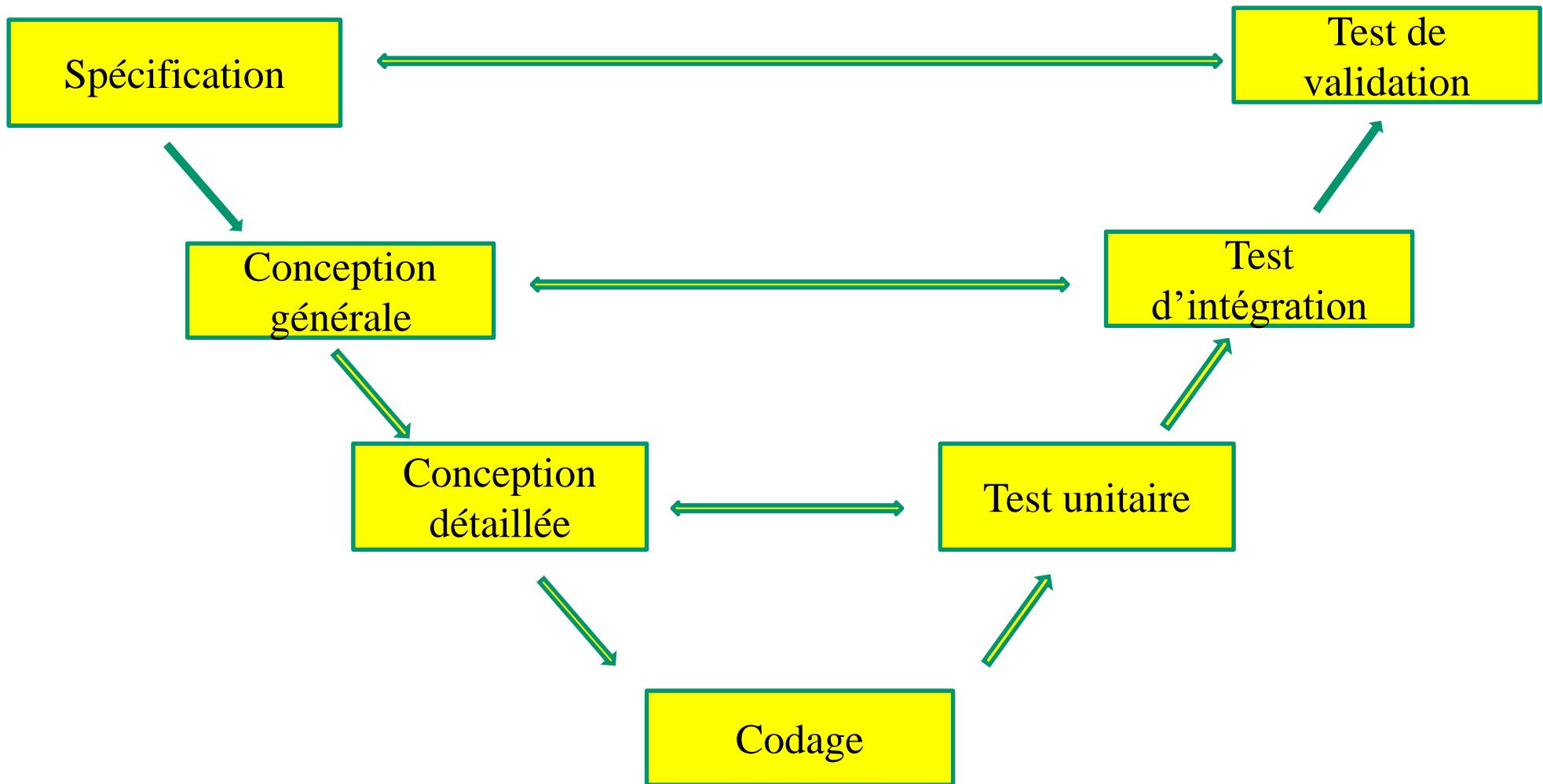
- Simple
- Logique
- Facilité de planification des étapes et des délais
- Contrôle facile
- Accent sur la documentation et la structure
- Idéal pour les projets logiciels stables

# Les inconvénients du modèle

- Absence de flexibilité
- Incapacité de revenir en arrière
- Nécessite une phase de conception parfaite
- De moins en moins de droit à l'erreur avec l'avancement du projet, notamment lors des tests
- Impossibilité de changer les besoins en cours de projet
- Difficulté d'évaluer le temps pour chaque partie sans les avoir entamées
- Les projets utilisant ce modèle ont tendance à être abandonnés
- Modèle trop rigide : une activité ne peut commencer que si l'activité précédente est complètement terminée
- Pas de possibilité de travail en parallèle et validation finale tardive

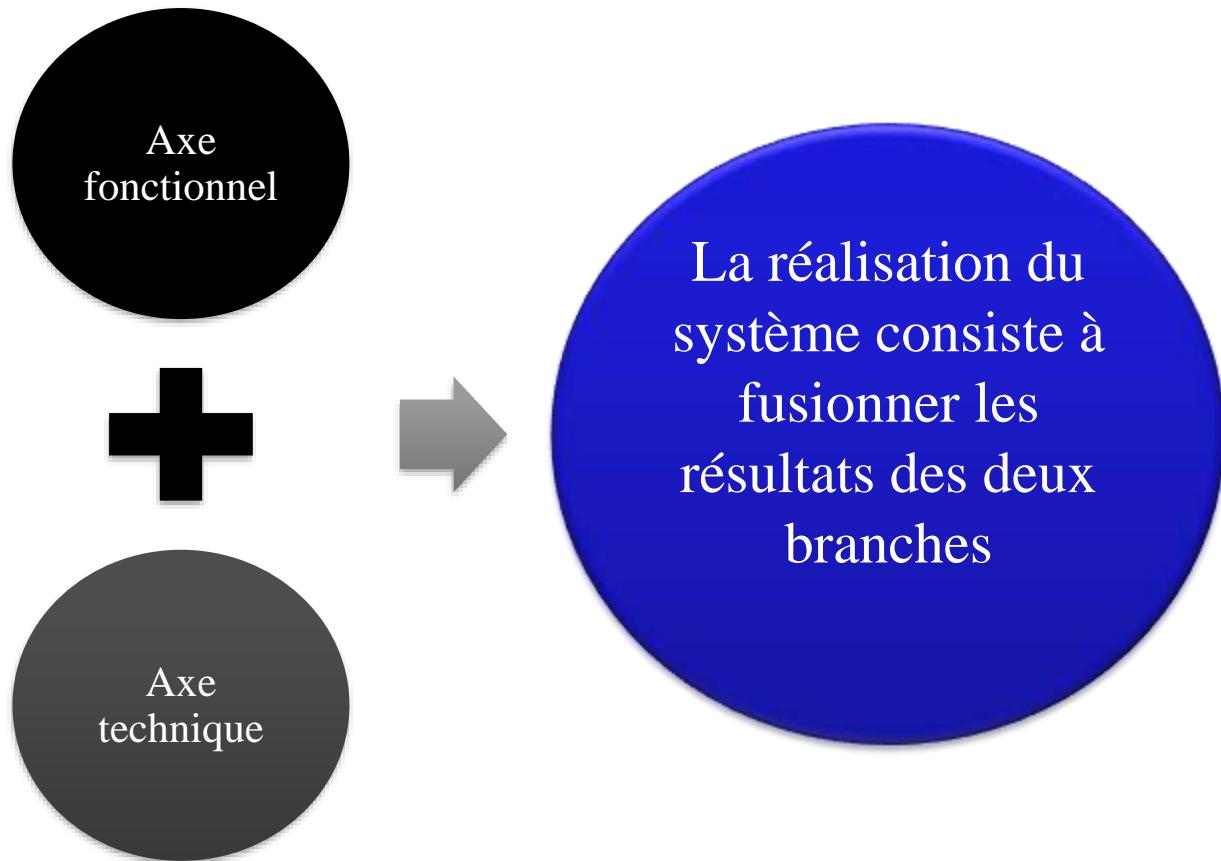
# Cycle en V

- Le modèle du cycle en V est un modèle de gestion de projet imaginé suite au problème du modèle en cascade.
- Son apport : Il permet de limiter les retours aux étapes précédentes.
- Mettre en aval, certaines tâches tardives : intégration, validation

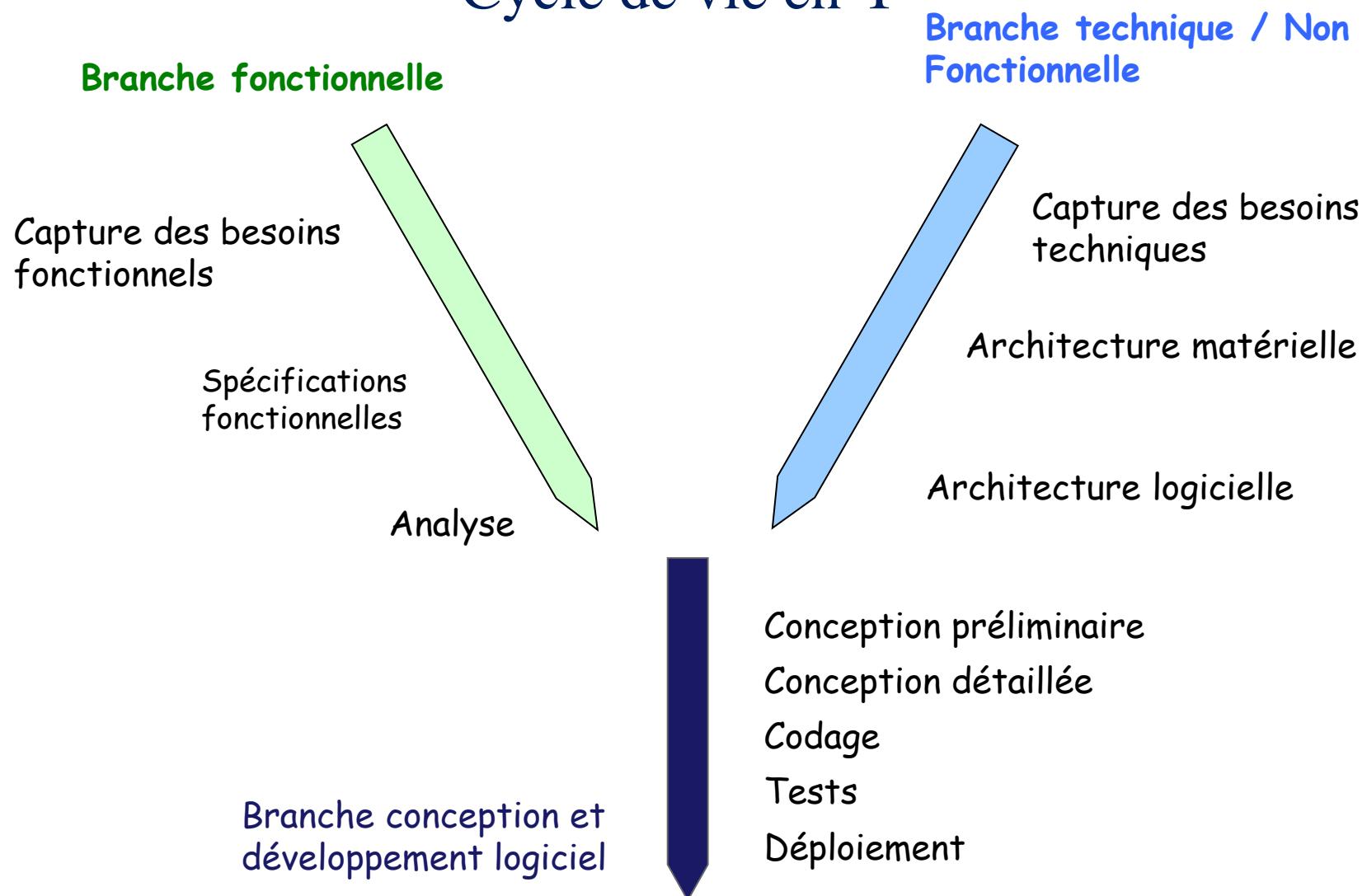


- Tâches effectuées en parallèle
  - Horizontalement : préparation de la vérification
  - Ex. : dès que la spécification fonctionnelle est faite :
    - ✓ plan de tests de validation
    - ✓ plan d'évaluation des performances
  - Les cas de tests sont élaborés (plans de test) lors de la partie descendante d'un cycle en V, en parallèle des phases de spécification, conception, et de codage. L'exécution des tests se fera dans chacune des phases de la partie remontante du cycle, sur la base des cas de tests élaborés dans la phase jumelle de la partie descendante. On parle alors de test unitaire, de test d'intégration, de test système (ou de test de conformité).

# Modèle en Y



# Cycle de vie en Y



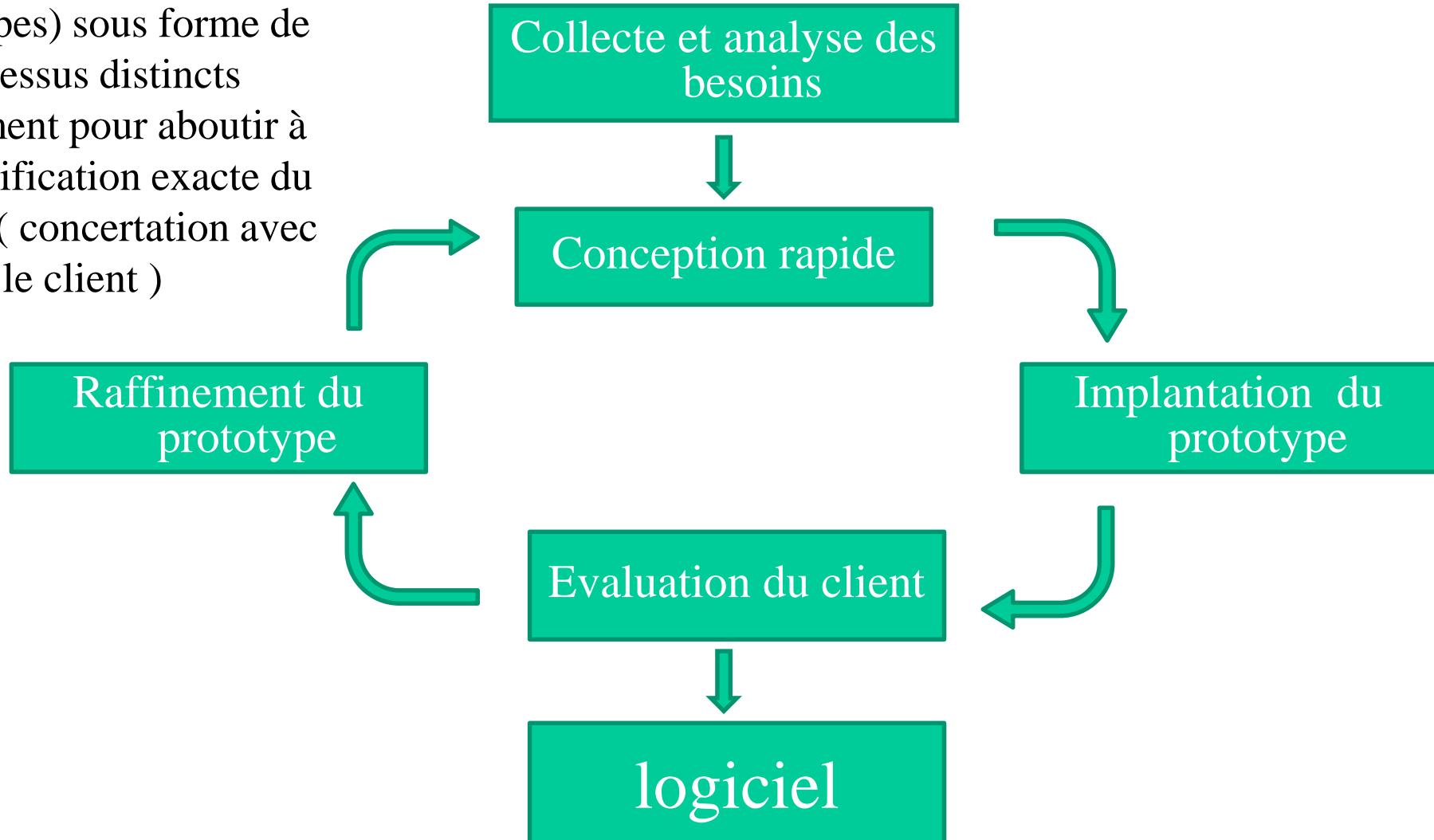
Le 2TUP (2 tracks unified process) propose un cycle de développement en Y itératif.

# Le modèle par prototypage

- Le modèle en cascade suppose que les besoins sont clairs, arrêtés et bien définis
- Le modèle par prototypage est intéressant
  - Besoins pas clairement définis
  - Besoins changeant au cours du temps
- Le prototypage permet le développement rapide d'une ébauche du futur logiciel
  - Prototype jetable
  - Prototype évolutif

# Le modèle par prototypage

Principe : On effectue une ou plusieurs itérations (prototypes) sous forme de processus distincts uniquement pour aboutir à une spécification exacte du système ( concertation avec le client )



# Avantage du modèle

On arrive à satisfaire le client après avoir testé différents prototypes du système

- Meilleure adaptation aux changements
- Mise en œuvre d'une analyse de risques
- Ajustement des choix techniques et fonctionnels tôt dans le processus

## Inconvénients :

Approche très coûteuse en terme de durée de développement ( elle nécessite des ressources humaines importantes )

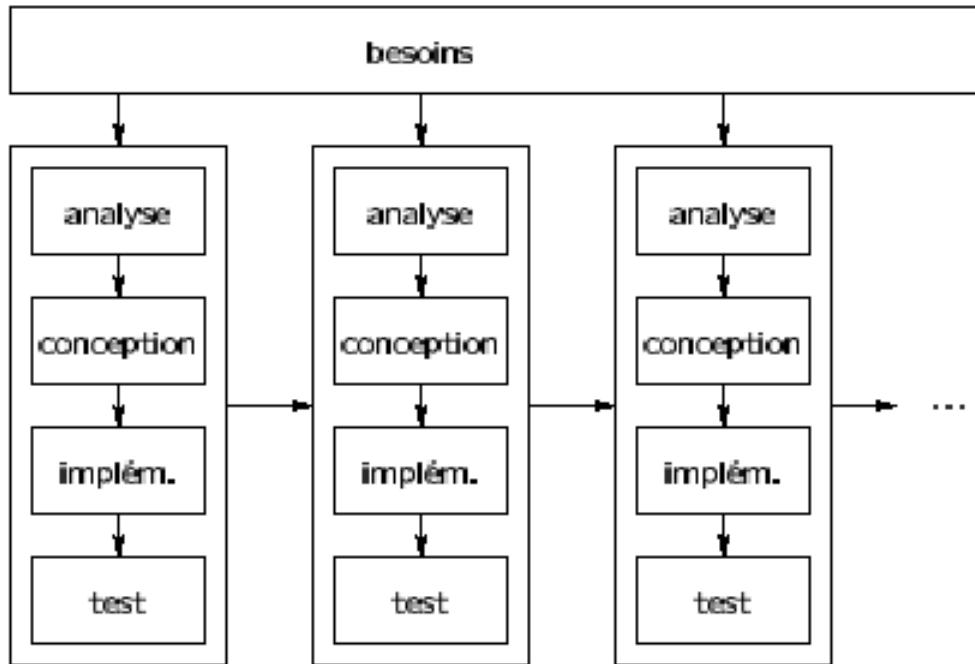
# Modèle incrémental ou itératif

## Principe :

- Après l'analyse des besoins ( spécification ) , on effectue plusieurs cycles de développements successifs ( itérations )
- Chaque cycle conduit à une version utilisable du produit et traite un petit nombre de besoins

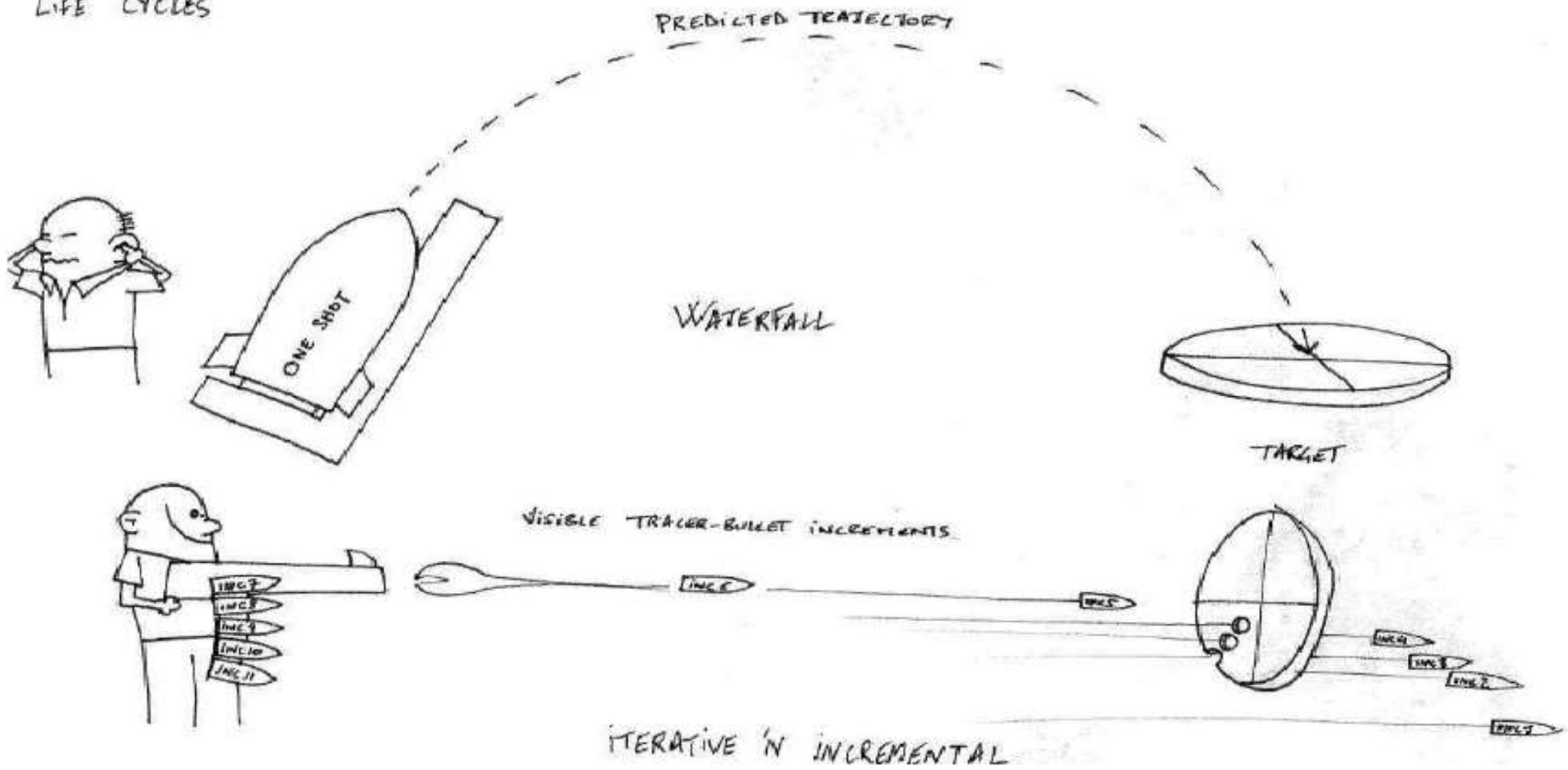
## Avantages :

- A chaque itération , la complexité du système diminue
- Des implémentations de parties du système sont disponibles très rapidement . On peut disposer alors d'informations utiles sur le comportement du système ( early feedback ) et son adéquation aux besoins .
- Ce modèle permet le travail en parallèle des équipes





## LIFE CYCLES



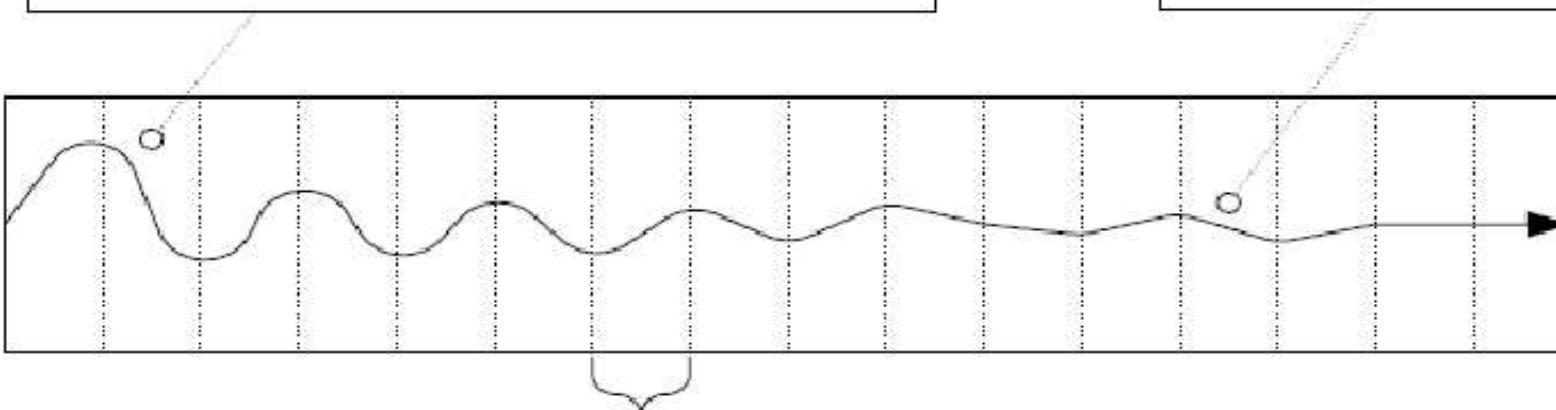
# Modèle incrémental ou itératif



L'intégration du changement tout le long du processus itératif ne signifie pas que le développement est incontrôlé et réactif, influencé par des « glissements ». Au contraire, la gestion intégrée du changement permet une convergence et stabilisation du système.

Les premières itérations sont loin de la « bonne voie ». Grâce au feedback et à l'adaptation, le système converge vers les spécifications et la conception les plus appropriés.

Lors des itérations ultérieures, un changement significatif des spécifications est rare mais peut survenir. De telles modifications tardives peuvent procurer à une entreprise un avantage concurrentiel.

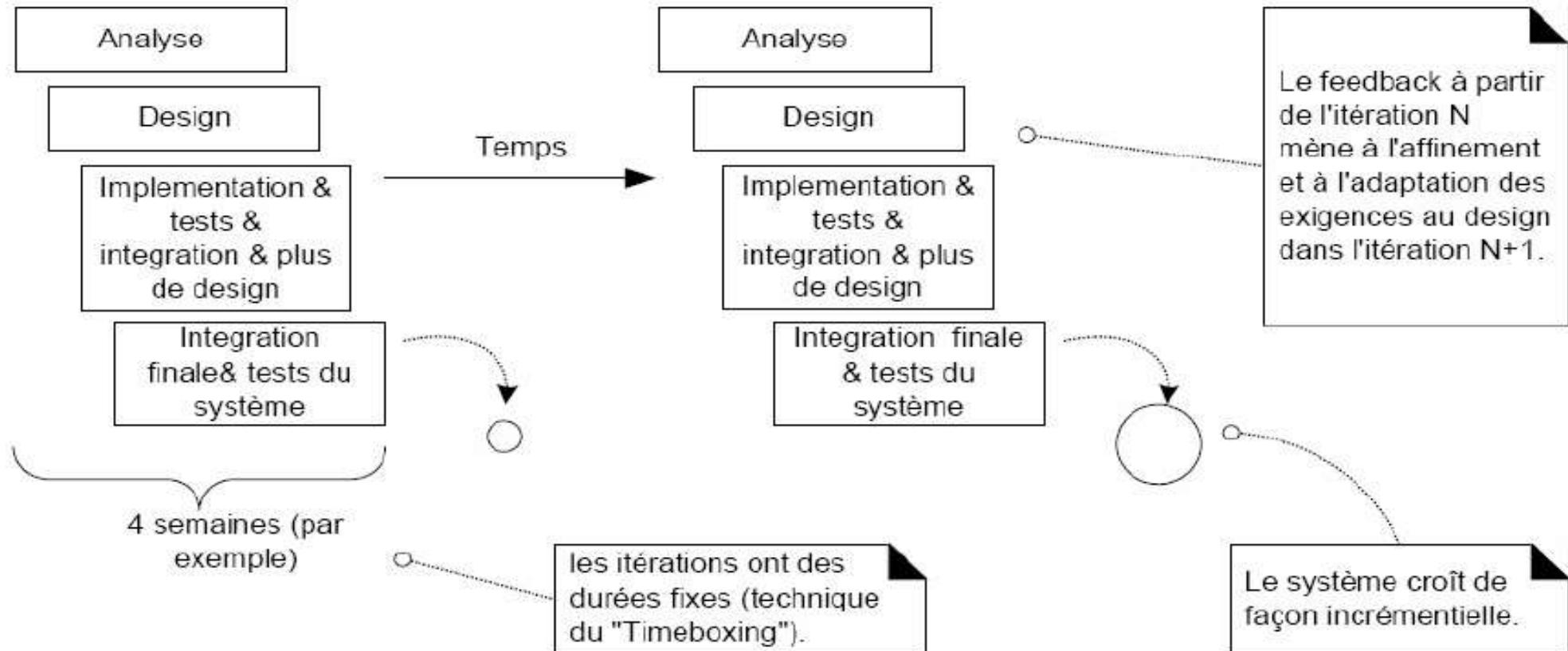


Une itération: exigences, conception, implémentation, intégration et tests.

# Modèle incrémental ou itératif



## Les itérations :



## **4 -Qualité du logiciel et Sûreté de fonctionnement**

### **Définition et mesure de la qualité à atteindre**

La qualité est définie de manière générale par « l'aptitude d'un produit ou d'un service à satisfaire les besoins des utilisateurs ». Cette définition s'applique aux systèmes informatiques.

La difficulté principale réside dans la capacité à exprimer ces besoins. On constate trop souvent que l'utilisateur n'est pas satisfait *a posteriori* ! Certes le cycle de vie en spirale, ou des approches par maquettage réduisent ces risques, mais, il est fondamental de permettre à l'utilisateur (et/ou au client) d'exprimer ces exigences; ce sur quoi il va juger la qualité du système.



La qualité est un processus qui dépasse la notion de logiciel. On peut l'appliquer à tout processus industriel pour peu qu'on souhaite l'améliorer. Améliorer quoi ? le produit, le service, la façon de développer le produit, les délais, les coûts, etc. Peu importe, ce qui compte c'est la volonté de mesurer, observer, contrôler, apprendre pour améliorer quelque chose.

Les premières études systématique de la qualité des logiciels datent de 1977. Depuis des modèles de qualités se sont développés, mais on retrouve toujours 3 niveaux :

- Les **facteurs** qualité : expression des exigences (point de vue externe, client)
- Les **critères** qualité : caractéristiques du produit (point de vue interne, technique)
- Les **métriques** : ce qui permet de mesurer un critère

Les facteurs de qualité du logiciel sont les suivants :

1. Conformité : satisfaire aux spécifications - il faut qu'elles existent !
2. Robustesse : ne pas tomber en panne (tolérance aux pannes, recouvrement d'erreurs, etc.)
3. Efficacité : optimisation des ressources (cpu, E/S, mémoire, etc.)
4. Sécurité : surveiller, contrôler, interdire les accès
5. Maniabilité: minimiser l'effort d'apprentissage de l'utilisation du système
6. Maintenabilité : minimiser l'effort pour localiser et corriger les fautes
7. Testabilité : minimiser, automatiser l'effort de test
8. Adaptabilité : minimiser l'effort d'évolution du système
9. Portabilité : minimiser l'effort pour changer de plate-forme
10. Réutilisabilité: optimiser la conception pour faciliter la réutilisation de parties du système
11. Interopérabilité: garantir l'ouverture du système à d'autres systèmes Ils doivent être appréciés par le client pour définir les points qu'il juge capitaux ou secondaires.

Les techniques mises en œuvre couvrent les critères qualité. On y retrouve les « bons » principes de programmation comme

- la modularité,
- l'auto-descriptivité (les commentaires),
- l'indépendance matérielle logiciel,
- la concision, etc.

Mais aussi des « bons » principes d'organisation comme

- l'utilisation de standards
- la **träçabilité**. Cette dernière est la capacité à suivre dans le temps l'évolution d'un produit et de ses plans et de pouvoir associer les éléments de la solutions aux éléments du problème.

Pour mesurer la qualité du logiciel, des métriques sont associés aux critères eux même rattachés aux facteurs. Ces métriques peuvent caractériser les qualités :

- · du produit
- · du processus de développement
- · du service rendu

Elles peuvent se faire par des mesures objectives (**comptages**) ou par des enquêtes d'opinion (« pensez-vous que les résultats soient présentés clairement ? - de **0 à 5** »). Les exemples des métriques concernant le logiciel sont :

- · nombre de lignes de code, de fonctions, d'opérateur par fonction,... (concision)
- · nombre de lignes de commentaires, ... (auto-descriptivité)
- · nombre des modules, nombre de liens avec d'autres modules - couplage (modularité)
- · nombre de chemins possibles dans une fonction (simplicité)
- · nombre de données en entrées, en sortie, fréquence...(simplicité)

# **Qualification des facteur**

Exigences

Facteur	Sous-rubrique	faible	moyenne	forte
<b>Efficacité</b>	occupation mémoire	< 50%	>50%	>75%
	mémoire auxiliaire	< 50%	>50%	>75%
	occupation lignes	< 50%	>50%	>75%
	charge calcul	< 50%	>50%	>75%
	% avec contr. durée	< 20%	<50%	>50%
<b>Maniabilité</b>	IHM	Non	peu imp.	imp.
	utilisateur	infor.	techn.	public
	résultats formatés	Non		Oui
	Aide en ligne	Non		Oui
<b>Robustesse</b>	reprise ap. coupure secteur	Non	A froid	A chaud
	protec. vs pannes	Non.		Oui
	contr. validité données	Non	Partielle	Oui
	Redondance	Non		Oui

Facteur de qualité : aptitude du logiciel à	Note 
<b>Adaptabilité</b> : minimiser l'effort nécessaire pour le modifier par suite d'évolution des spécifications	
<b>Conformité</b> : contenir un minimum d'erreurs, à satisfaire aux spécifications et à remplir ses missions dans les situations opérationnelles définies.	
<b>Efficacité</b> : se limiter à l'utilisation des ressources strictement nécessaires à l'accomplissement de ses fonctions.	
<b>Maintenabilité</b> : minimiser l'effort pour localiser et corriger les fautes.	
<b>Maniabilité</b> : minimiser l'effort nécessaire pour l'apprentissage, la mise en œuvre des entrées et l'exploitation des sorties.	
<b>Réutilisabilité</b> : être partiellement ou totalement utilisé dans une autre application.	
<b>Sécurité</b> : surveiller, recenser, protéger et contrôler les accès au code et aux données ou fichiers.	
<b>Robustesse</b> : accomplir sans défaillance l'ensemble des fonctionnalités spécifiées, dans un environnement opérationnel de référence et pour une durée d'utilisation donnée.	
<b>Interopérabilité</b> : s'interconnecter à d'autres systèmes.	
<b>Portabilité</b> : minimiser l'effort pour se faire transporter dans un autre environnement matériel et/ou logiciel.	
<b>Testabilité</b> : faciliter les procédures de test permettant de s'assurer de l'adéquation des fonctionnalités	

# UML

# 1 - Evolution des Méthodes de Conception Orientées Objet

- En 1994, plus de **50 méthodes OO**
  - Fusion, Shlaer-Mellor, ROOM, Classe-Relation, Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS...
- Les **méta-modèles se ressemblent** de plus en plus  
( 1 méta-modèle = modèle de structure d'une méthode )
- Les **notations graphiques** sont toutes **différentes**
- L'industrie a besoin de **standards**

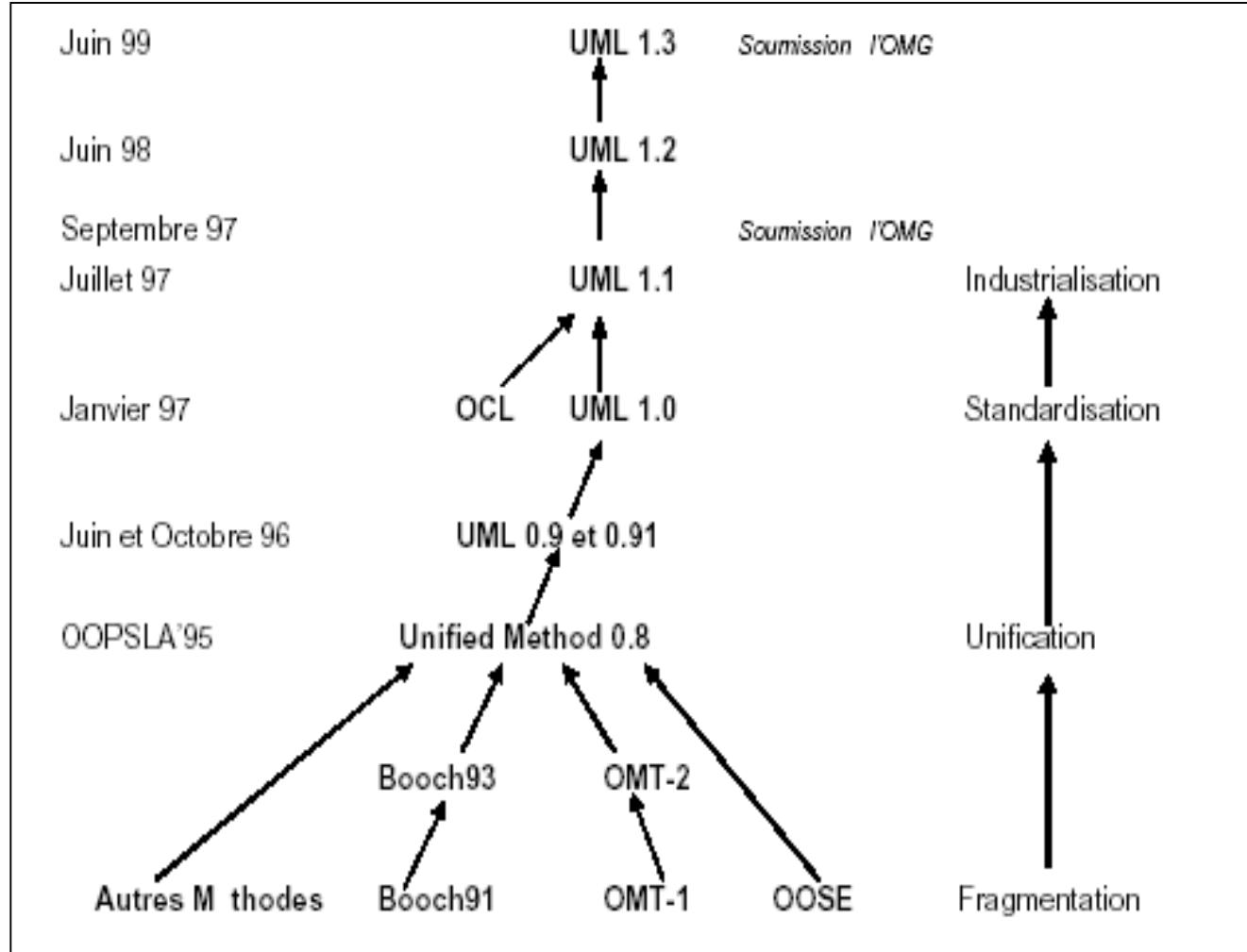
## Naissance d'UML :

- 1993-1994: Les 2 méthodes ( **Booch'93** et **OMT-2** ) deviennent **leaders** sur le marché, elles sont de plus en plus **proches**
- Octobre 1994
  - **J. Rumbaugh** (OMT) rejoint **G. Booch** chez **Rational**
  - Annonce de l'unification des deux méthodes
- Octobre 1995: **Méthode Unifiée** v0.8
- Fin 1995: le fondateur d 'Objectory, **Ivar Jacobson**, rejoint à son tour **Rational**
- Janvier 97 : **Soumission** à l'**OMG** de la version UML 1.0

## **OMG:** Object Management Group

- Organisme à but non lucratif fondé en 1989
  - Plus de 700 entreprises y adhèrent
  - Connue pour la norme CORBA
- Septembre 97 : **UML 1.1**

## 2 – Historique d’UML



- **UML**: Prendre le **mieux** de chacune des 3 méthodes ( **OOSE** (Jacobson): **Use Cases** , **OMT** (Rumbaugh): **Analyse** et **Booch**: **Conception, Architecture** )
- UML a été standardisée par l'OMG dès Janvier 1997 et soutenu depuis par le **marché** ( Microsoft, HP, Oracle, IBM...) et par les universités .

### 3 – Le Processus de développement dans UML

- UML est une **notation, pas une méthode**
- UML est un **langage** de modélisation objet : il convient à **toutes** les méthodes objets et convient à **tous** les langages objets ( C++ , Java , Eiffel , VB , etc... )
- **UML ne définit pas de processus standard de développement**
  - Ce n'est le but ni d 'UML, ni de l 'OMG
- Les processus de développement de Booch, OMT, et OOSE **restent applicables**
- **Rational propose pour UML Objectory et RUP ( Rational Unified Process )**
- RUP = Processus **itératif et incrémental**
  - Segmentation du travail d 'après les cas d 'utilisation, et l 'étude des risques
  - Les premières itérations sont des prototypes
  - Développement incrémental
  - Possibilité de paralléliser les itérations

UML n'est pas propriétaire, elle est accessible à toute entreprise qui peut librement en faire usage. En fonction du type d'application, des compétences ou des contraintes imposées à l'équipe de développement, le processus d'analyse peut-être différent et l'accent peut être mis sur tel ou tel autre modèle.

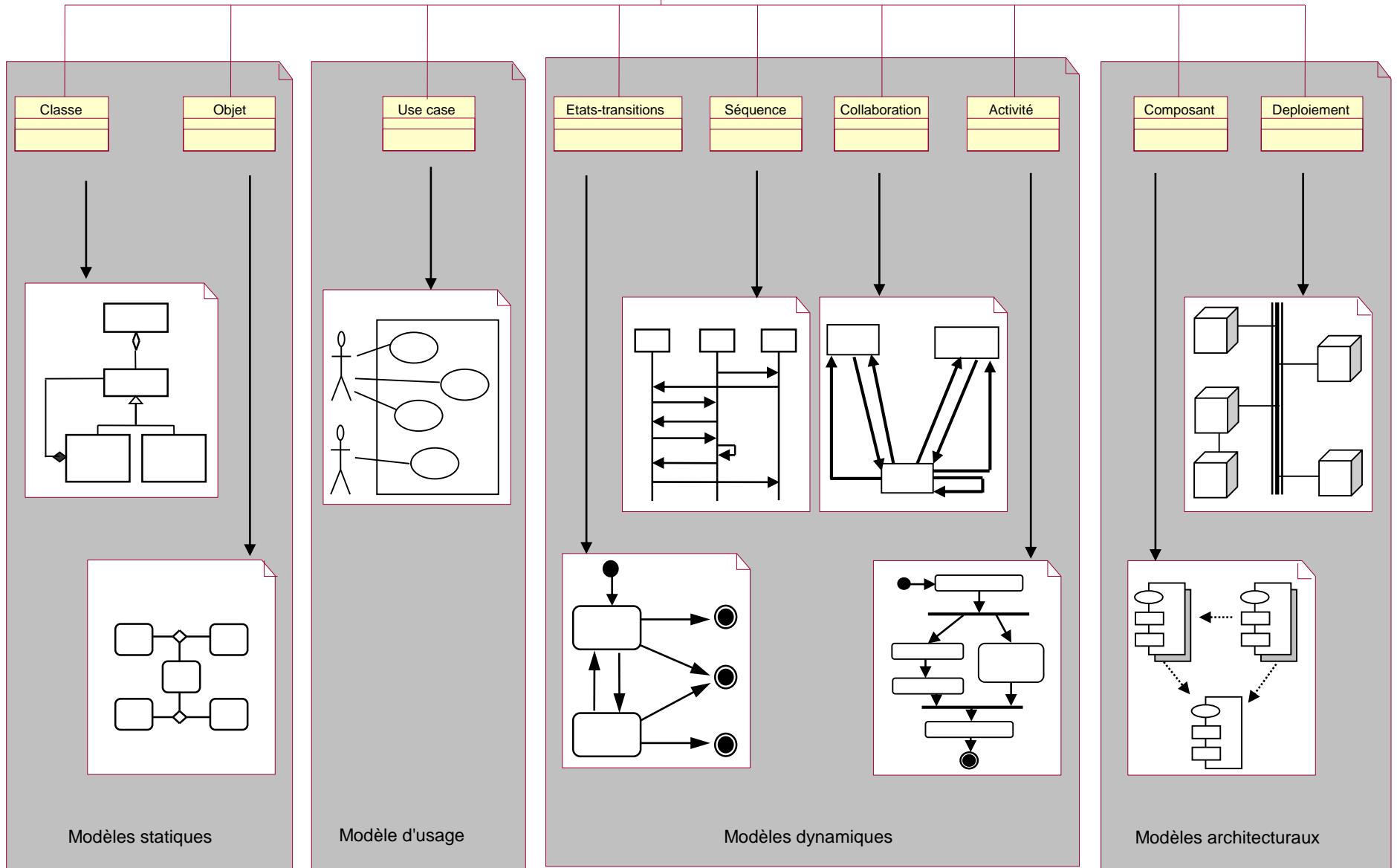
Certains fabricants offrent des ateliers, supportant UML, qui permettent l'élaboration des diagrammes et assurent la cohérence entre les modèles. Des générateurs de code Java, VB ou C++ permettent de passer aisément de l'analyse au codage. Certains intègrent même des outils de "reverse engineering" pour maintenir la cohérence entre le code et la conception.

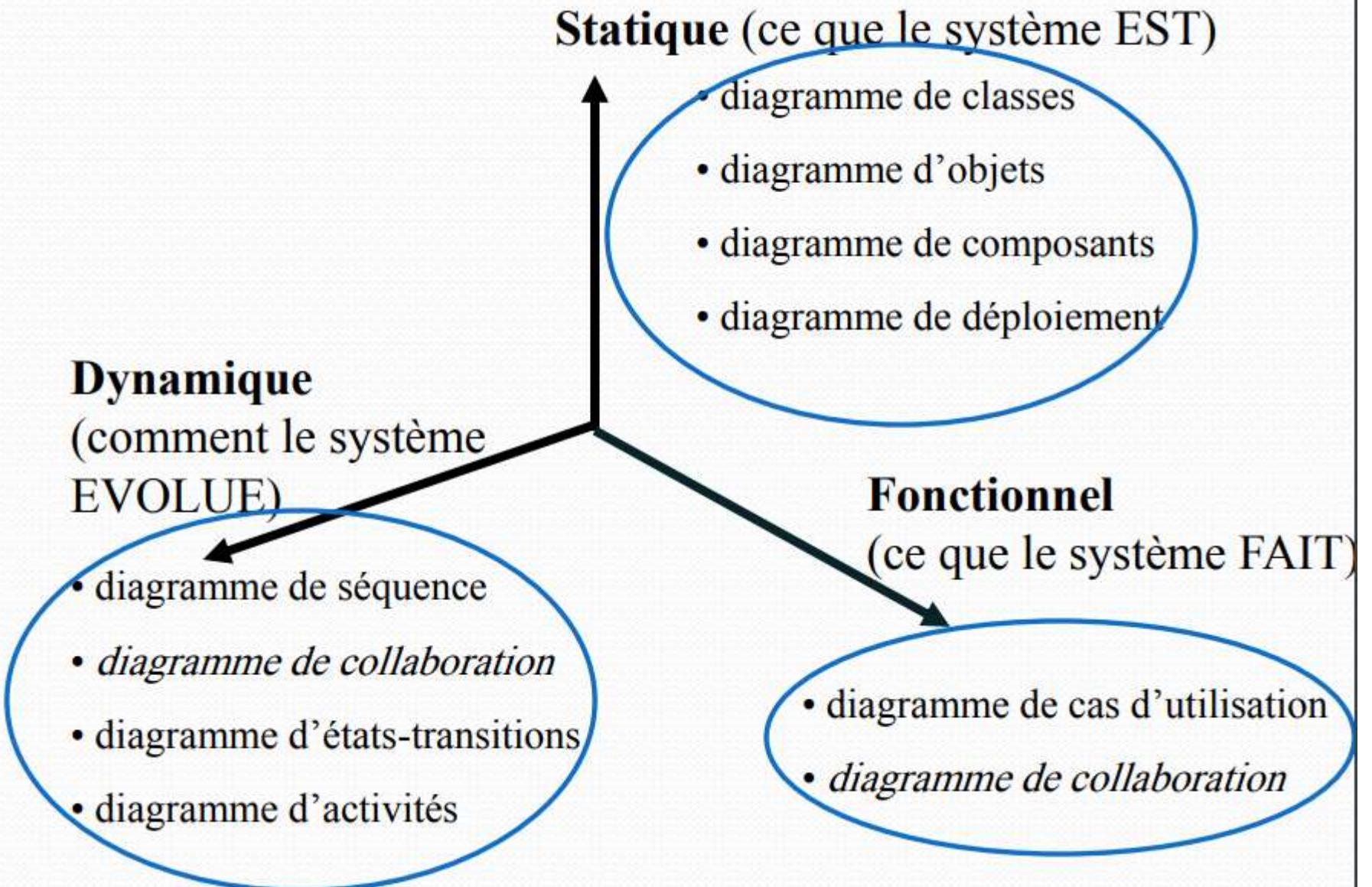
Parmi les offres du marché les principaux sont:

- RATIONAL avec Rational Rose 2001 [www.rational.com](http://www.rational.com)
- SOFTEAM avec Objecteering UML Modeler 5.1 [www.softeam.com](http://www.softeam.com)
- MICROSOFT avec Visual Modeler pour VB [www.microsoft.com](http://www.microsoft.com)
- SAP avec PowerAMC [www.sap.com](http://www.sap.com)
- ArgoUML
- Visual Paradigm
- Plugin Omundo, Eclipse
- Astah

Chaque atelier supportant UML propose un processus d'analyse. Un processus peut être assimilé à une méthode de développement logiciel par le fait qu'il propose une démarche et une chronologie dans l'analyse. Par exemple Rational utilise RUP (*Rational Unified Process*). D'autres entreprises proposent un processus de développement propriétaire basé sur UML, comme Communications & Systems avec *UML/CS SI Development Process*.

Diagramme UML





## **DEMARCHE GENERALE**

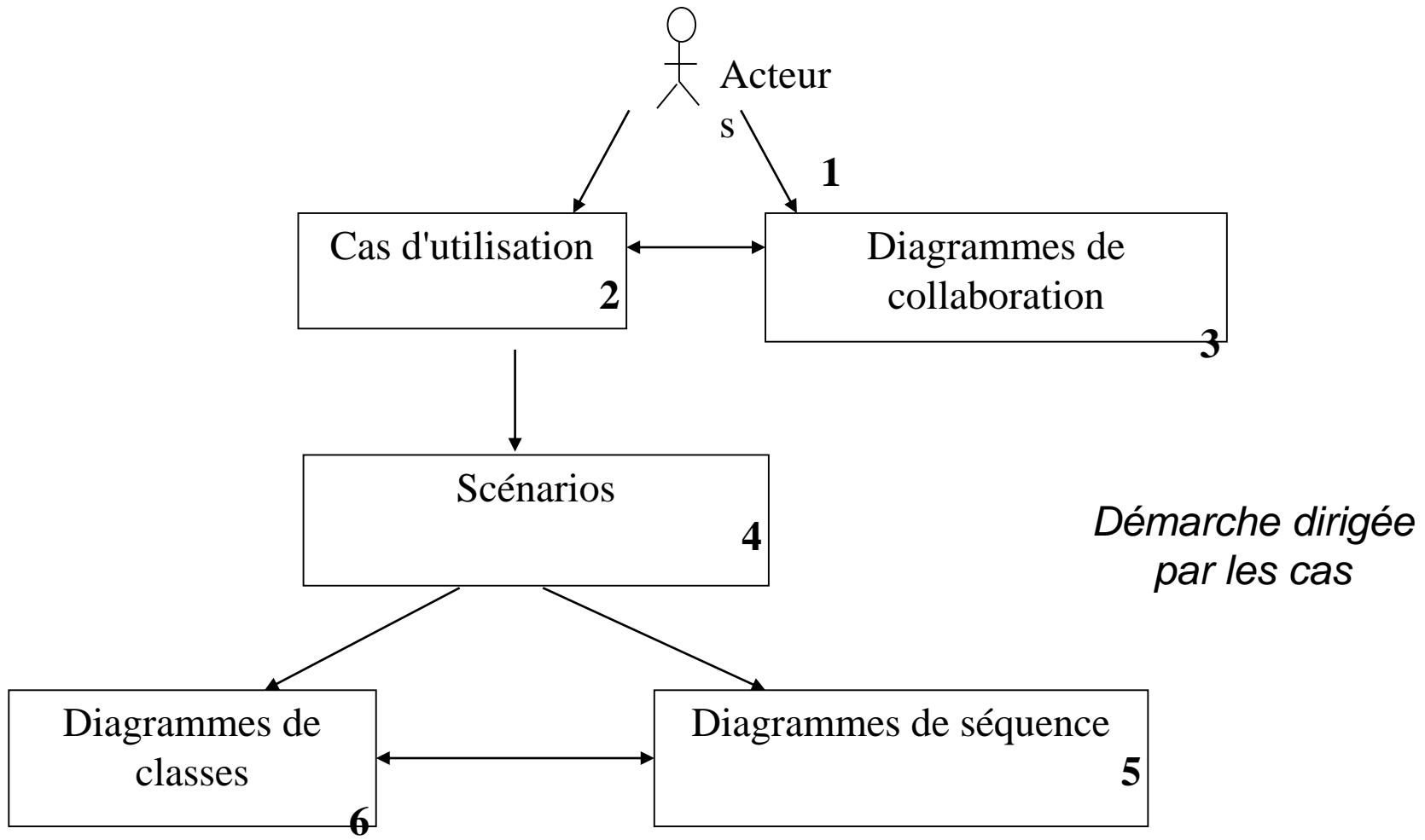
La démarche proposée est parfois appelée "*Approche dirigée par les cas*". Elle consiste en:

### **Vue des cas d'utilisation**

1. Identification des **acteurs** qui agissent sur le système à étudier
2. Définition et description des **cas d'utilisations** qui montreront les interactions entre les acteurs actifs externes et le système vu comme une boîte noire,
3. Construction de **diagrammes de collaboration** entre les acteurs et le système afin de visualiser les échanges d'information,
4. Pour chaque cas d'utilisation seront décrits les principaux **scénarios** qui visualisent la chronologie des échanges entre les acteurs et le système,

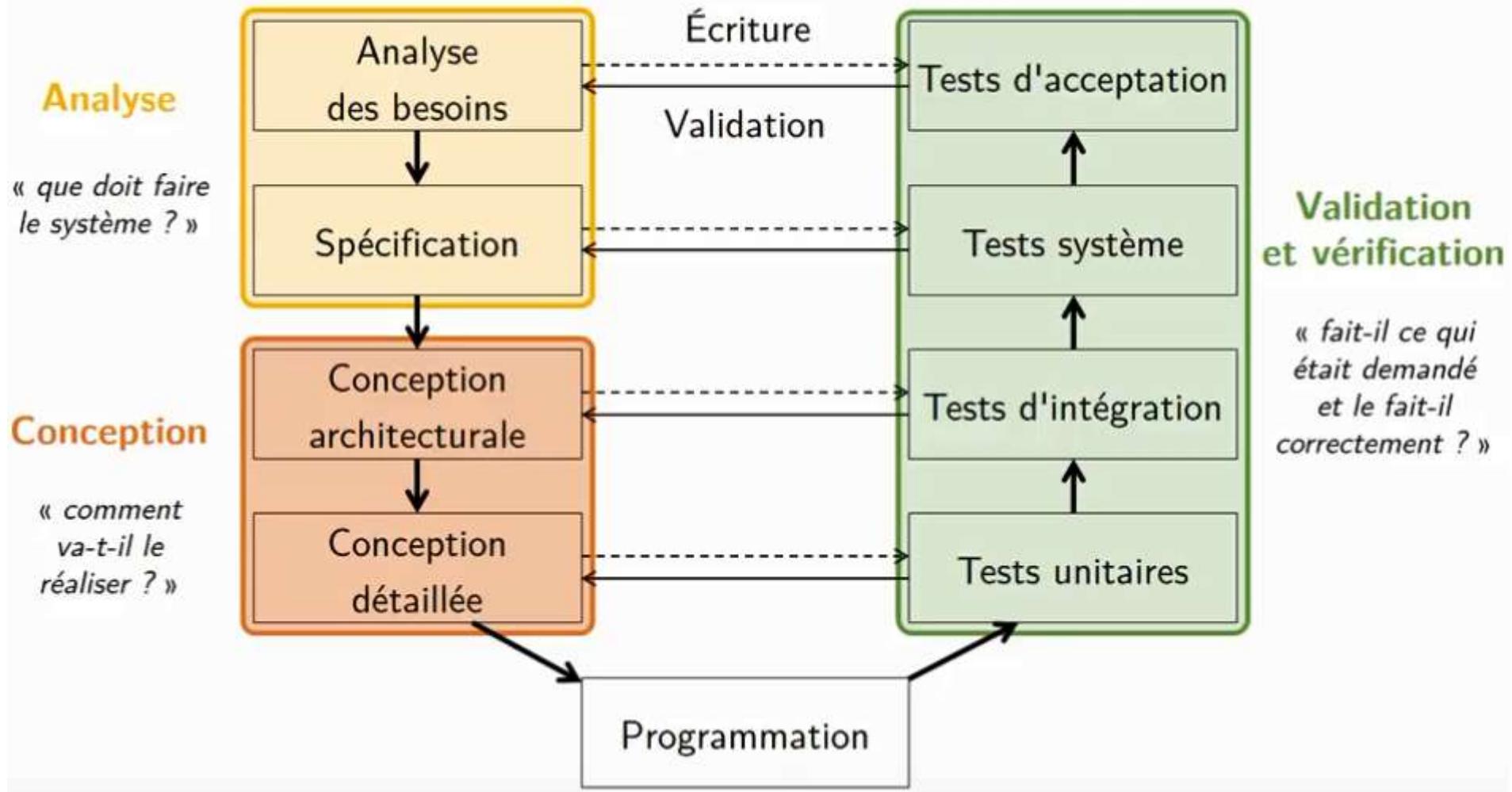
### **Vue logique**

5. A chaque scénario sera associé un **diagramme de séquence** qui mettra en avant l'aspect temporel des interactions entre les objets de l'application qui participent au scénario,
6. En parallèle avec les diagrammes précédent, seront construits les **diagrammes de classes** qui visualisent l'architecture du logiciel, la hiérarchie et les associations entre classes



Une autre démarche est possible, c'est "*L'approche dirigée par les classes*" où sont d'abord identifiées les classes de l'application puis les cas d'utilisation

# Processus de développement en V



# Cas d'utilisation

**Objectif** : Comprendre les **besoins du client** pour rédiger le **cahier des charges fonctionnel**

**Trois questions :**

1. Définir les **utilisations principales** du système : à quoi sert-il ?
2. Définir l'**environnement** du système : qui va l'utiliser ou interagir avec lui ?
3. Définir les **limites** du système : où s'arrête sa responsabilité ?

**Éléments de description :**

- Diagramme de cas d'utilisation
- Description textuelle des cas d'utilisation
- Diagrammes de séquence des **scénarios d'utilisation**

## 4 – Mise en œuvre d'une itération dans le processus de développement UML

**Itération = Spécification + Analyse + Conception + Implémentation + Test**

### 4.1 - La Phase de spécification

Cette phase sert à la capture des besoins du système à réaliser ; elle permet de :

- Prendre connaissance du cahier des charges
- Définir le cadre du système
- Délimiter la portée du projet

Tâches à effectuer :

- Établir le dictionnaire
- Élaborer le modèle des concepts saillants
- Identifier les acteurs et la description de leurs besoins
- Identifier et Regrouper les cas d'utilisation en packages

#### a - Le Dictionnaire

C'est un outil de dialogue entre le client et le développeur. Il est informel, évolutif et simple à réaliser. Son rôle est d'établir et de figer la terminologie du domaine d'application. Il permet d'enlever les ambiguïtés et constitue le point référentiel initial du système. On y définira les **concepts**<sup>1</sup> (ou notions) et les **actions**<sup>2</sup> de la description du projet.

---

<sup>1</sup> Abstractions du système qui deviendront des objets

<sup>2</sup> Verbes qui deviendront des méthodes

## 4.1 - La Phase de spécification

### a - Le Dictionnaire ( Suite 1 )

**Exemple de dictionnaire des notions : Exemple => Gestion d'une bibliothèque publique**

<b>Notion</b>	<b>Définition</b>
client	un client est une personne, généralement résidante de la ville, capable de fournir une preuve d'identité et de résidence et une photo d'elle-même lors de son inscription. Des résidants de villes avoisinant Mirabelle sont aussi acceptés comme clients moyennant des frais annuels d'inscription d'un montant de 25 \$.
Client inactif	un client est considéré inactif quand sa dernière transaction (prêt et location, réservation, retour de document) remonte à plus d'un an.
Amende	un client qui retourne un document (livre ou cassette) après la date de retour attendue doit payer une amende calculée en fonction du type de document et du nombre de jours de retard. Un client doit normalement payer cette pénalité lors de la remise des documents; dans le cas contraire, il ne pourra effectuer de nouvelles transactions tant qu'il n'aura pas payé le montant de son amende au complet.
Etc.	

## 4.1 - La Phase de spécification

### a - Le Dictionnaire ( Suite 2 )

#### **Exemple de Dictionnaire des Actions**

Action	Définition
Classifier les nouveaux documents	La bibliothécaire s'occupe de classifier les nouveaux documents qu'elle reçoit soit d'un fournisseur ou d'un donateur. Elle doit classifier un document selon son type (livre, périodique, cassette) et lui attribuer par conséquent un statut de prêt (à louer, à emprunter, à consulter).
Consulter les index des documents	Pour effectuer une recherche ou repérer un document toute personne peut consulter l'index des documents à partir d'un des terminaux accessibles. Un menu lui offrira différentes clés de recherche. La personne pourra également imprimer les résultats de sa recherche et localiser facilement un document sur les rayons de la bibliothèque
Inscrire un client	Tout nouvel inscrit se verra remettre une carte de client sur laquelle on retrouvera sa photo ainsi qu'un code zébré lisible au moyen d'un lecteur optique. C'est le préposé au comptoir de retour qui s'occupe d'inscrire les nouveaux clients et d'imprimer leur carte de client
Prêter ou louer des documents aux clients	Le client qui le désire peut emprunter un document ou louer un livre qu'il a pris sur un rayon de la bibliothèque. Il doit se présenter au comptoir de prêt avec ses documents et sa carte de client. S'il a oublié sa carte de client, le préposé, selon l'achalandage de l'heure, pourra, au besoin, rechercher son dossier avec son nom ou son numéro de téléphone. Le client désirant effectuer un emprunt ne doit pas avoir d'amende impayée, sinon il sera invité à la payer au complet et sur le champ. Le préposé doit vérifier le statut de prêt de chaque document. Un livre en location peut être loué si le client paie le montant de la location pour lequel le préposé lui émet un reçu. Un document à consulter ne peut être ni emprunté ni loué.
Etc.	

## 4.1 - La Phase de spécification

### b - Le Modèle des Concepts Saillants : Modèle de classe d'analyse

Le **modèle des concepts saillants** est une représentation graphique à l'aide de rectangles des classes des concepts importants du domaine à informatiser et des liens entre ces concepts .

Ce **diagramme** est composé de rectangles représentants les concepts fondamentaux et importants ( objets ) du système reliés par des segments de droite ( liens entre ces concepts ) .

Le diagramme résultant du modèle des concepts saillants est un premier brouillon qui permettra d'évoluer après plusieurs itérations vers le diagramme de classes .

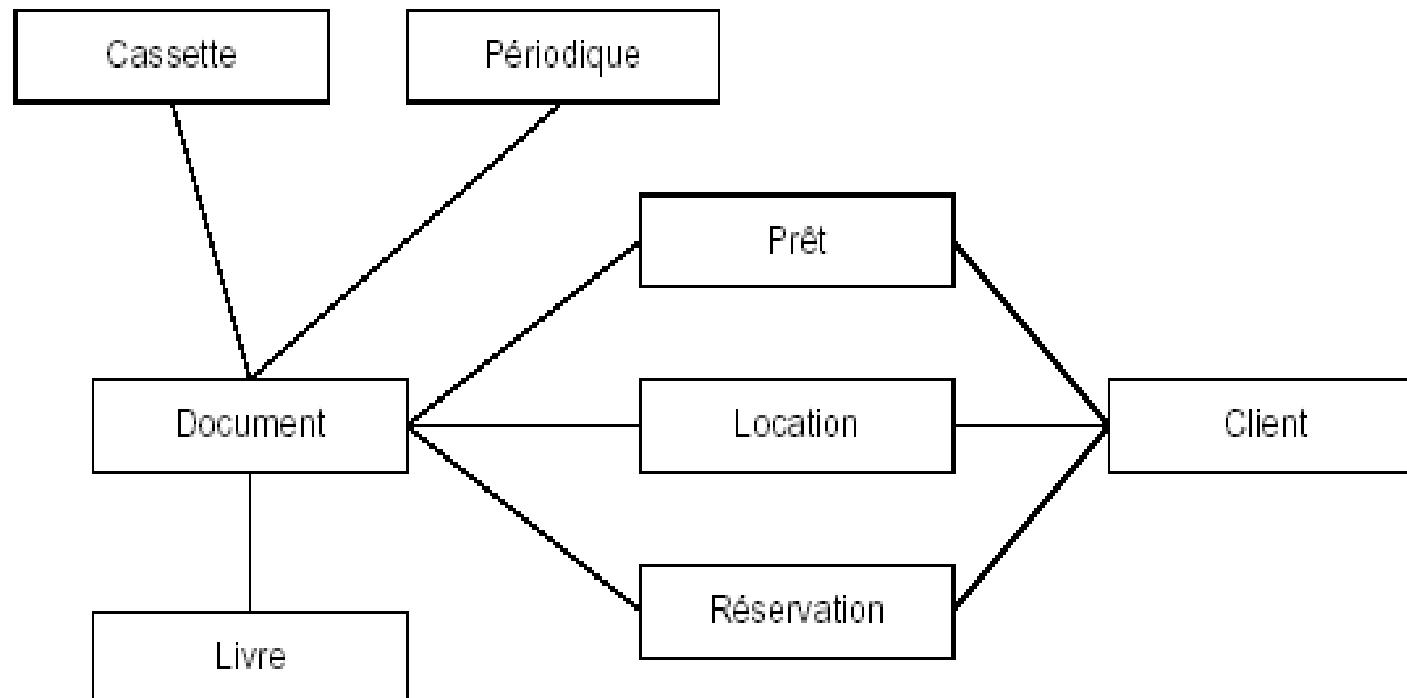


Figure 1: Modèle des concepts pour la bibliothèque publique

## 4.1 - La Phase de spécification

### c – Identifier les Acteurs et la description de leurs besoins

La prochaine étape est d'identifier le plus clairement possible les limites ( ou frontières ) du système .  
On découvre ces limites en définissant les **Acteurs** et les **Cas d'utilisation du système** .

#### Notion d'Acteur :

Un Acteur est une entité externe au système qui est amenée à interagir avec lui .

Un Acteur peut être une personne , un autre système , un périphérique d'ordinateur , une base de données , un réseau , etc... Chaque acteur définit un rôle précis .

#### Comment déterminer les acteurs d'un système ?

En trouvant une réponse aux questions suivantes : Qui utilise , installe démarre , maintient ou arrête le système ? Qui obtient ou entre l'information dans le système ? Le système interagit-il avec d'autres systèmes ?

Exemple de la bibliothèque :

Acteurs	Rôle
Client Préposé au comptoir	S'inscrit Emprunte des documents Loue des livres Réserve des livres Retourne des documents Consulte les index sur les documents Paie une amende
Bibliothécaire	Classifie les documents
Direction	Demande un rapport
Comptabilité	Encaisse les frais

## 4.1 - La Phase de spécification

### d – Identifier les Cas d’Utilisation fondamentaux du système ( Use Cases )

Un cas d'utilisation est une manière particulière d'utiliser le système.

Un cas d'utilisation est une séquence d'interactions entre le système et un ou plusieurs acteurs.

L'ensemble des cas d'utilisation représente de manière informelle le service rendu par le système via une expression fonctionnelle.

Il est très important que les cas d'utilisation capturés utilisent la terminologie décrite dans le dictionnaire.

Cas d'utilisation	Acteur	Message déclencheur
Classifer les documents	Bibliothécaire	Réception de nouveau document
Consulter les index	Client Préposé a comptoir	Demande de consultation
Inscrire un client	Client Préposé a comptoir	Un client se présente a comptoir pour s'inscrire
Prêter des documents	Client Préposé a comptoir	Un client se présente a comptoir avec des documents à prêter
Retourner des documents	Client Préposé a comptoir	Un client se présente a comptoir avec des documents à retourner
Payer une amende	Client Préposé a comptoir	Un client se présente au comptoir et a des documents en retard
Réserver des livres	Client Préposé a comptoir	Un client veut réserver un document déjà loué
Louer sur réservation	Client Préposé a comptoir	Un client qui a une réservation veut louer un document
Produire un rapport	La direction	

## Cas d'utilisation

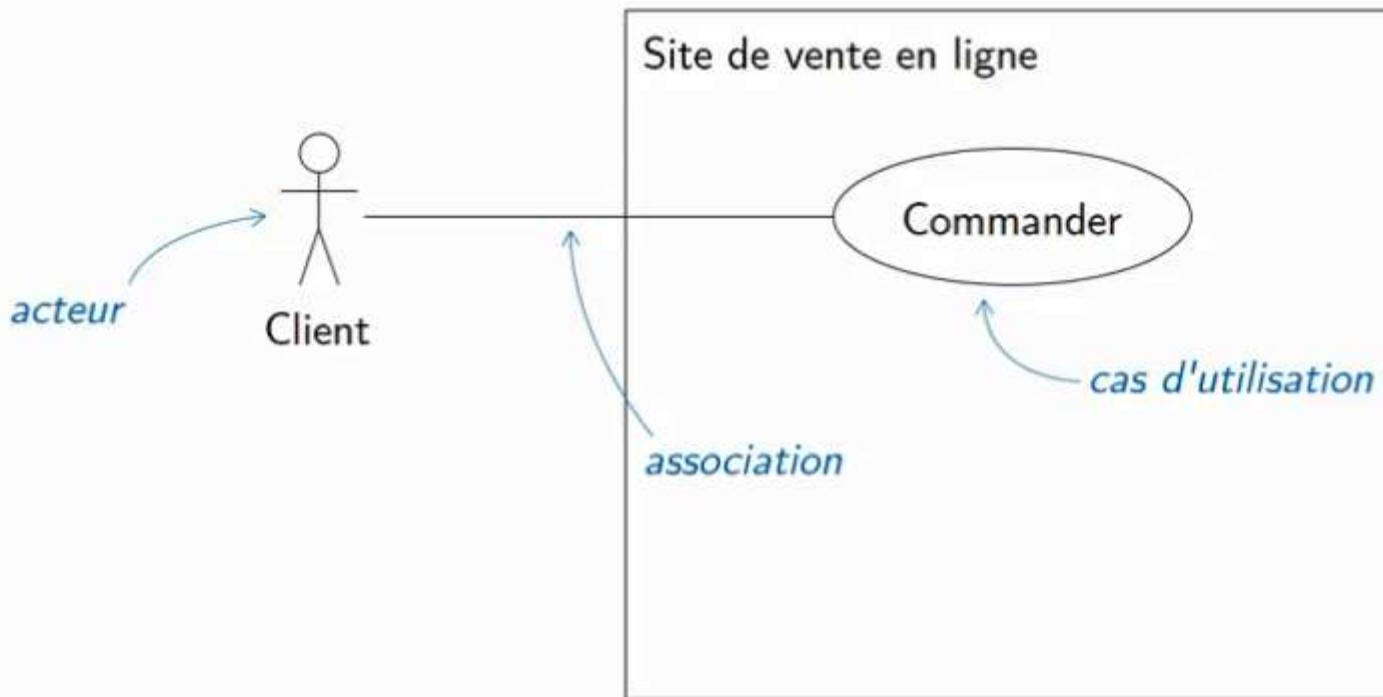
- Ensemble de scénarios réalisant un objectif de l'utilisateur
- Fonctionnalités principales du système du point de vue extérieur

**Acteur** : Entité qui interagit avec le système

- Personne, chose, logiciel, extérieur au système décrit
- Représente un rôle (plusieurs rôles possibles pour une même entité)
- Identifié par le nom du rôle

**Cas d'utilisation** : Fonctionnalité visible de l'extérieur

- Action déclenchée par un acteur
- Identifié par une action (verbe à l'infinitif)



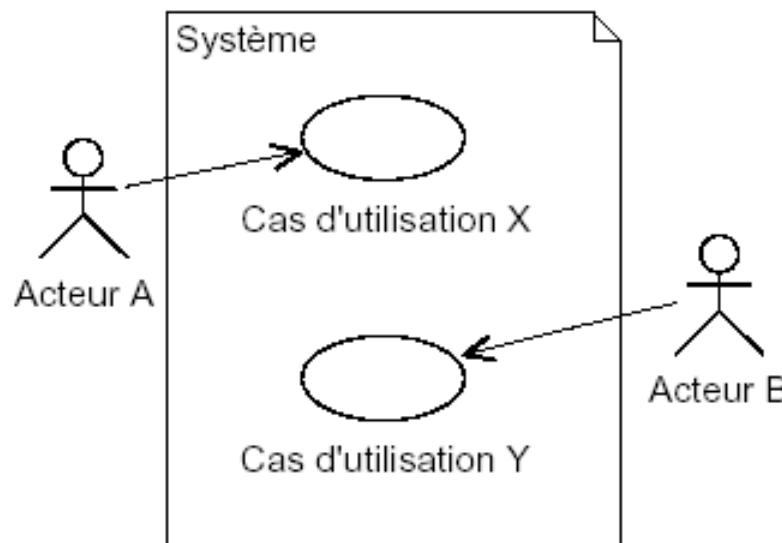
### Association :

- Relation entre acteurs et cas d'utilisation
- Représente la possibilité pour l'acteur de déclencher le cas

## 4.1 - La Phase de spécification

### d – Identifier les Cas d’Utilisation ( Suite 1 )

#### Notation des Cas d'utilisation ( diagramme )



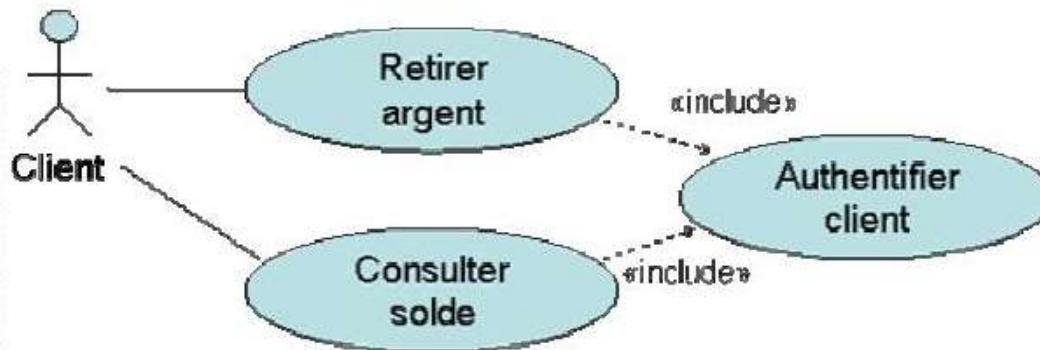
#### Relation entre les Cas d’Utilisation d’un système

Il existe trois types de relations entre les cas d'utilisation :

- ✓ Une relation d'inclusion dénotée par le stéréotype <<include>>
  - À utiliser lorsqu'un cas de base est inclus à l'intérieur de la séquence d'un autre cas plus élaboré. Il ne peut jamais être exécuté seul.
- ✓ Une relation d'extension dénotée par le stéréotype <<extend>>
  - À utiliser pour séparer un comportement qui varie sous certaines conditions à certains points (d'extension) de son exécution. Il peut s'exécuter seul ou être complété par un autre (devenant un point d'extension).
- ✓ Une relation d'héritage dénotée par le sigle de l'héritage pointant sur le *super-acteur*.
  - À utiliser lorsqu'un cas comprend des interactions ou des fonctionnalités communes



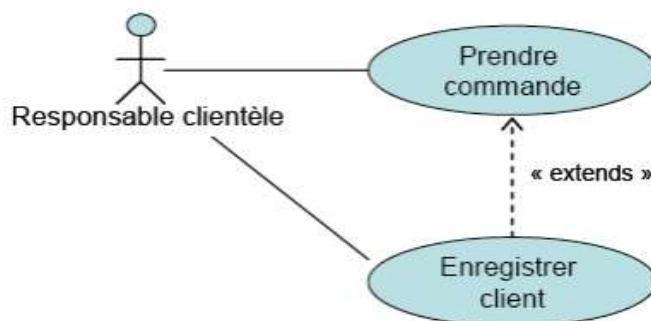
Include



- Le cas d'utilisation contient un autre cas d'utilisation

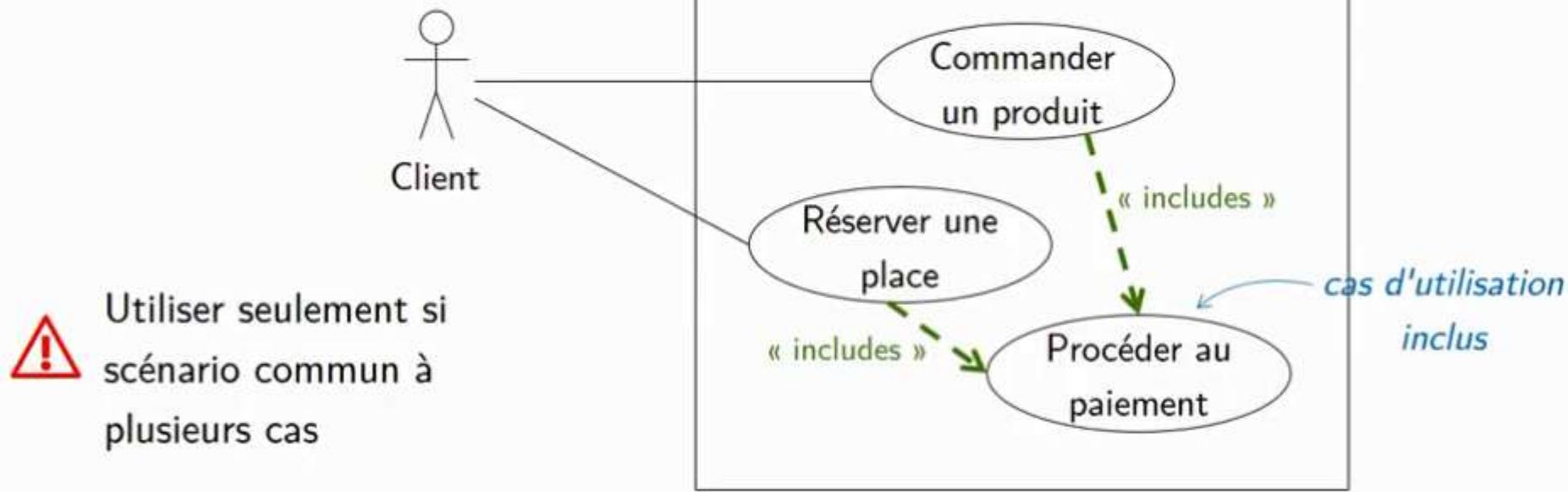


Extends :



- Le cas d'utilisation étend (précise) les objectifs (le comportement) d'un autre cas d'utilisation.
- On dit qu'un cas d'utilisation X étend un cas d'utilisation Y lorsque le cas d'utilisation X peut être appelé au cours de l'exécution du cas d'utilisation Y.

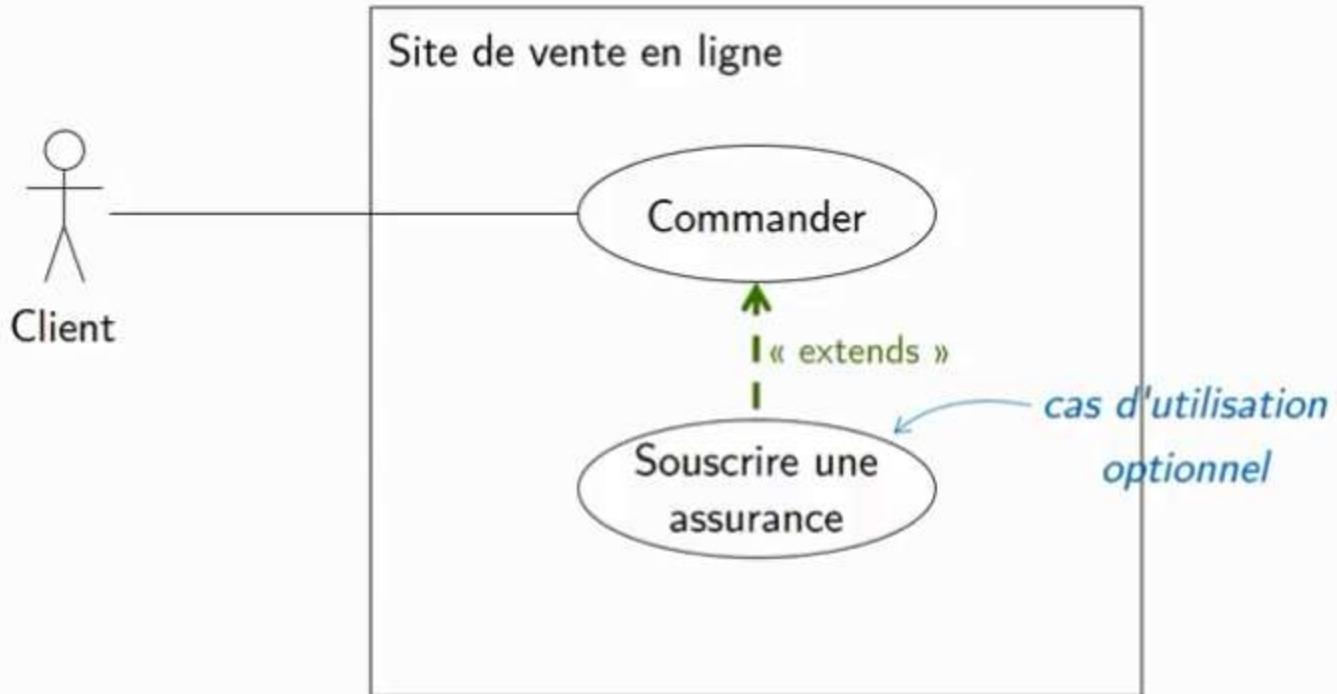
# Relations entre cas d'utilisation



**Inclusion** : X « includes » Y

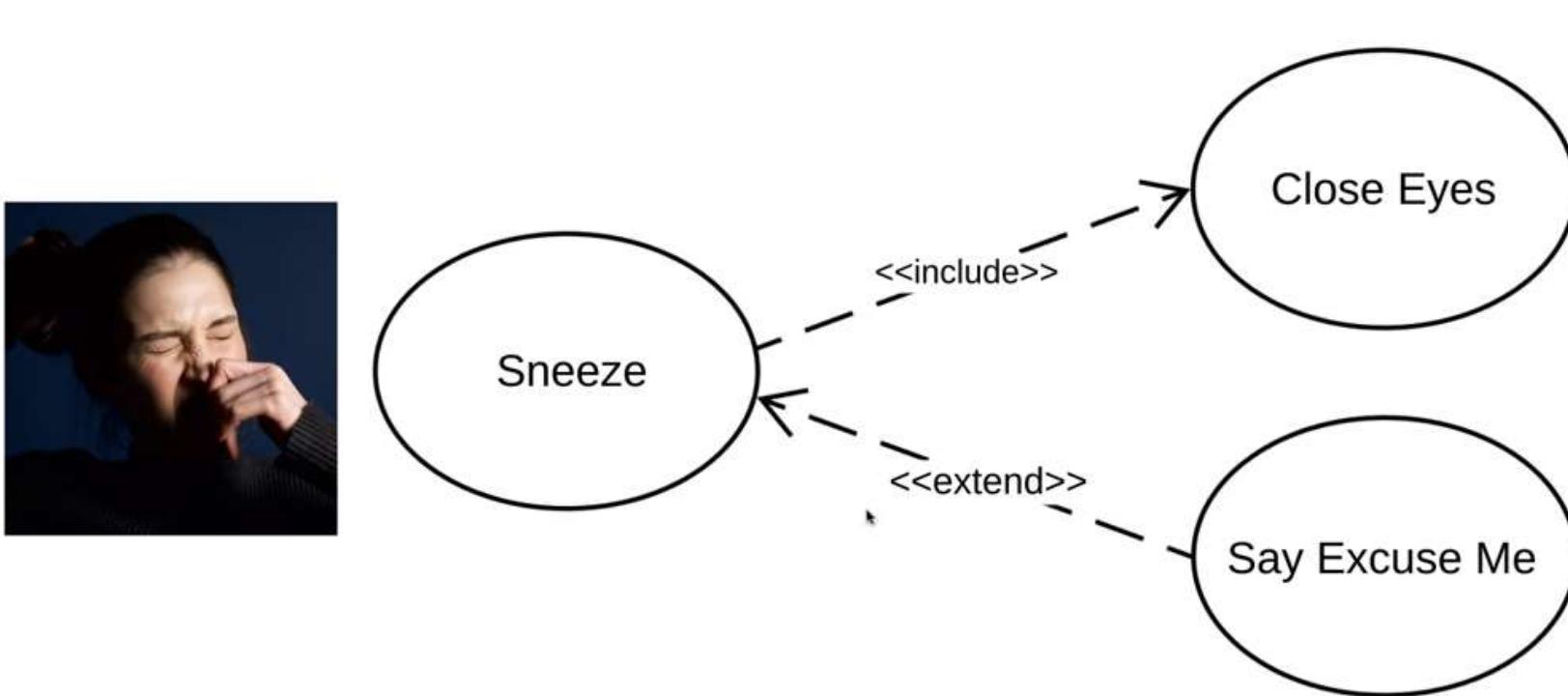
- Scénario de Y inclus dans le scénario de X
- Cas d'utilisation Y déclenché au cours du scénario de X

# Relations entre cas d'utilisation



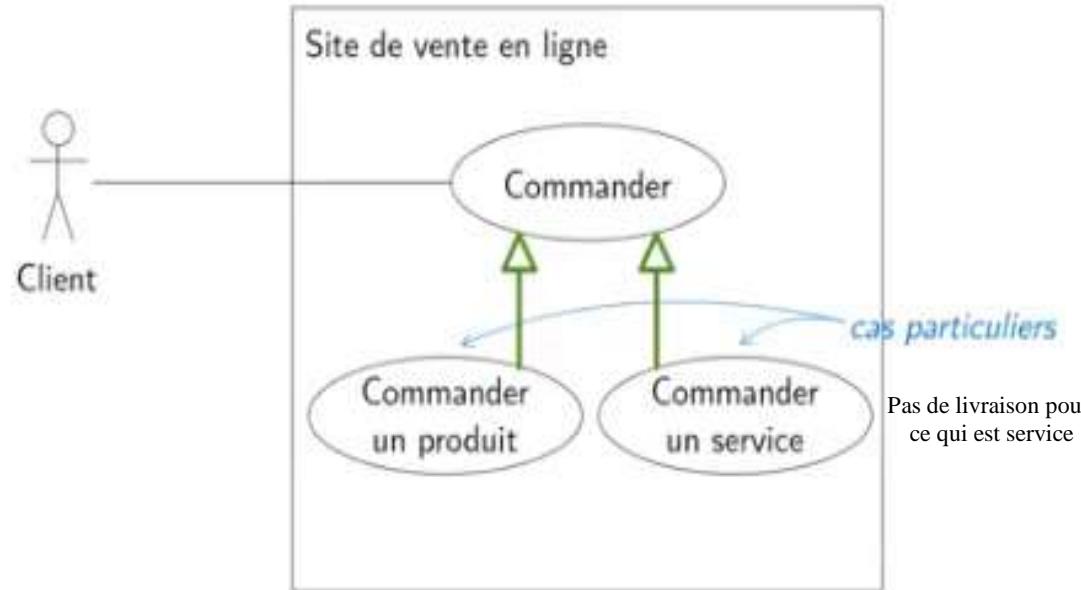
Extension : X « extends » Y

- Cas d'utilisation X peut être déclenché au cours du scénario de Y
- X est optionnel pour Y



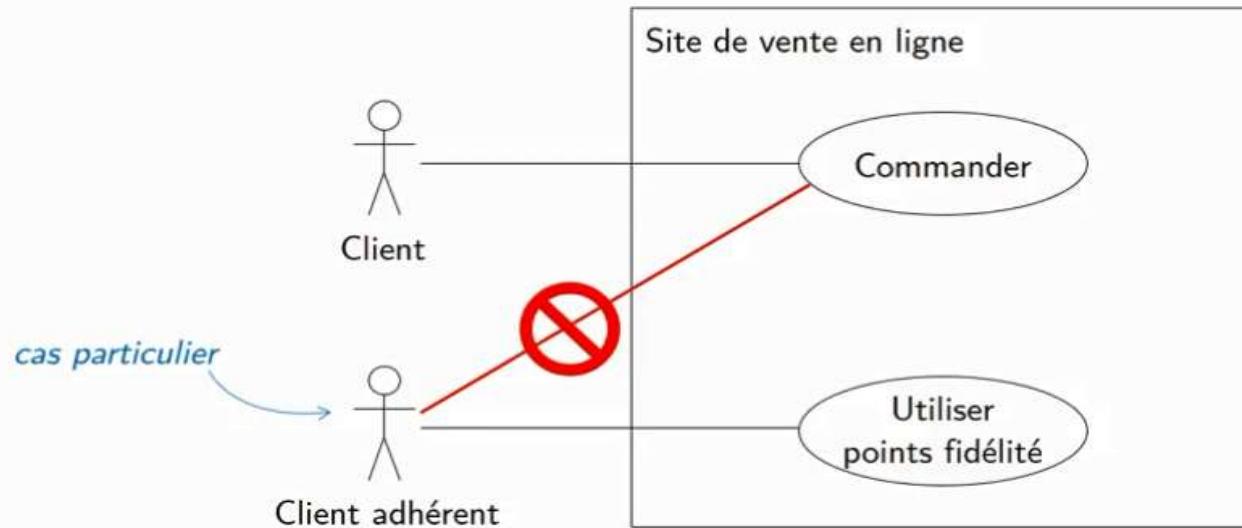


## Généralisation

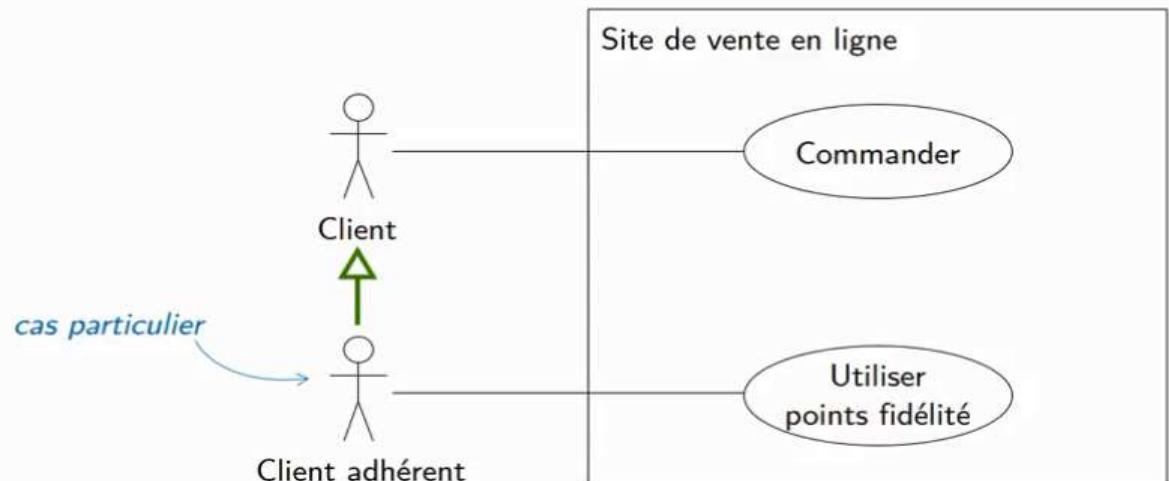
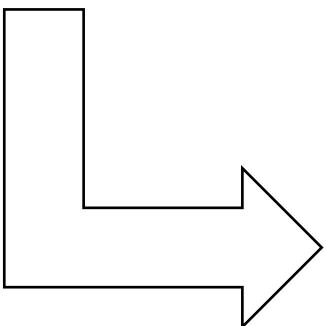


Généralisation : X est un cas particulier de Y  
Tout ou partie du scénario de Y est spécifique à X

- Les cas d'utilisation descendants héritent de la description de leur parent commun. Chacun d'entre eux peut néanmoins comprendre des interactions spécifiques supplémentaires.



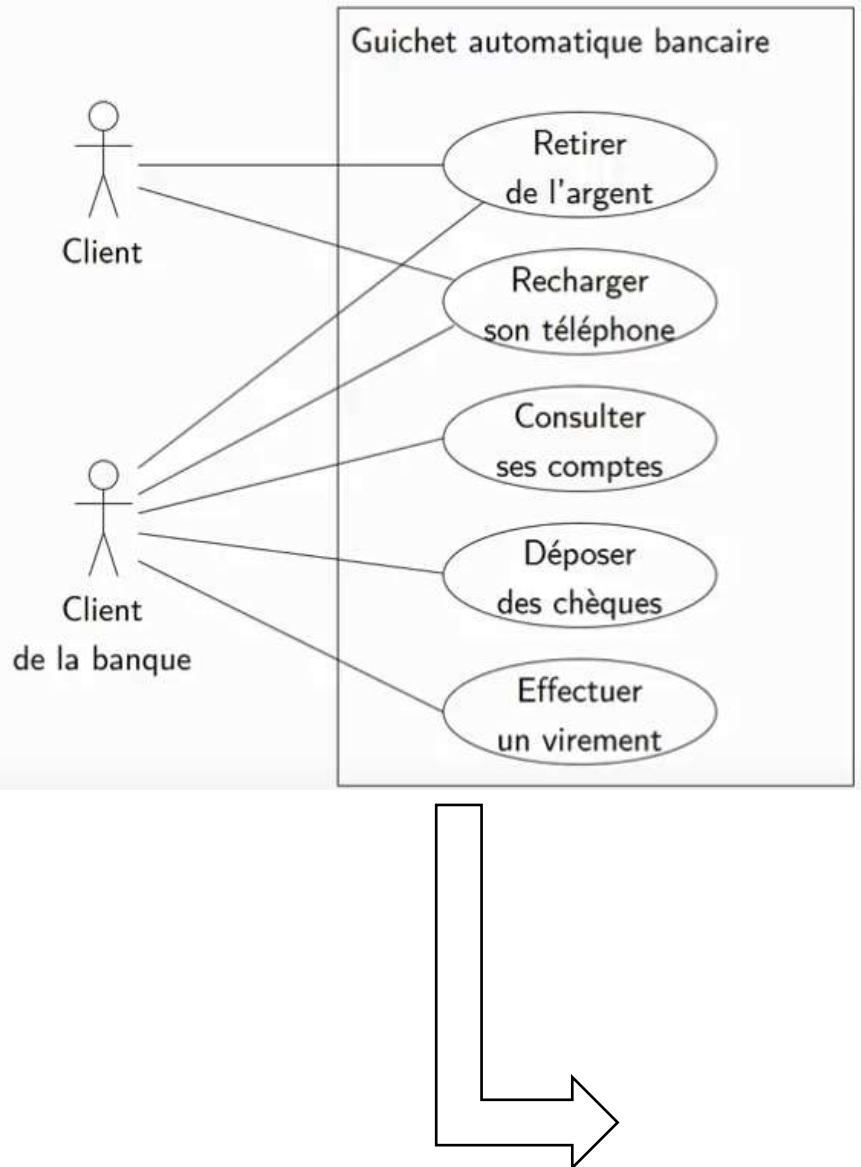
## Généralisation de rôle



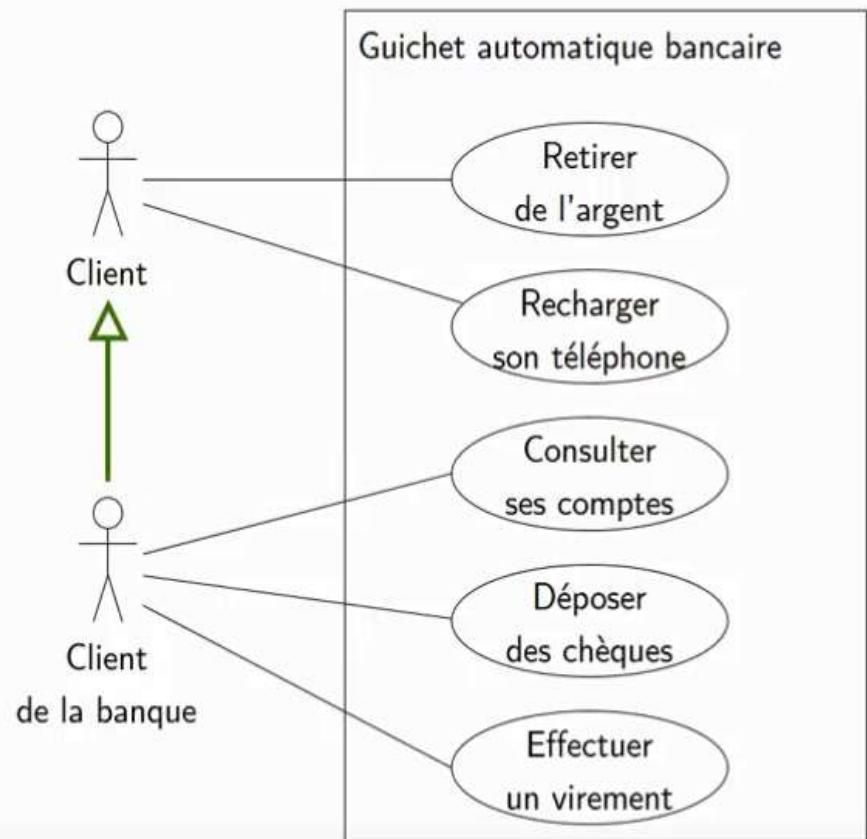
Situation : Y peut faire tout ce que fait X

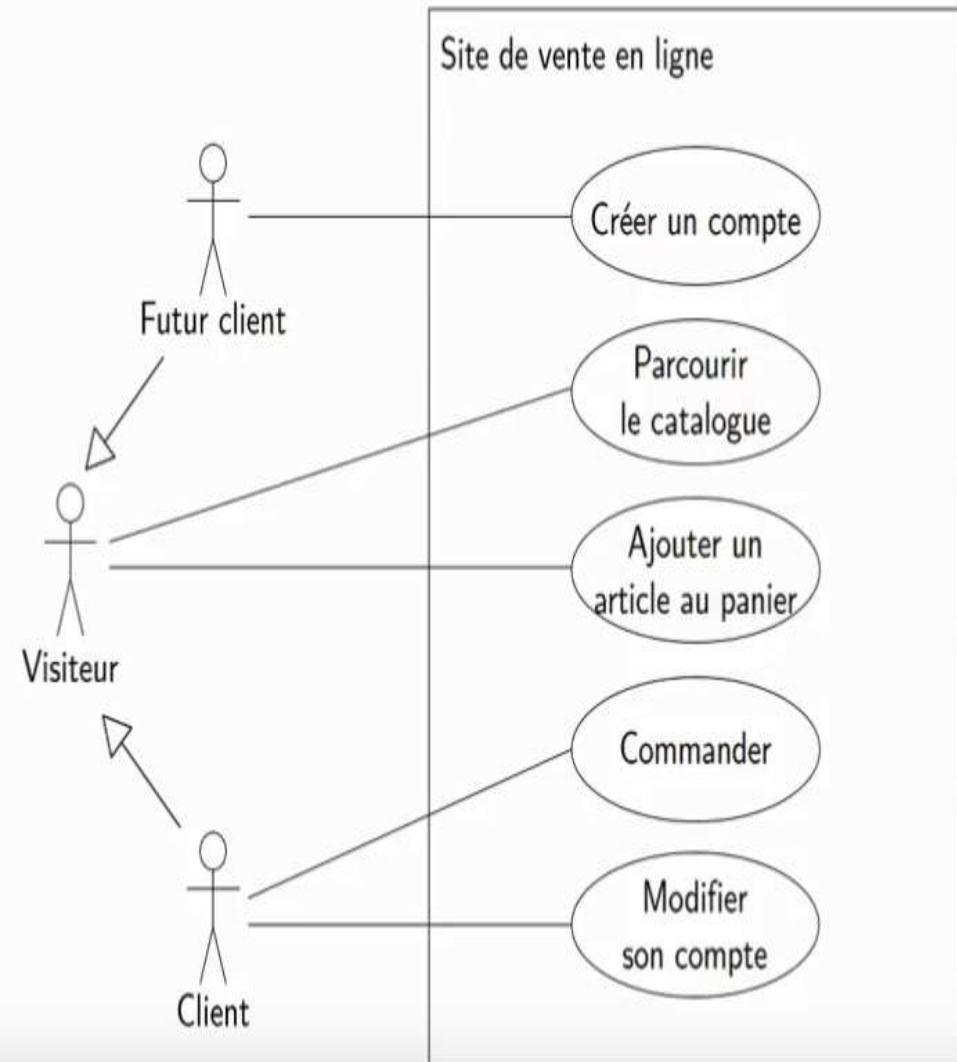
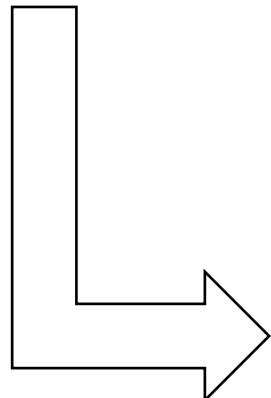
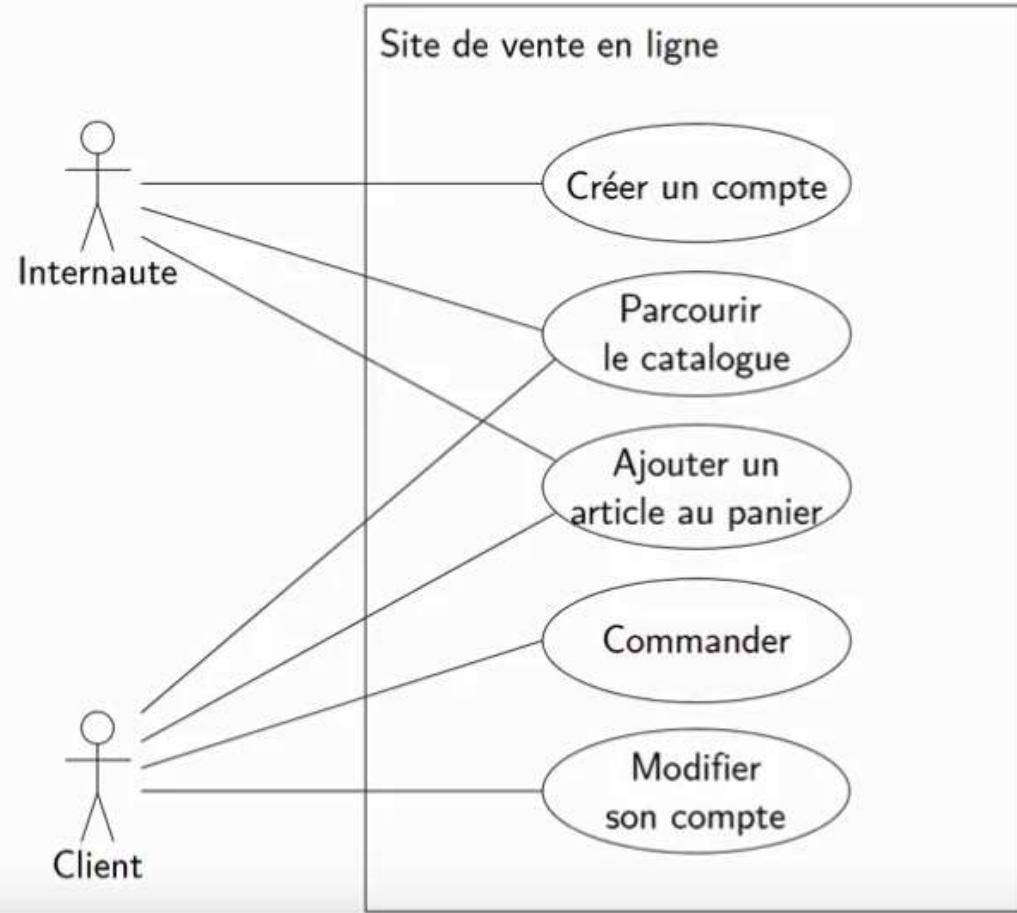
Modélisation : Faire apparaître Y comme un cas particulier de X  
(ou X généralisation de Y)

## Exemple : généralisation de rôle

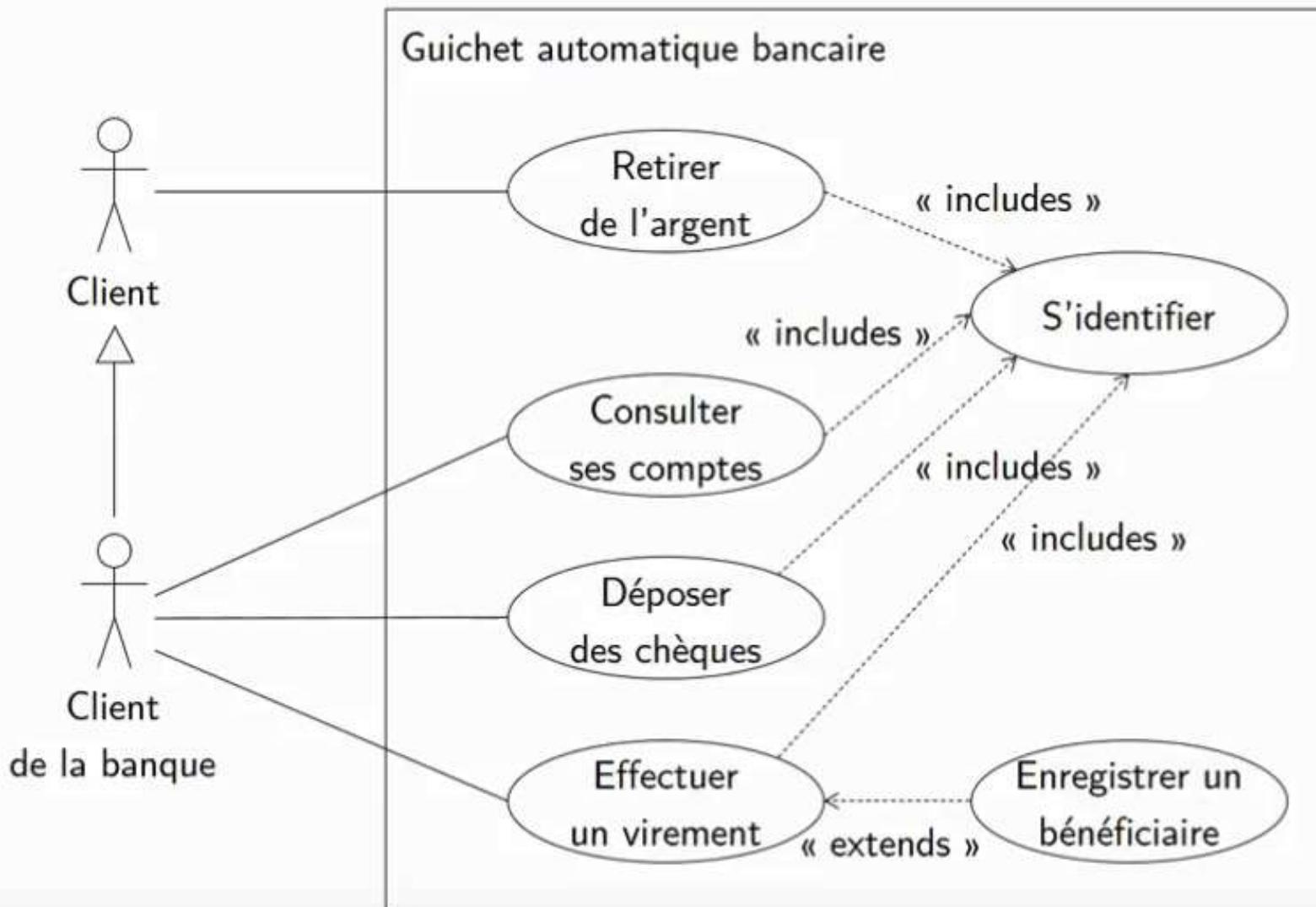


## Exemple : généralisation de rôle





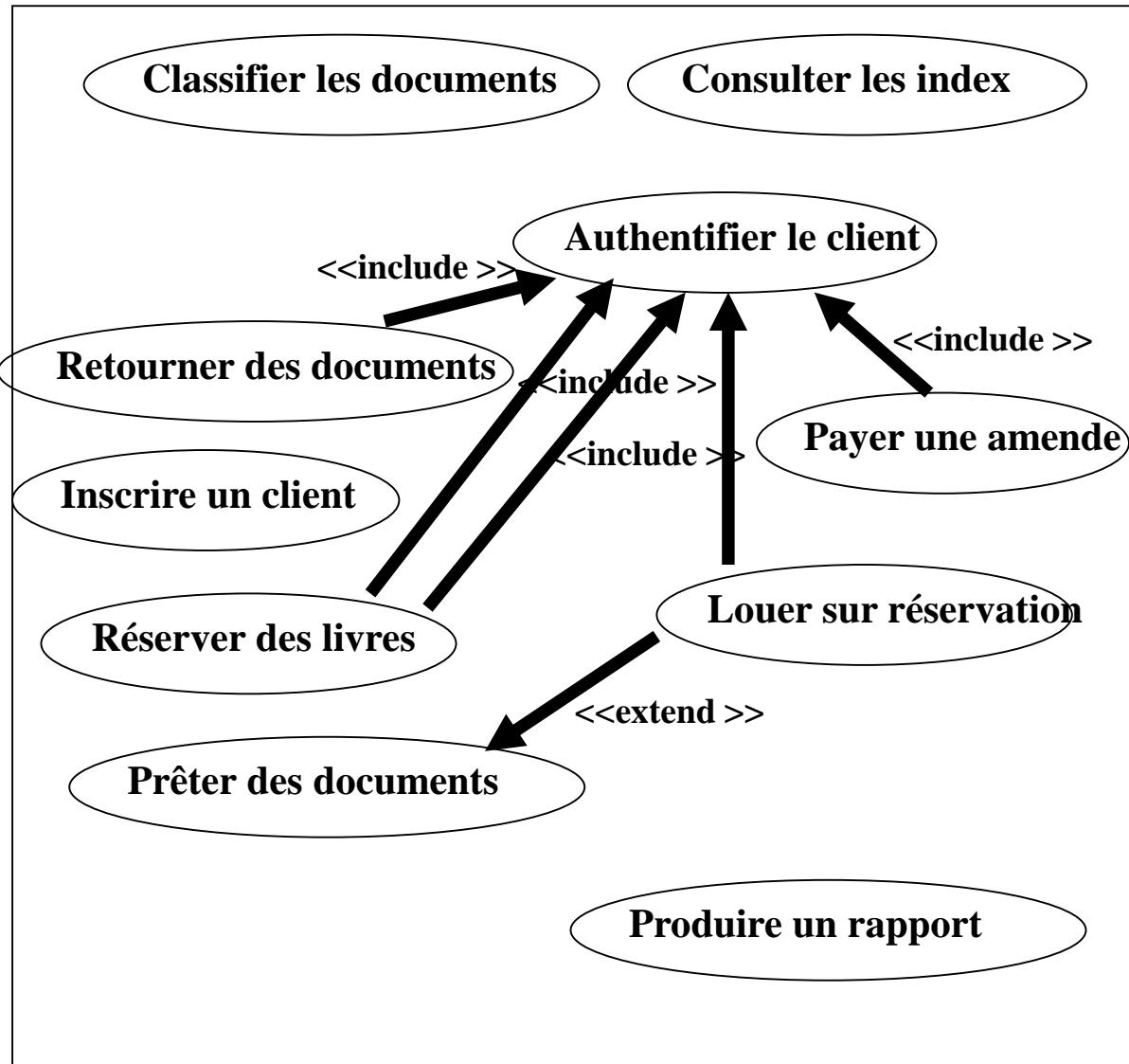
# Exemple : relations entre cas d'utilisation



## 4.1 - La Phase de spécification

### e – Représenter le diagramme des cas d'utilisation du système ( suite )

**Relation entre les Cas d'utilisation**



## Conseils

Rester lisible :

- Pas plus de 6 ou 8 cas dans un diagramme
- Au besoin, faire plusieurs diagrammes (si cas disjoints entre acteurs, pour détailler un cas...)
- Relations entre cas seulement si nécessaires et pas trop lourdes

Pour les détails, privilégier la description textuelle

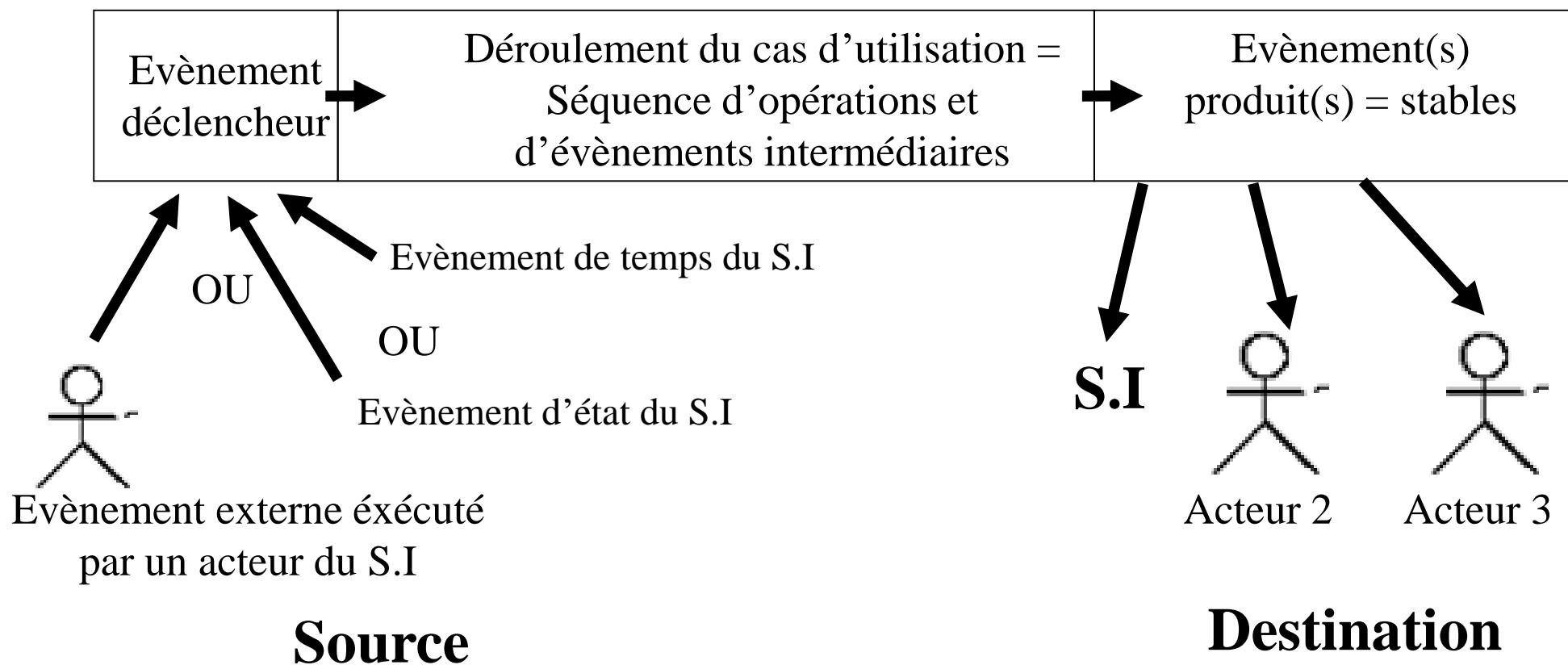
## 4.1 - La Phase de spécification

## f - Structure d'un cas d'utilisation

Un cas d'utilisation est une activité élémentaire du S.I générée par un événement particulier

## 3 types d'évènements déclencheurs :

- les événements externes au système qui sont souvent initiés par un acteur ;
  - les événements temporels qui résultent de l'atteinte d'un moment dans le temps ;
  - les événements d'état qui se produisent quand quelque chose dans le système nécessite un besoin de traitement.



# Scénarios d'utilisation

## Séquences d'étapes

- décrivant une **interaction** entre l'utilisateur et le système
- permettant à l'utilisateur de réaliser un **objectif**

**Système** : Site de vente en ligne

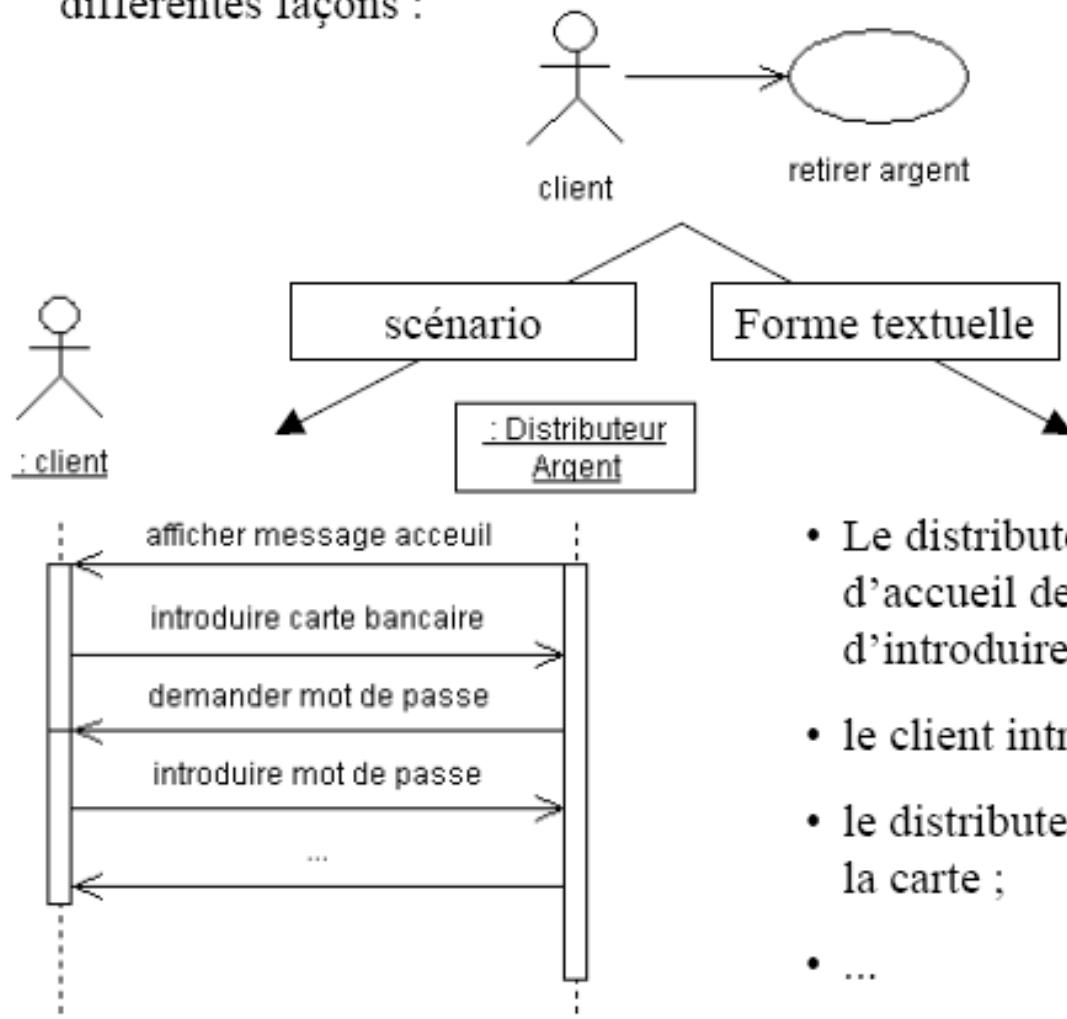
**Scénario** : Commander

Le client s'authentifie dans le système puis choisit une adresse et un mode de livraison. Le système indique le montant total de sa commande au client. Le client donne ses informations de paiement. La transaction n'est pas autorisée, le système invite le client à changer de mode de paiement. Le client modifie ses informations. La transaction est effectuée et le système en informe le client par e-mail.

## 4.1 - La Phase de spécification

### h - Documentation du scénario d'un cas d'utilisation

- Les use cases peuvent être décrits sous la forme de flots d'événements de différentes façons :

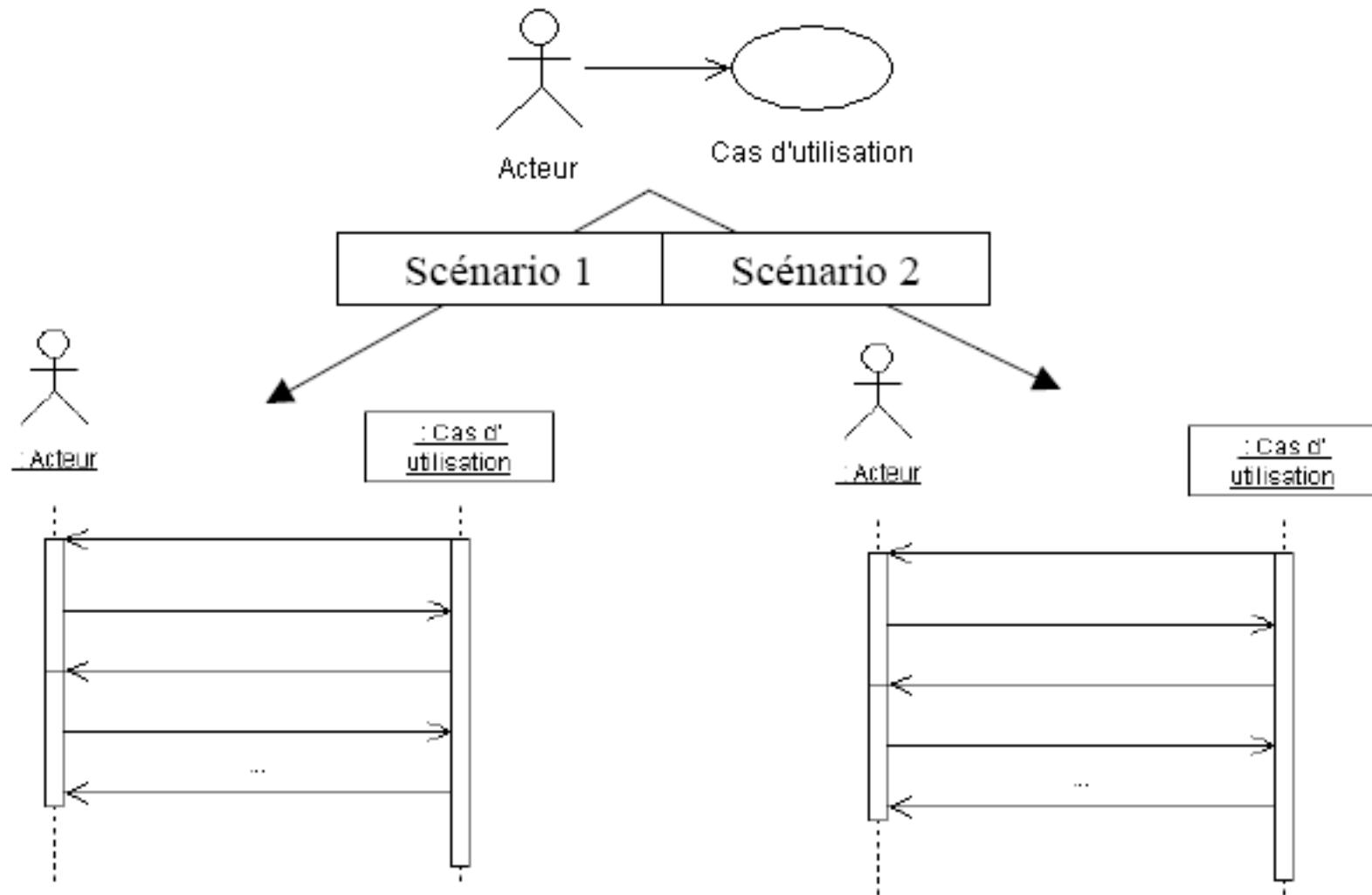


- Le distributeur affiche un message d'accueil demandant à un client d'introduire sa carte bancaire ;
- le client introduit sa carte bancaire ;
- le distributeur demande le mot de passe de la carte ;
- ...

## 4.1 - La Phase de spécification

### i - Cas de plusieurs scénarios

- Un cas d'utilisation peut être décrit par plusieurs scénario :



## 4.1 - La Phase de spécification

### e - Déttailler chaque Cas d'Utilisation

UML n'a pas standardisé la description textuelle (fiche) d'un cas d'utilisation . Par conséquent, l'entreprise doit faire un choix sur le format d'un cas d'utilisation en fonction de ses besoins et de son domaine.

Voici le format que nous utiliserons dans le cours :

#### **Identification**

- ✓ Titre, résumé, Acteurs

#### **Description des scénarios**

- ✓ Préconditions, scénarios, exceptions, postconditions

#### **Besoins d'interfaces graphiques**

#### **Contraintes non fonctionnelles**

**Dans l'exemple  
de la bibliothèque :**

#### **Identification**

- ✓ *Titre* : Prêter des document
- ✓ *Résumé* : Le client se présente au comptoir de prêt dans le but d'emprunter des documents
- ✓ *Acteurs* : Le client, le préposé au comptoir

#### **Description des scénarios**

## 4.1 - La Phase de spécification

### e - Détailier chaque Cas d'Utilisation

UML n'a pas standardisé la description textuelle (fiche) d'un cas d'utilisation . Par conséquent, l'entreprise doit faire un choix sur le format d'un cas d'utilisation en fonction de ses besoins et de son domaine.

Voici le format que nous utiliserons dans le cours :

#### **Identification**

- ✓ Titre, résumé, Acteurs

#### **Description des scénarios**

- ✓ Préconditions, scénarios, exceptions, postconditions

#### **Besoins d'interfaces graphiques**

#### **Contraintes non fonctionnelles**

**Dans l'exemple  
de la bibliothèque :**

#### **Identification**

- ✓ *Titre* : Prêter des document
- ✓ *Résumé* : Le client se présente au comptoir de prêt dans le but d'emprunter des documents
- ✓ *Acteurs* : Le client, le préposé au comptoir

#### **Description des scénarios**

- ✓ *Préconditions* :
  - Le client a une carte de client valide
  - Le client n'a pas atteint sa limite de prêt
  - Le client n'a pas d'amende à payer
  - Le client n'a pas de prêtés en retard

## 4.1 - La Phase de spécification

### e - Détailier chaque Cas d'Utilisation ( Suite 1 )

Suite de l'exemple sur la bibliothèque ( description des scénarios ) :

✓ *Scénarios :*

## 4.1 - La Phase de spécification

### e - Détailler chaque Cas d'Utilisation ( Suite 1 )

**Suite de l'exemple sur la bibliothèque ( description des scénarios ) :**

✓ *Scénarios :*

Le client se présente au comptoir de prêt avec des documents à emprunter ou à louer. Il remet sa carte de client au préposé au prêt. Le préposé fait lire le code zébré de la carte et vérifie le dossier du client (règles d'initiation). Si toutes les conditions sont satisfaites, le préposé fait lire le code zébré de chaque document. Il s'assure qu'un document "à consulter" n'est pas sorti, que les dates de retour assignées par le système sont acceptées par le client et que la limite de prêt n'est pas dépassée.

✓ *Exceptions :*

## 4.1 - La Phase de spécification

### e - Détailler chaque Cas d'Utilisation ( Suite 1 )

**Suite de l'exemple sur la bibliothèque ( description des scénarios ) :**

✓ *Scénarios :*

Le client se présente au comptoir de prêt avec des documents à emprunter ou à louer. Il remet sa carte de client au préposé au prêt. Le préposé fait lire le code zébré de la carte et vérifie le dossier du client (règles d'initiation). Si toutes les conditions sont satisfaites, le préposé fait lire le code zébré de chaque document. Il s'assure qu'un document "à consulter" n'est pas sorti, que les dates de retour assignées par le système sont acceptées par le client et que la limite de prêt n'est pas dépassée.

✓ *Exceptions :*

1. Le client a oublié d'apporter sa carte de client. Avec son nom ou son numéro de téléphone, le préposé procède à une recherche du dossier du client à l'ordinateur. S'il le trouve, il poursuit le processus selon la description ci-dessus, sinon il lui recommandera de revenir avec sa carte.

2. Le client qui a une amende à payer peut poursuivre le processus s'il accepte de payer cette amende au complet et sur-le-champ; le préposé inscrit alors le paiement et lui émet un reçu pour le montant payé.

## 4.1 - La Phase de spécification

### e - Détailler chaque Cas d'Utilisation ( Suite 2 )

**Suite de l'exemple sur la bibliothèque :**

**Besoin d'interface graphique pour le Cas d'utilisation :**      **Oui**

**Contraintes non fonctionnelles :**

- Population du quartier = 35 000 habitants avec environ 12 000 inscrits à la bibliothèque
- Le Système devra être accessible tous les jours de 13 H à 22 H .
- Besoins en équipement : 1 Serveur + 8 terminaux

## 4.1 - La Phase de spécification

### e - Détailler chaque Cas d'Utilisation ( Suite 2 )

**Suite de l'exemple sur la bibliothèque :**

**Besoin d'interface graphique pour le Cas d'utilisation :**      **Oui**

**Contraintes non fonctionnelles :**

- Population du quartier = 35 000 habitants avec environ 12 000 inscrits à la bibliothèque
- Le Système devra être accessible tous les jours de 13 H à 22 H .
- Besoins en équipement : 1 Serveur + 8 terminaux

Terminal n° 1 : Prêt des documents

Terminal n° 2 : Retour des documents

Terminal n° 3 : Inscription des nouveaux clients

Terminal n° 4,5 et 6 : Consultation des usagers

Terminal n° 7 et 8 : Bibliothécaire

Certains terminaux devront être munis d'un lecteur optique permettant de lire le code à barre des documents et des cartes des clients .

## 4.1 - La Phase de spécification

### e - Détailler chaque Cas d'Utilisation ( Suite 2 )

Suite de l'exemple sur la bibliothèque :

**Besoin d'interface graphique pour le Cas d'utilisation :**      **Oui**

**Contraintes non fonctionnelles :**

- Population du quartier = 35 000 habitants avec environ 12 000 inscrits à la bibliothèque
- Le Système devra être accessible tous les jours de 13 H à 22 H .
- Besoins en équipement : 1 Serveur + 8 terminaux

Terminal n° 1 :	Prêt des documents
Terminal n° 2 :	Retour des documents
Terminal n° 3 :	Inscription des nouveaux clients
Terminal n° 4,5 et 6 :	Consultation des usagers
Terminal n° 7 et 8 :	Bibliothécaire

Certains terminaux devront être munis d'un lecteur optique permettant de lire le code à barre des documents et des cartes des clients .

**Niveau de priorité retenu :**

- 1 – Inscription des nouveaux clients**
- 2 – Comptoir de prêt**
- 3 – Comptoir de retour**
- 4 – Terminaux de consultation des index**
- 5 – Terminaux pour l'administration ( bibliothécaire )**

## 4.1 - La Phase de spécification

### e - Détailler chaque Cas d'Utilisation ( Suite 3 )

**Suite de l'exemple sur la bibliothèque :**

#### **Les bases de données :**

L'estimation des besoins de stockage minimaux peut être faite de la façon suivante :

<b>Données (volumes : an1/ an2/ an3)</b>	<b>An 1 (début)</b>	<b>An 2</b>	<b>An 3</b>
Unités	Méga-octets	Méga-octets	Méga-octets
Client (12 000/ 15 000/ 16 500)	1,60	1,95	2,15
Livre (500 000/ 550 000/ 600 000)	11,25	12,40	13,50
Périodique (400/ 500/ 600)	0,09	1,12	1,35
Cassettes (2 000/ 2 500/ 3 000)	0,40	0,51	0,60
Prêt		9,50	10,80
Location		0,84	1,20
Réservation		1,30	1,82
Total	13,74	27,62	31,41

**Temps de réponse du système :** Temps maximal acceptable : 10 secondes

Temps normal : 5 secondes

Temps idéal : <= 3 secondes

#### **Normes générales du système :**

Le système devra être développé dans un environnement PC et pouvoir évoluer facilement sur une période d'une dizaine d'années. Le choix d'une base de données et d'un système d'exploitation qui rende cette condition possible et l'évolution des logiciels développés sont des exigences primordiales dans ce dossier.

## **4.1 - La Phase de spécification**

### **j - Les limites des cas d'utilisation**

- Les cas d'utilisation ne sont pas la seule manière de documenter les besoins d'un système :
  - une description textuelle peut suffire ;
  - un maquettage simple représentant l'interface graphique d'un système est très utile ;
- les cas d'utilisation se limitent à décrire le « quoi » d'un système mais pas le « comment » ;
- les cas d'utilisation sont une description fonctionnelle d'un système après quoi il faut passer à une description objet (les scénarios) utilisant :
  - les diagrammes de séquences ou les diagrammes d'activités comme alternative ;
  - les diagrammes de collaboration ;
- il y a en général peu de cas d'utilisation (de 10 à 20) mais beaucoup de scénarios.

## 4.1 - La Phase de spécification

### f - Regrouper les Cas d'utilisation en Packages

On regroupe les Cas d'Utilisation en Packages en s'inspirant des acteurs et des rôles .

Suite de l'exemple sur la bibliothèque :

#### Package Gestion des documents

Classifier document	(Bibliothécaire)
Consulter index	(Client, Préposé au comptoir)
Produire rapport	(Direction)

#### Package Prêt de document

Emprunter un document	(Client, Préposé au comptoir)
Retourner un document	(Client, Préposé au comptoir)

#### Package Location de document

Emprunter un document	(Client, Préposé au comptoir)
Louer sur réservation	(Client, Préposé au comptoir)
Retourner un document	(Client, Préposé au comptoir)

#### Package Réserver un document

Réserver des livres	(Client, Préposé au comptoir)
---------------------	-------------------------------

# La Phase d'élaboration

Cette phase se concentre sur l'**Analyse et la Conception** .

Elle permet pour chaque cas d'utilisation du système de :

- Capturer les requis ( description de ce que le système doit faire ( le Quoi ? )  
=> **Analyse**
- Identifier et détailler les classes et leur interaction ( associations entre classes )  
=> **Conception du comportement statique**
- Identifier les messages échangés entre les objets ( instances des classes ) et leur synchronisation
  - + Traduire ces messages en méthodes de classe  
=> **Conception du comportement dynamique**
- Regrouper les classes en **Packages** ( composants assurant chacun une activité principale dans le cas d'utilisation)

Les diagrammes à produire dans la phase d'élaboration sont :

- ✓ Diagramme de classes (modèle statique)
- ✓ Diagramme de séquence ou de collaboration (modèle dynamique)
- ✓ Diagrammes de composants
- ✓ Diagrammes de déploiement

# La Phase d'élaboration

On va rappeler ici les principaux concepts objets, car en effet, ces notions sont très importantes pour la compréhension des différents modèles d'UML et leurs implémentations, plus exactement les diagrammes qui vont décrire les concepts de UML (qui est avant tout un langage graphique).

## 1 – Notion d'Objet et de Classe

Un objet est une entité hermétique qui contient à la fois des données et des traitements (les données sont appelées attributs et les traitements opérations ou méthodes).

Une classe est une abstraction d'objets (ne pas confondre avec l'abstraction des concepts objets). C'est à dire une structure qui rassemble des propriétés communes à une collection d'objets.

On dit qu'un objet est une instance de classe, c'est à dire une valeur particulière de la classe (notion d'instanciation). Par exemple une instance de la classe « Voiture » est : « Renault Laguna ».

NomDeLaClasse
attributs
opérations ( )

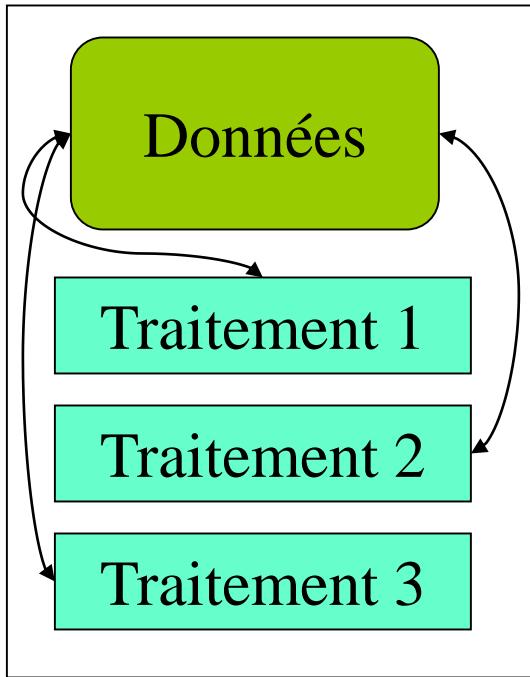
### **Représentation graphique d'une classe :**

Le premier compartiment contient le nom de la classe

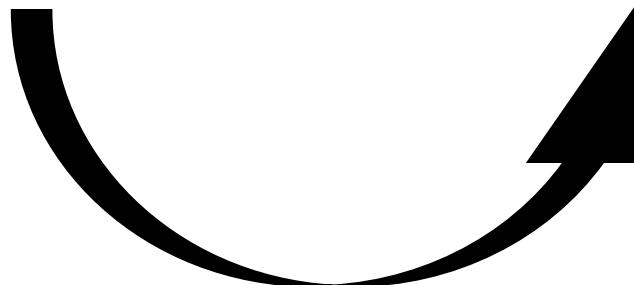
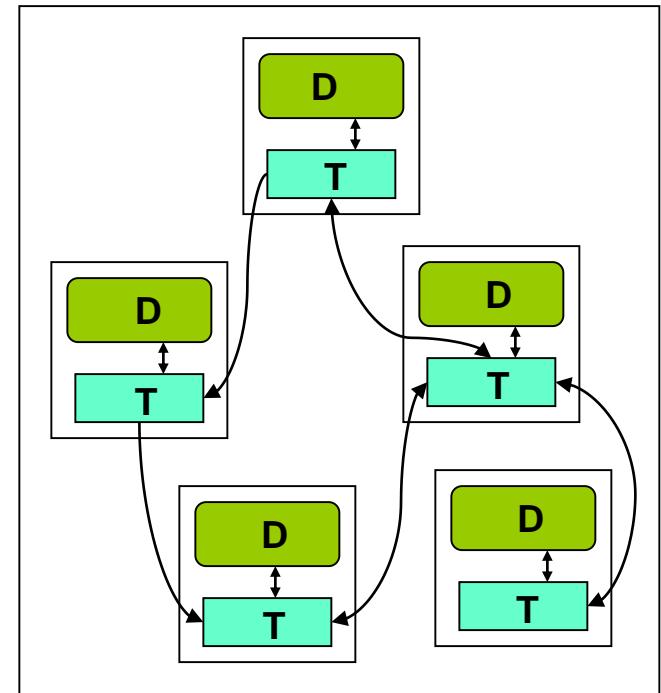
Le second compartiment contient la liste des attributs

Le troisième compartiment contient la liste des opérations

## Langages procéduraux et fonctionnels



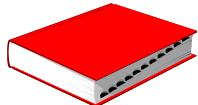
## Langages Orientés-Objet



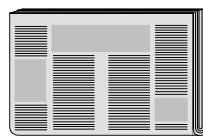
# Objet

- Approche procédurale :  
"Que doit faire mon programme ?"
- Approche orientée-objet :  
"De quoi doit être composé mon programme ?"
- Cette composition est conséquence d'un choix de modélisation fait pendant la conception

## Exemple: Gestion d'une bibliothèque



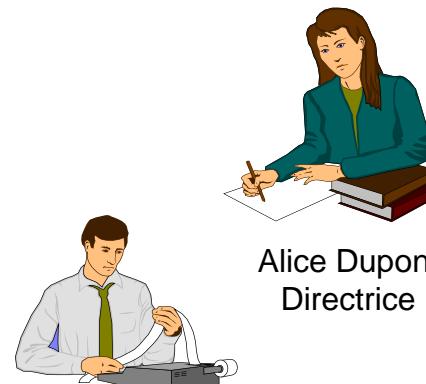
Germinal  
E. Zola



Le Monde



Le seigneur des anneaux  
J.R.R.Tolkien



Alice Dupont  
Directrice

Michel Martin  
Bibliothécaire



Arsène Deschamps  
Lecteur

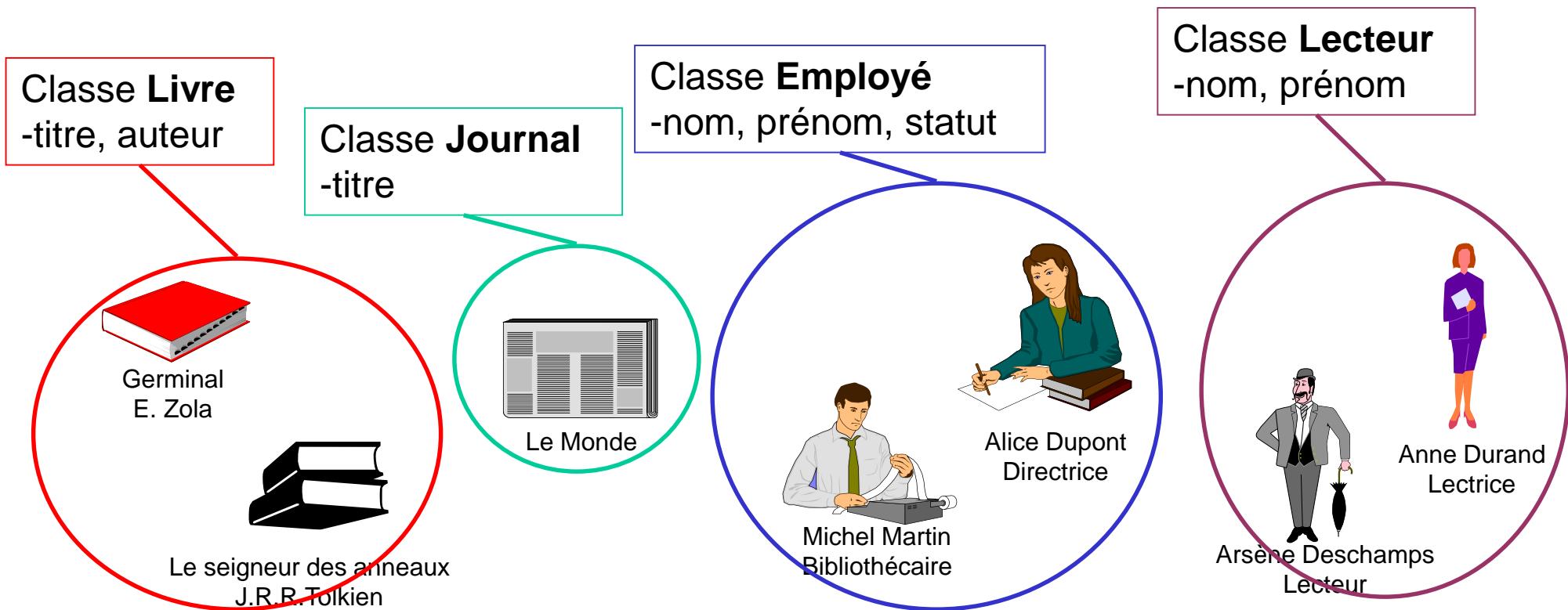


Anne Durand  
Lectrice

# Classe

Des objets similaires peuvent être informatiquement décrits par une même abstraction : une **classe**

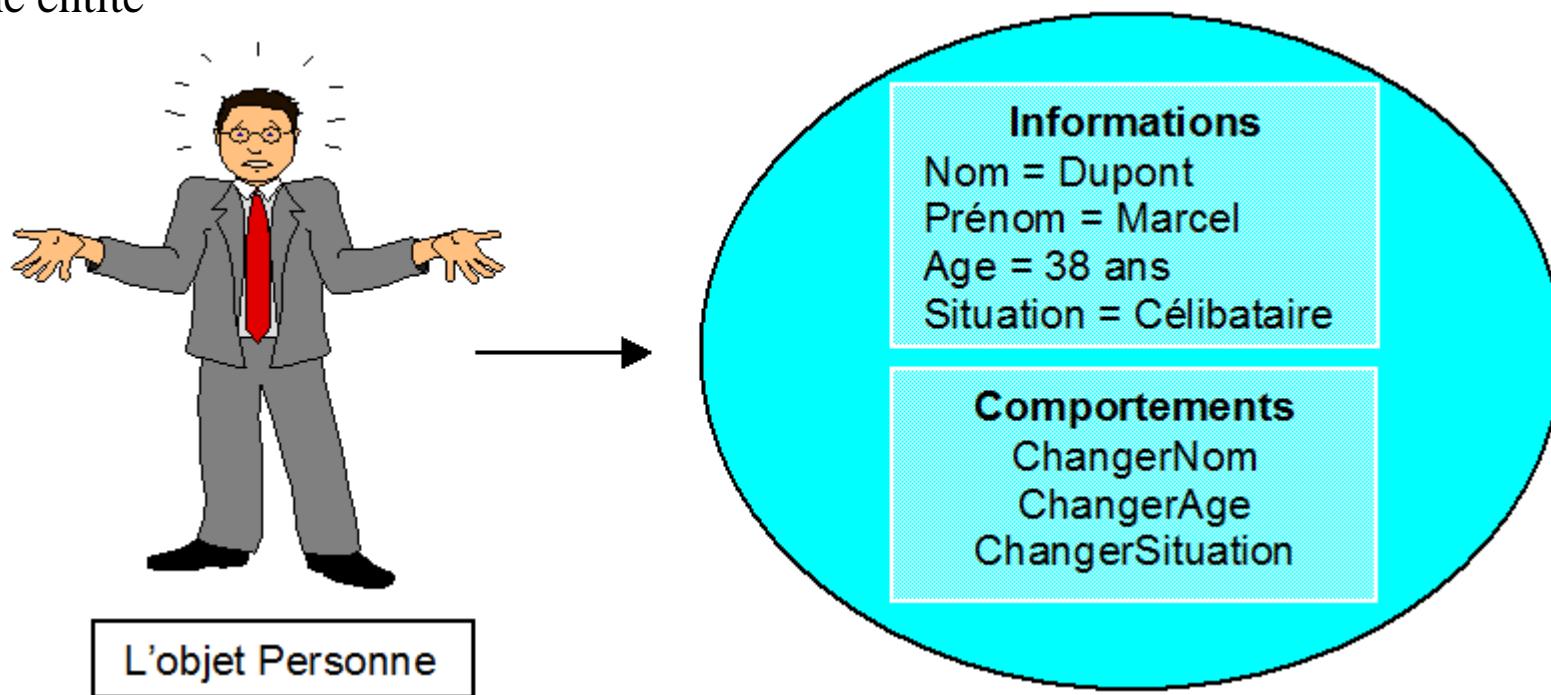
- même structure de données et méthodes de traitement
- valeurs différentes pour chaque objet



# PARADIGMES OBJET

## A- Encapsulation et abstraction

On dit que les informations (qui sont des données) et les comportements (qui sont les traitements ou encore méthodes ou opérations) d'un objet sont encapsulés. En effet celles-ci sont à l'intérieur d'une entité

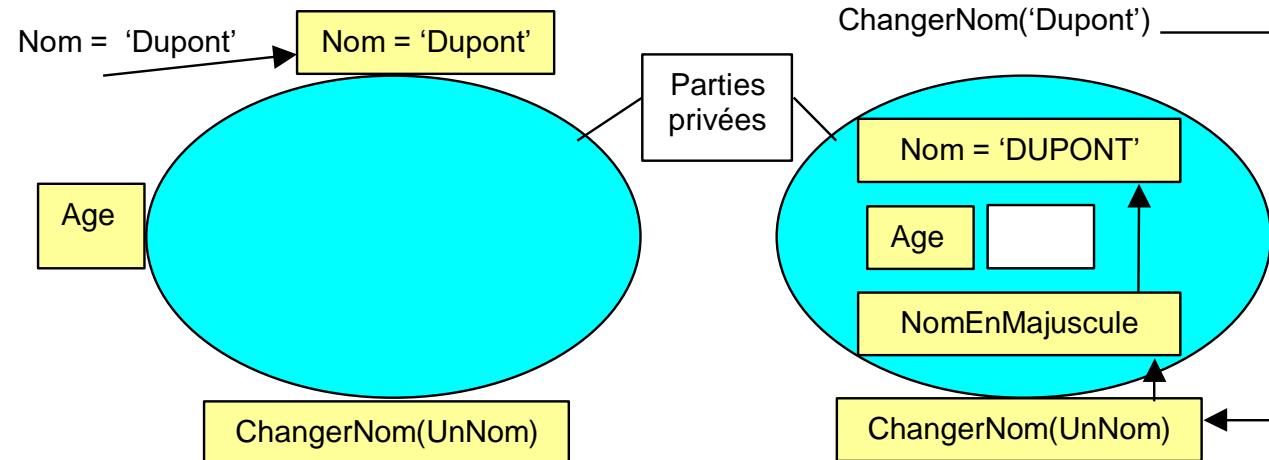


Les informations (données) et les comportements (traitements ou opérations) de l'objet «Personne» sont encapsulés c'est à dire regroupés dans une entité qui est l'objet «Personne».

**Intérêt de l'encapsulation :** On peut protéger le contenu des classes d'une manipulation maladroite et ou mal intentionnée.

Un objet est caractérisé par ses données et ses traitements mais il est aussi caractérisé par une partie publique, une partie privée et une partie implémentation, c'est ce que l'on appelle l'**abstraction**.

On dit que les données ont un accès privé c'est à dire que seuls les traitements de cet objet peuvent, le cas échéant, modifier ces données (attention les données peuvent être aussi publiques mais cela n'a aucun intérêt). Les traitements ont un accès privé ou public. Si les traitements sont publics, ceux-ci appartiennent à l'objet et peuvent être appelés et modifier les données. Si les traitements sont privés, alors seuls des traitements publics appartenant à l'objet pourront déclencher l'exécution des traitements privés (à condition qu'ils figurent dans le codage). Les traitements privés sont appelés **méthodes d'implémentation**



Les attributs et les méthodes sont publics. Les attributs peuvent être modifiés directement sans passer par les méthodes.

Les attributs sont privés. Les attributs ne peuvent être modifiés que par la méthode publique (ici «`ChangerNom`») qui fait appel à une méthode d'implémentation (ici «`NomEnMajuscule`»)

#### 4 niveaux de visibilité des attributs et des opérations :

- **public** : élément visible à tous les clients de la classe (+),
- **protégé** : élément visible aux sous-classes de la classe (#),
- **privé** : élément visible à la classe seule (-).
- **Package** : élément visible a toutes les classes de même package (~)

Nom_de_Classe
+Attribut public
#Attribut protégé
-Attribut privé
<u>Attribut de Classe</u> (Italique)
+Opération publique()
#Opération protégée()
-Opération privée()
<u>Opération de Classe()</u>

Syntaxe d'un attribut :

Visibilité nomAttribut:type=[valeur Initiale]

Syntaxe d'une méthode:

Visibilité nomMethode(liste des paramètres):typedeRetour

Exemple : Représentation de la classe VEHICULE

Nom de la Classe ←

Description des Propriétés ←  
( Données )

Description des Méthodes ←  
( Actions )

VEHICULE

- N° Véhicule : Chaîne de car.
- Marque : Chaîne de caractères
- Puissance fiscale : Entier court
- Vitesse maximale : Réel
- Vitesse courante : Réel

- + Créer un Véhicule ()
- + Détruire un Véhicule ()
- + Démarrer ()
- + Accélérer ( Taux : Réel )
- + Avancer
- + Reculer ( )

- attributs et opération soulignés : visibles globalement dans toute la portée lexicale de la classe

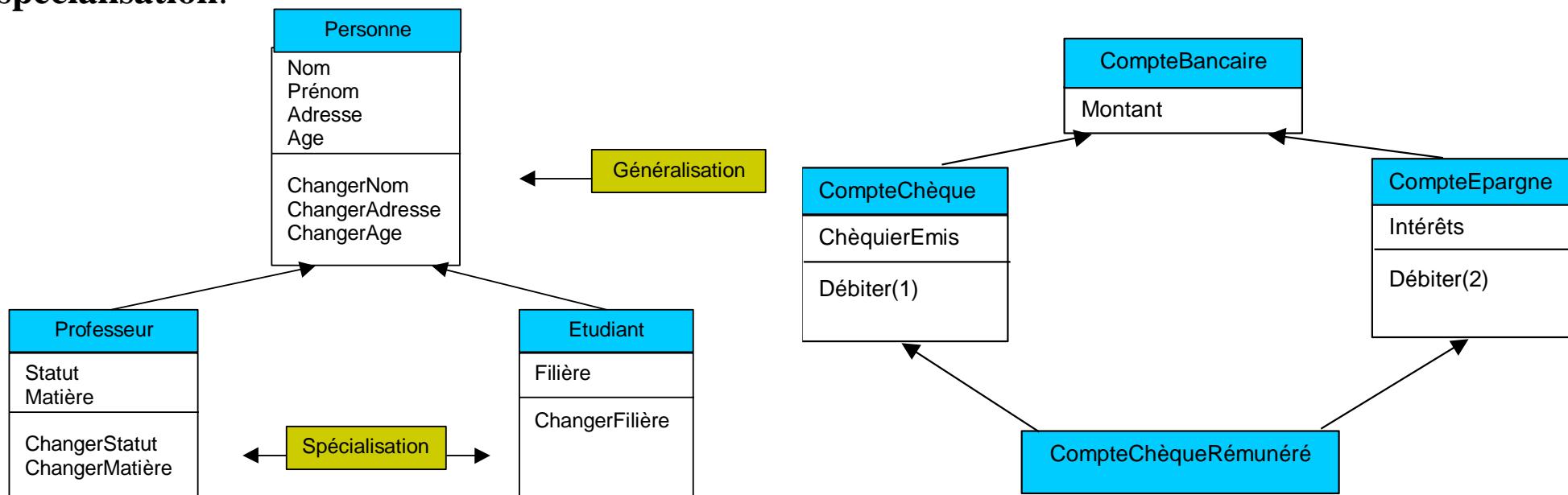
## B Héritage

On parle d'héritage lorsque l'on a affaire à des objets et des classes et de généralisation / spécialisation uniquement pour des classes.

On distingue deux types d'héritage :

- l'héritage simple
- l'héritage multiple

On va factoriser les attributs et les méthodes communs à plusieurs classes dans une classe principale : on parle de **généralisation**. Les classes dérivées deviennent des sous-classes : on parle de **spécialisation**.

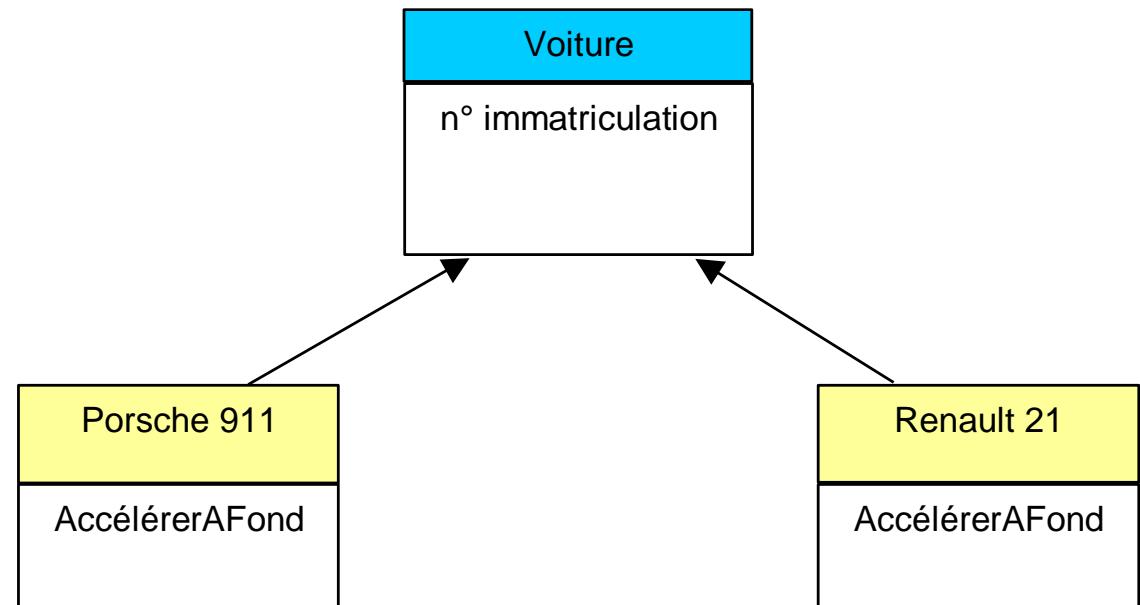


**Intérêts de l'héritage :** L'héritage permet la réutilisation du code. En effet lorsque l'on va instancier une classe spécialisée, le code des attributs et des méthodes de la classe héritée ne seront pas implémentés à nouveau. L'autre avantage de l'héritage et qu'il permet l'organisation hiérarchique des classes. C'est à dire qu'il rend plus aisés l'exploration et la maintenance d'une bibliothèque de classe pour une équipe de développement.

## C Polymorphisme

Le polymorphisme est la possibilité pour un même message de déclencher des traitements différents, suivant les objets auxquels il est adressé. Le mécanisme de polymorphisme permet donc de donner le même nom à des traitements différents

**Intérêt du polymorphisme :** Le polymorphisme permet de donner le même nom à des traitements différents ou encore, permet le choix dynamique d'un traitement en fonction de l'objet auquel il est appliqué (en effet le choix de la méthode polymorphe à exécuter est déterminé à l'exécution du programme. C'est pour cela qu'on dit qu'elle est dynamique, par opposition à statique qui est déterminé lors de la compilation).



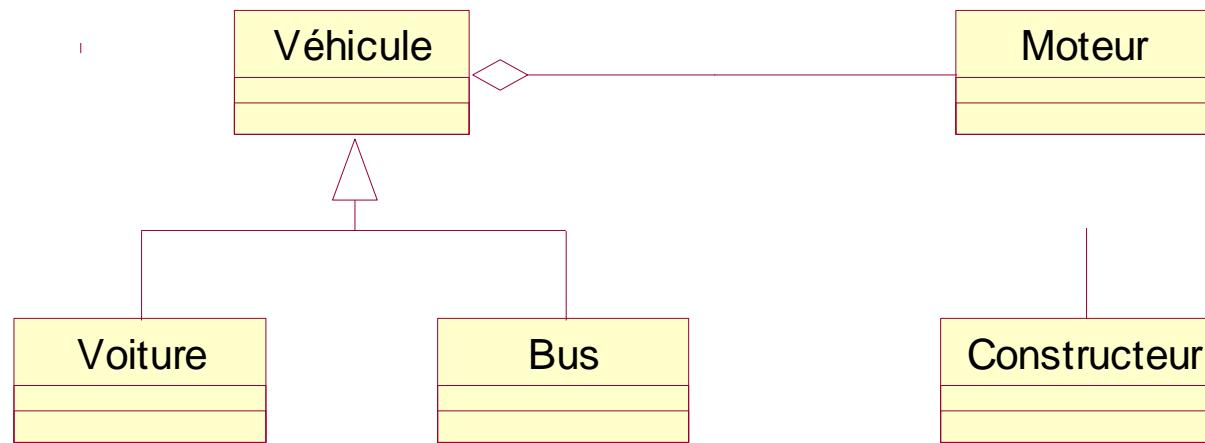
## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe

#### Définition

C'est un diagramme qui montre une collection d'éléments statiques (classes), leur contenu et les relations entre eux.

Exemple :



**Figure IV-7 : Exemple de diagramme de classe (Atelier Rational Rose)**

Chaque diagramme de classe est formé des entités suivantes :

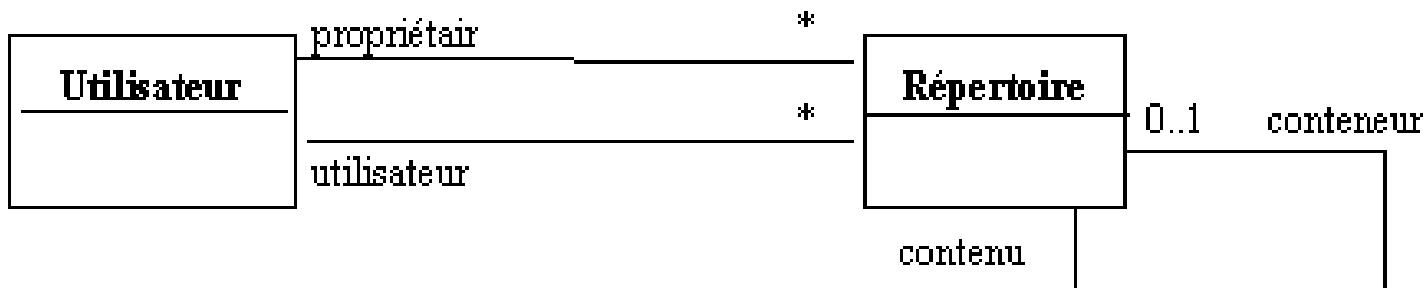
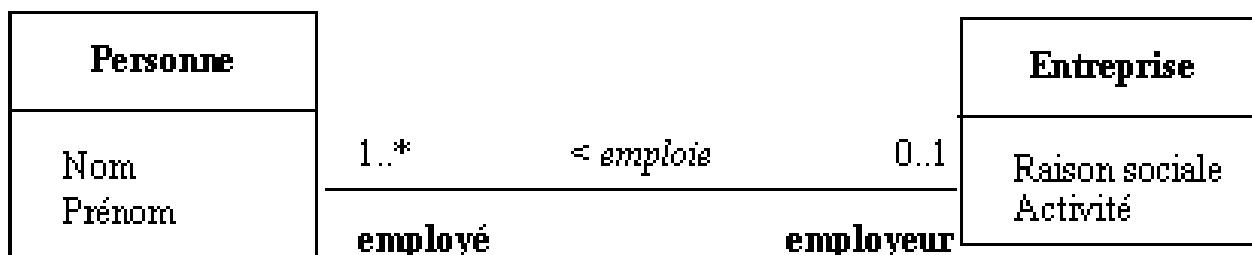
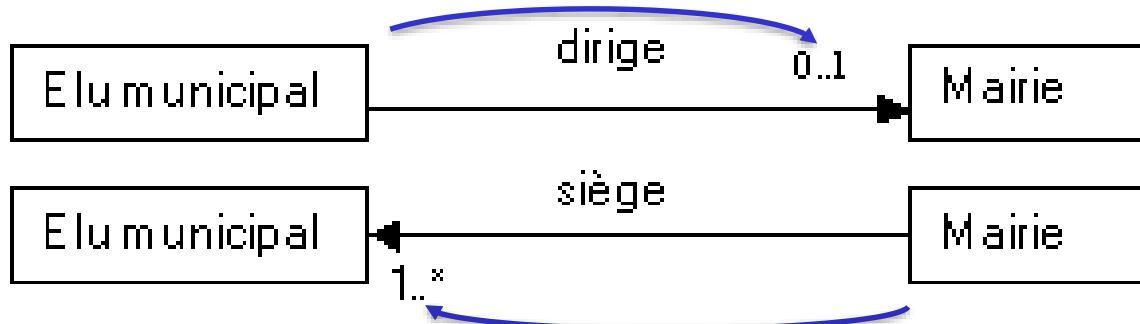
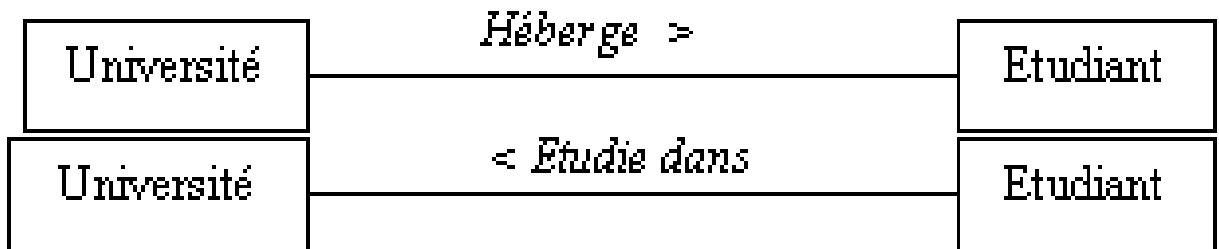
- ⇒ les **classes** : Les classes se désignent par un nom (1<sup>re</sup> partie), contiennent des attributs (2<sup>ème</sup> partie) et des méthodes associées (3<sup>ème</sup> partie) :
- ⇒ les **relations** interclasses : Elles sont appelées associations. On a défini différents types d'associations :
  - association simple (binaire ou n-aire)
  - agrégation, composition
  - héritage (spécialisation, généralisation)
- ⇒ les **noms de rôle** : ceux sont les noms des relations interclasses ;

- ⇒ les **multiplicités** : associées aux relations, les multiplicités permettent de déterminer le nombre d'occurrence d'une classe par rapport à une autre classe en utilisant le nom de rôle ;

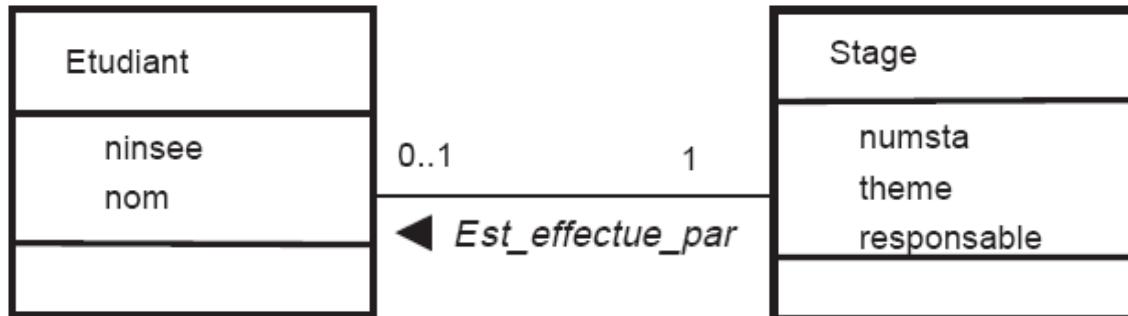
La notation générale adoptée est :      **min .. max**

<b>1</b>	obligatoire
<b>0..1</b>	optionnel
<b>0..*</b> ou <b>*</b>	quelconque (cas général)
<b>1..*</b>	au moins 1
<b>1..5, 10</b>	entre 1 et 5, ou 10

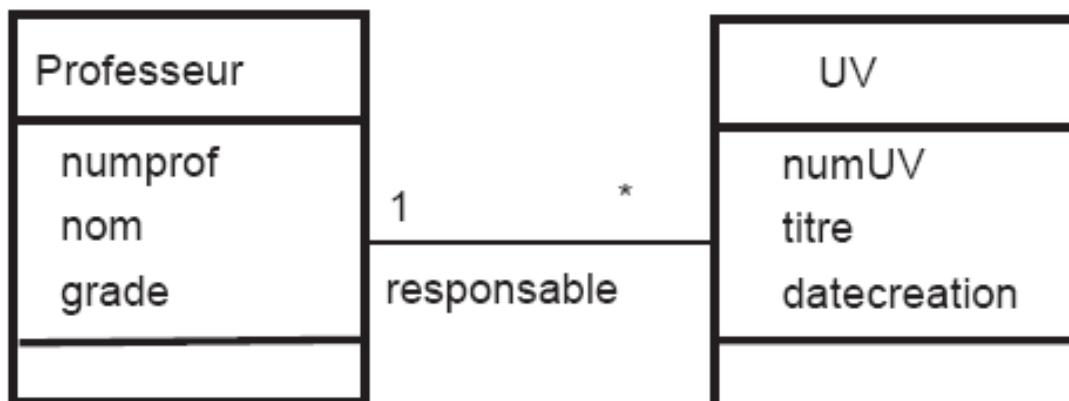
**Association Simple binaire** une association binaire exprime une relation sémantique bidirectionnelle entre deux classes, le sens d'une association peut être précisé par une flèche. Exemples :



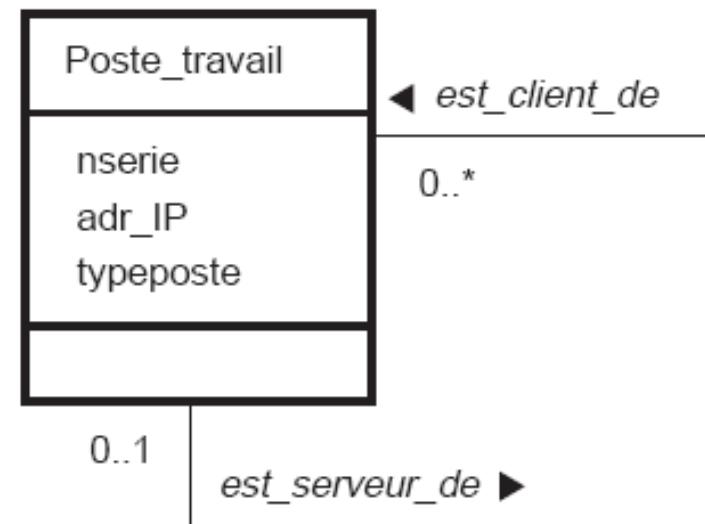
# Types d'associations binaires



*Association un-à-un UML*

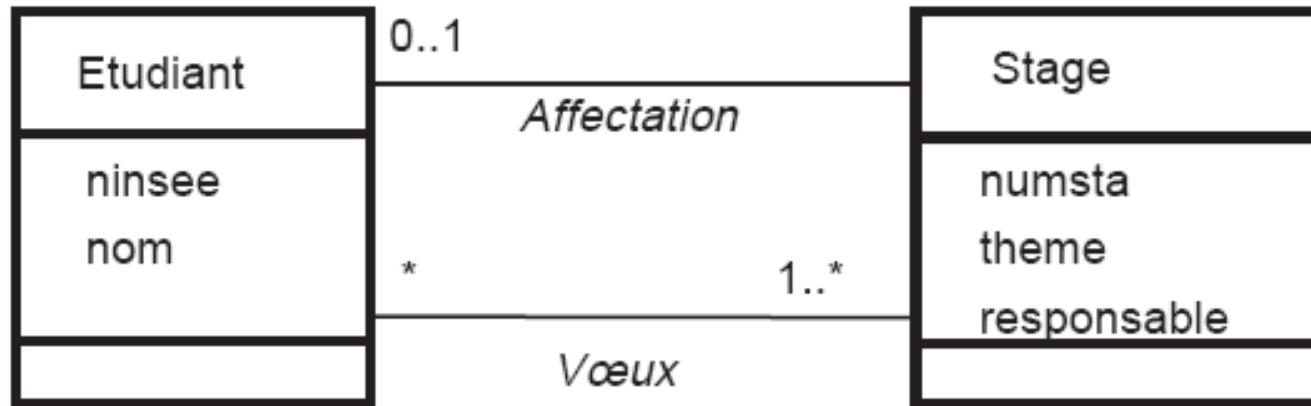


*Association un-à-plusieurs UML*

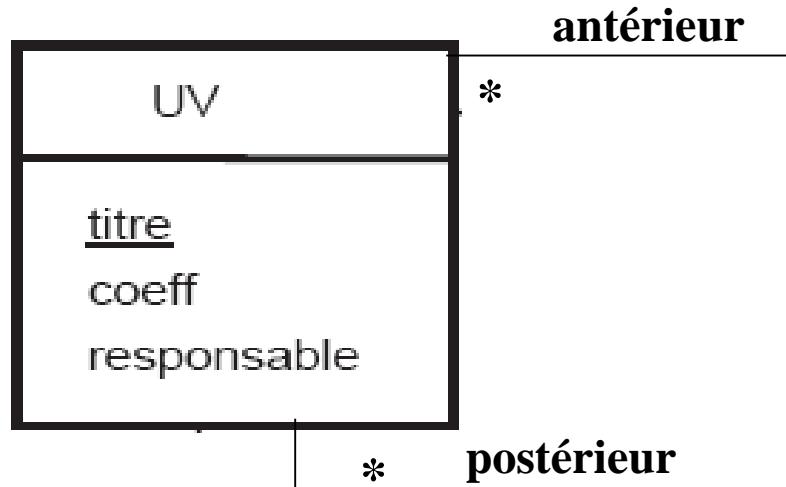


*Association un-à-plusieurs réflexive UML*

## Types d'associations binaires

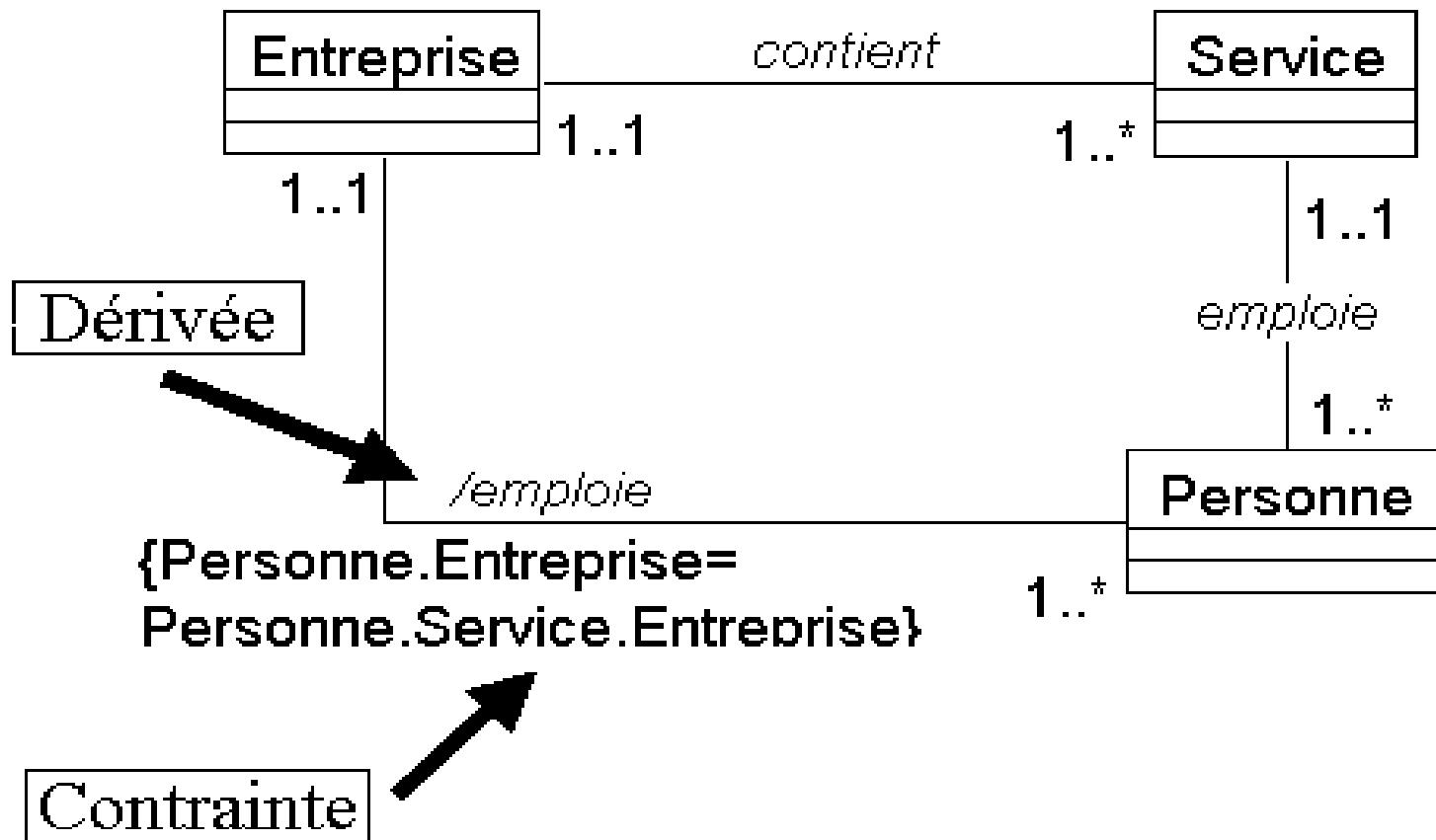


L'association « Vœux » est de type plusieurs à plusieurs sans attribut



Association plusieurs à plusieurs réflexive et sans attribut

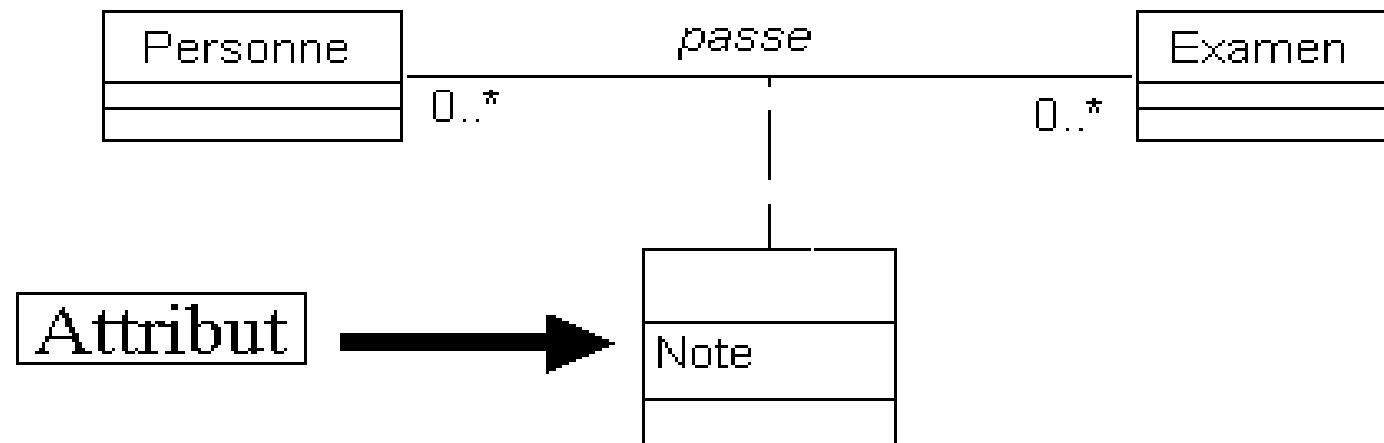
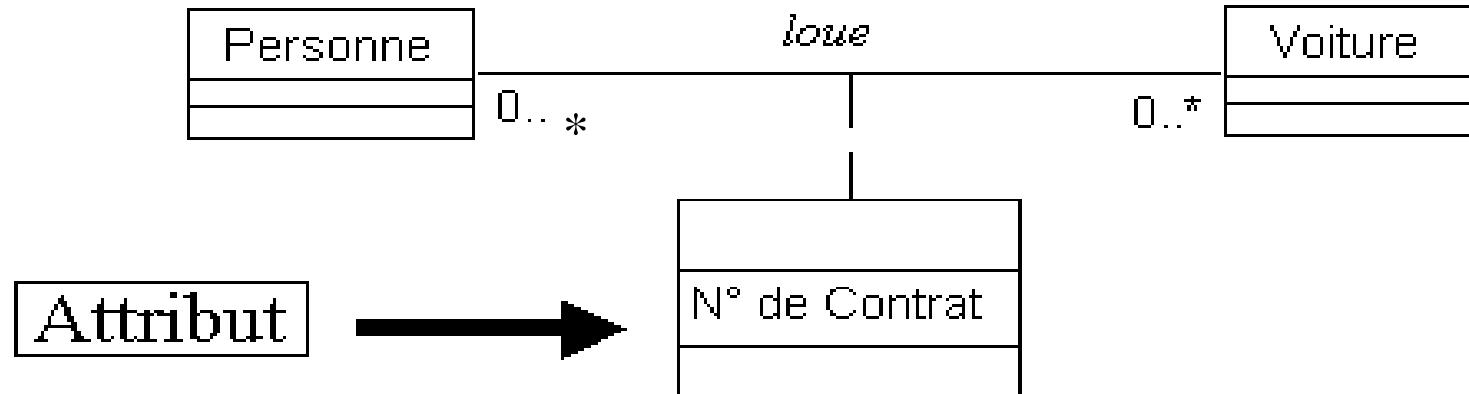
- Une **association dérivée** est une association qui peut se **déduire** des associations existantes



# La Phase d'élaboration

## B – Le Diagramme de Classe : les Attributs d’association ( Suite )

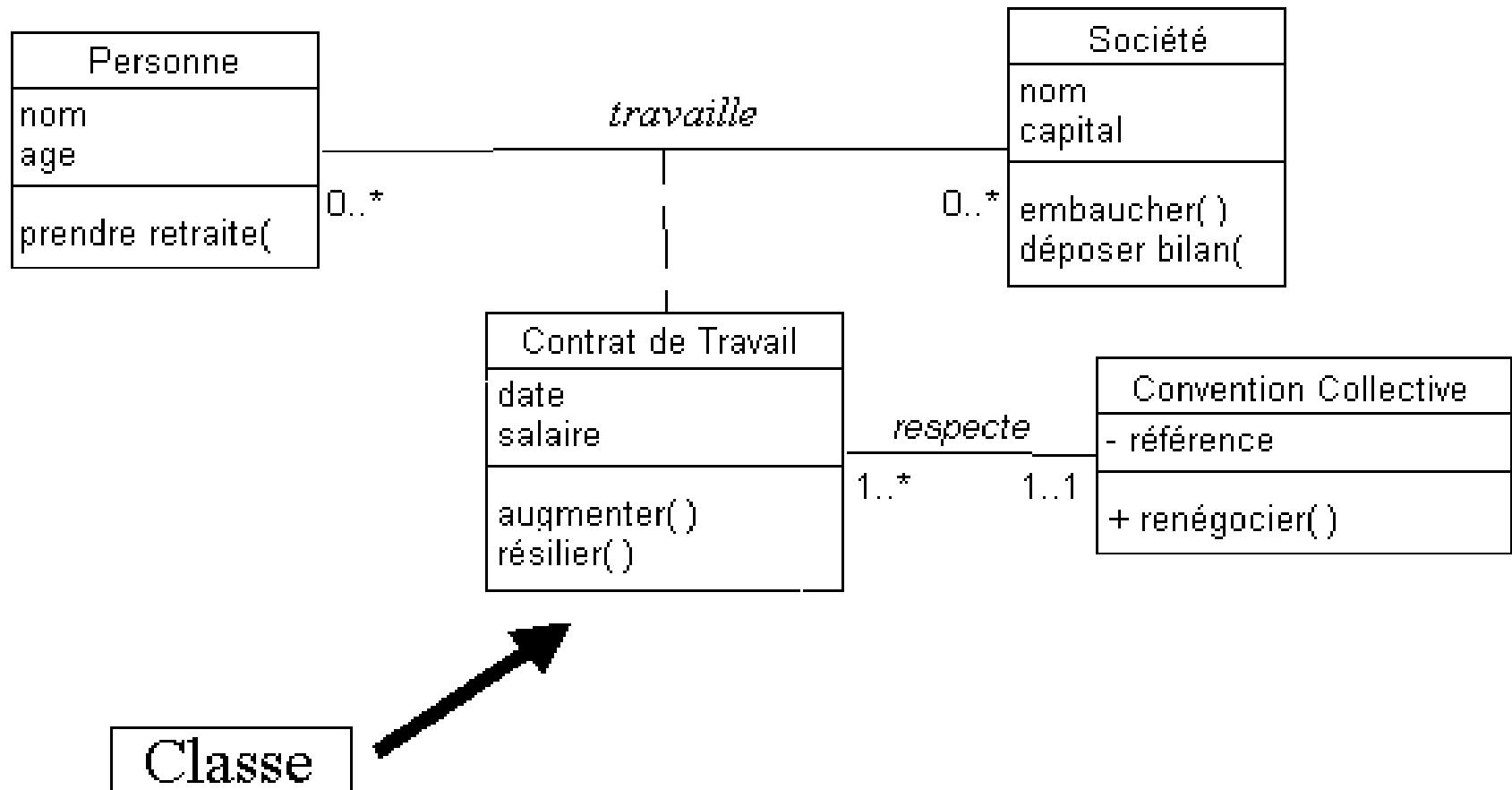
- **Attribut d’association = propriété du lien entre deux objets**



# La Phase d'élaboration

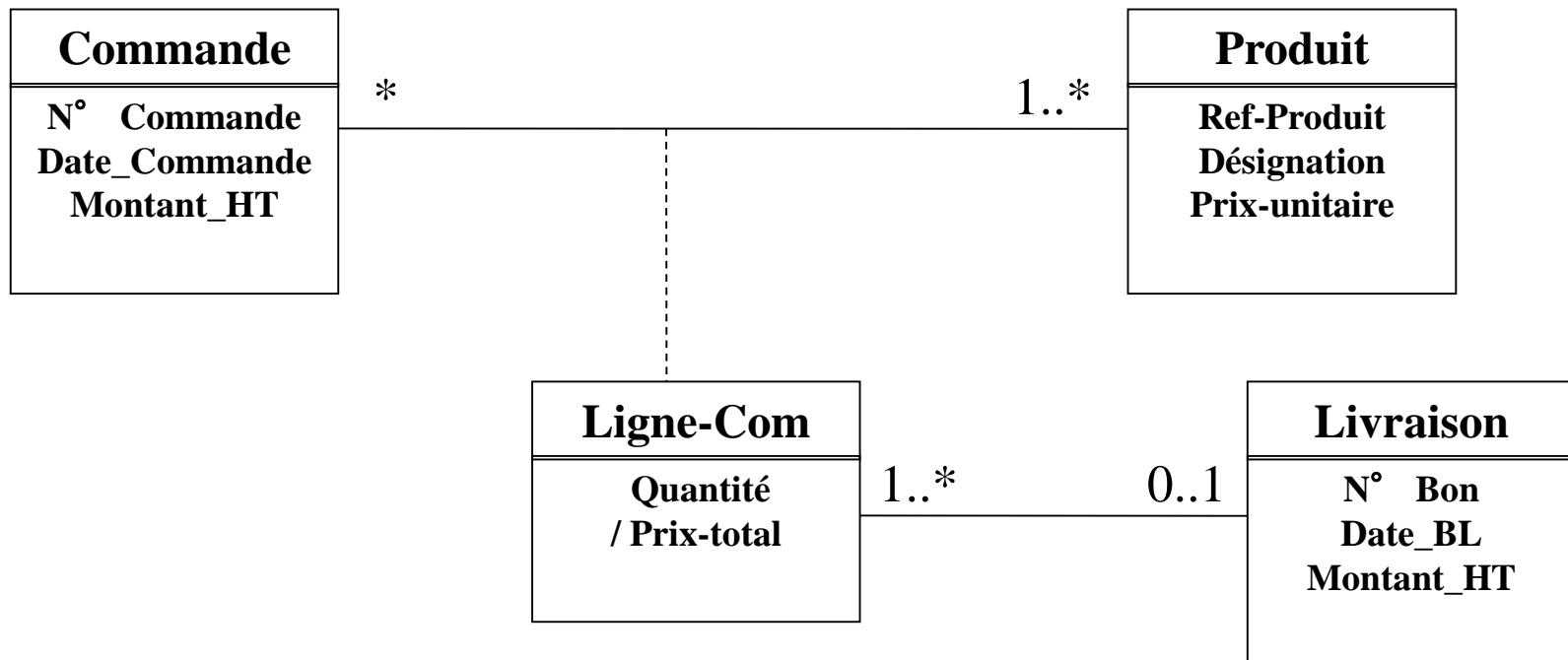
## B – Le Diagramme de Classe : Classe d’association ( Suite )

- **Classe d’association** = Elément ayant à la fois les propriétés d'une classe et d'une association



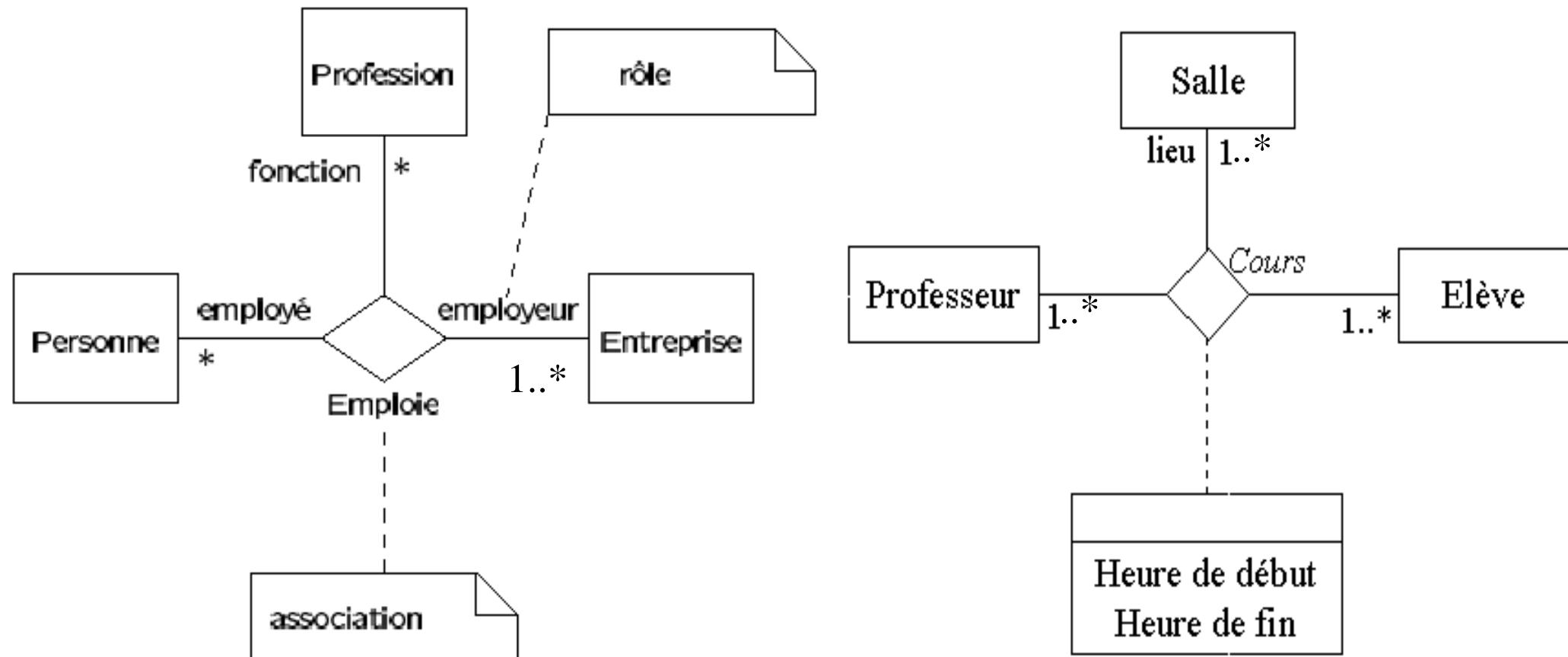
## B10 – Classe d’association ( Suite )

Une classe associative dans UML ou « classe d’association » est une classe qui peut participer à d’autres associations ( association d’associations dans l’approche Merise )



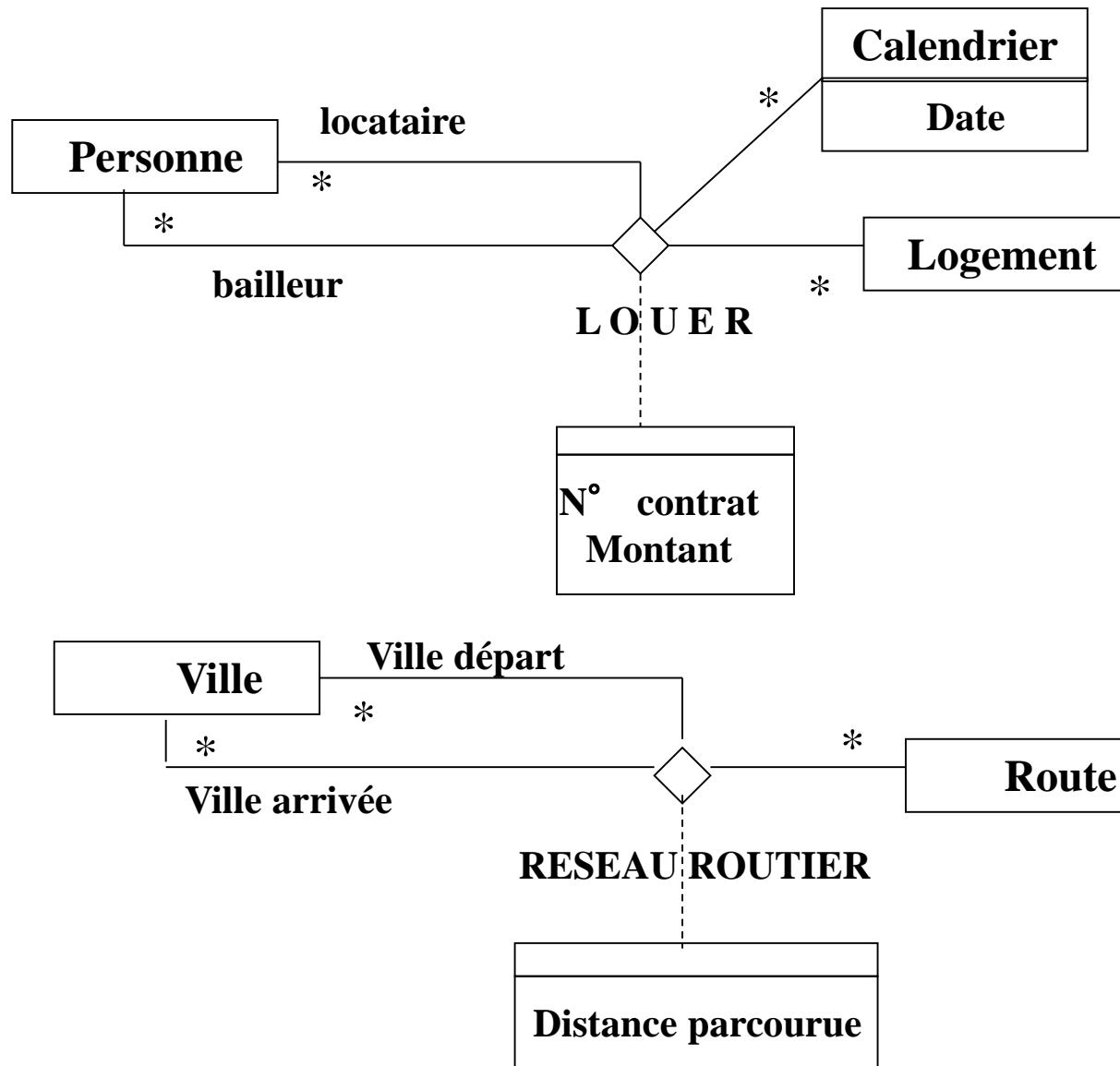
## B – Le Diagramme de Classe : Association n-aire ( Suite )

- **Association n-aire** = Une association parmi 3 classes ou plus. Chaque instance de l'association est un n-tuple de valeurs des classes respectives.

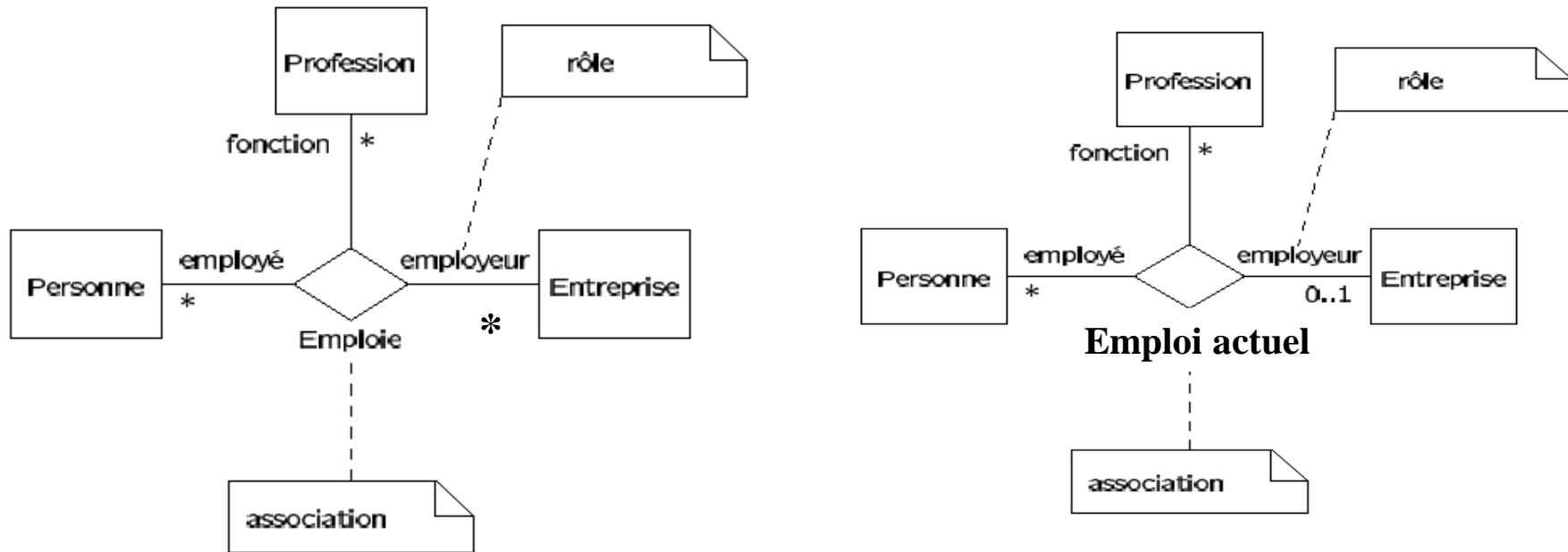


## 4.2.1 - La Phase d'analyse

### B8 – Les Association n-aires réflexives



## B8 – Les Association n-aires : lecture des multiplicités



Si (A,B,C) représente un triplet d'instances de type ( Personne,Profession,Entreprise ) :  
Pour obtenir la multiplicité côté C , on fixe un couple d'instances (A,B) et on examine  
le nombre d'instances Cmin et Cmax qui lui sont reliées à travers l'association .

Même raisonnement pour trouver les multiplicités côté A ou côté B .

Exemple : Une personne ayant une profession ( couple (A,B) fixé ) peut être employée  
dans 0 à plusieurs entreprises selon l'association « Emploie » alors qu'elle peut  
posséder 0 ou 1 employeur ( entreprise ) selon l'association « Emploi actuel » .

## **4.2.1 - La Phase d'analyse**

### **B11 - Personnalisation d'une association**

Une association doit être personnalisée dans les cas suivants :

#### **Cas n° 1 : Arité de l'association $\geq 2$**

La classe d'association correspondante peut être identifiée à l'aide d'un nouvel attribut unique ( autre que « l'identifiant composé » de l'association ) et cette classe doit participer à d'autres associations

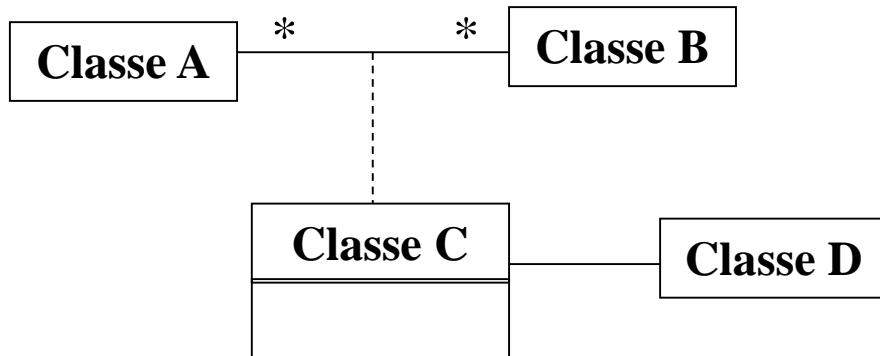
#### **Cas n° 2 : Arité de l'association $\geq 3$**

Une dépendance fonctionnelle supplémentaire existe au niveau de l'association ( une cardinalité maxi = 1 se trouve sur l'une des pattes d'une association d'arité  $\geq 3$  )

## 4.2.1 - La Phase d'analyse

### B11 - Personnalisation d'une association ( cas n° 1 )

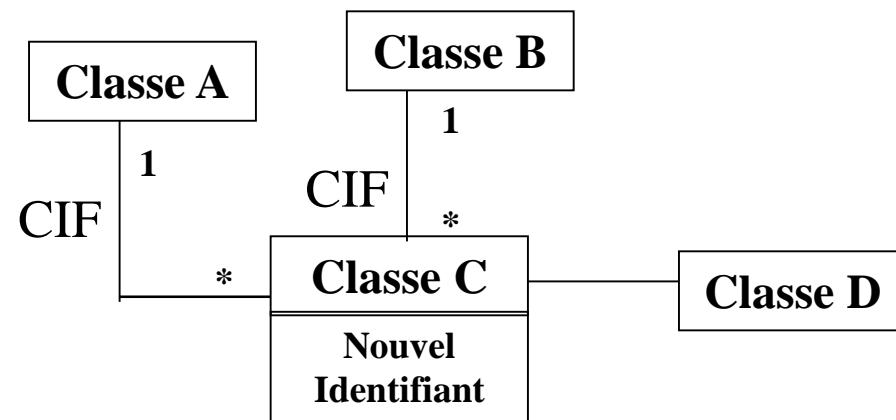
Cas n° 1 :



Entité	Identifiant
C avant personnalisation	Identifiant de l'association = (Ao, Bo)
C après personnalisation	Nouvel identifiant Co

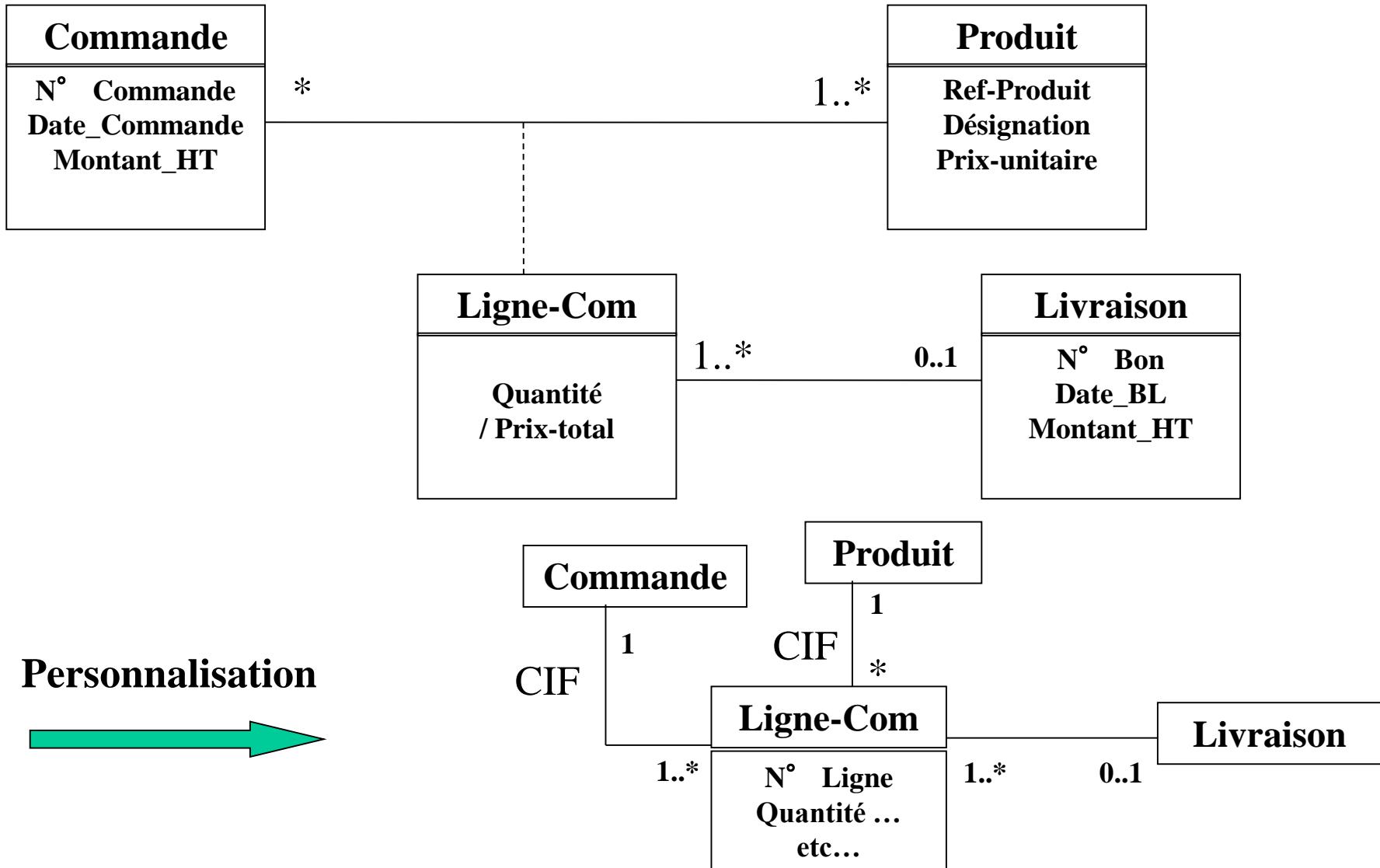
La décomposition de l'association fait apparaître des contraintes d'intégrité fonctionnelles ( CIF ) = associations binaires hiérarchiques fortes et stables

Personnalisation



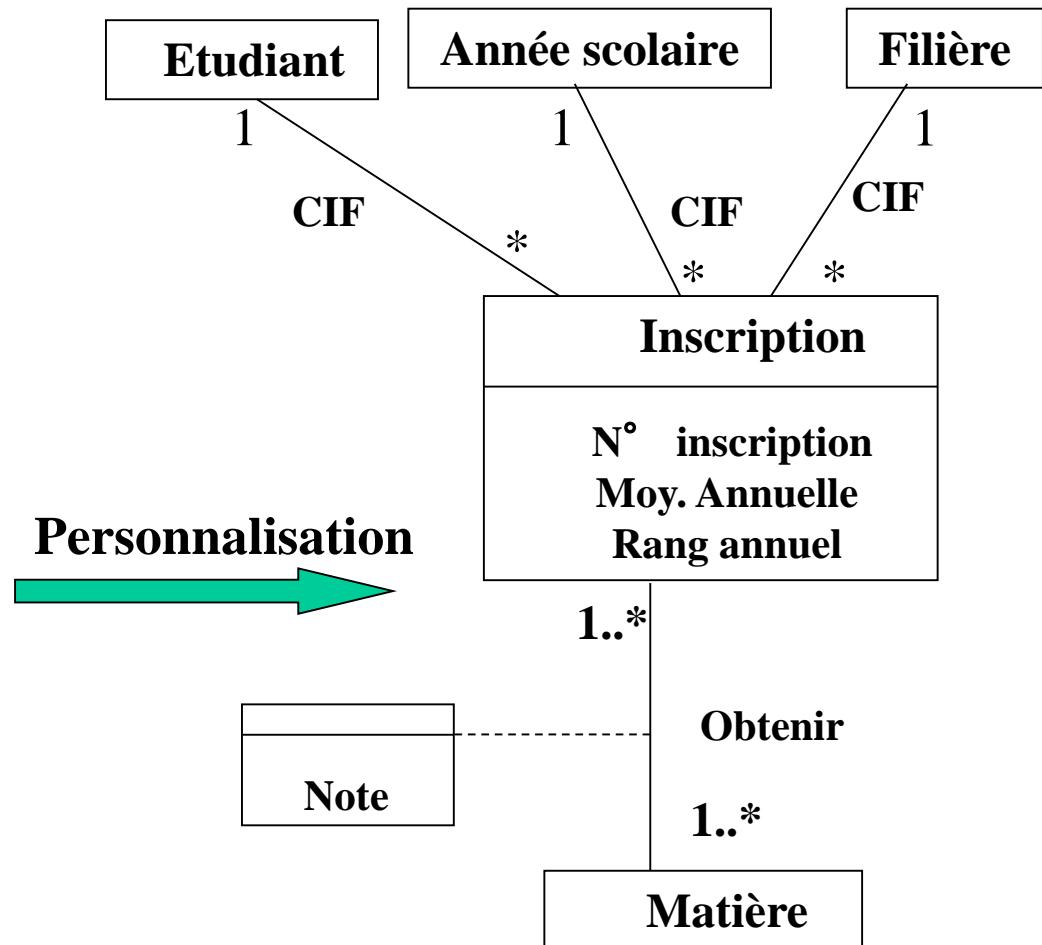
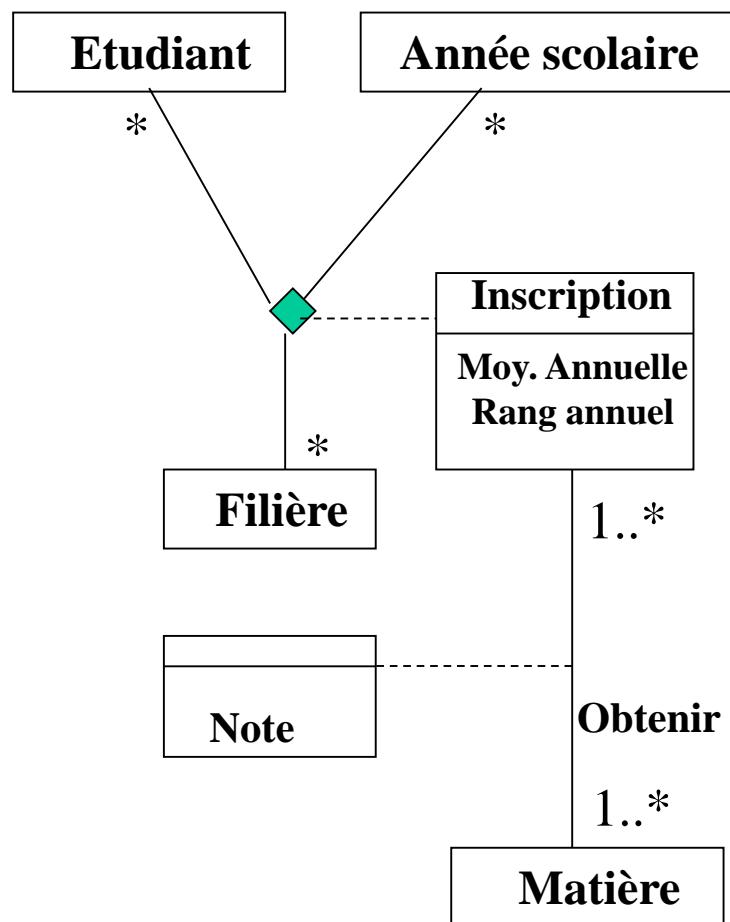
## 4.2.1 - La Phase d'analyse

### B11 - Personnalisation d'une association ( cas n° 1 ) : Exemple 1



## 4.2.1 - La Phase d'analyse

### B11 - Personnalisation d'une association ( cas n° 1 ) : Exemple 3



## 4.2.1 - La Phase d'analyse

### B11 - Personnalisation d'une association ( cas n° 1 ) : Exemple 3 ( suite )

Si A = Année scolaire ; B = Etudiant ; C = Filière ; D = Matière ; E = Inscription

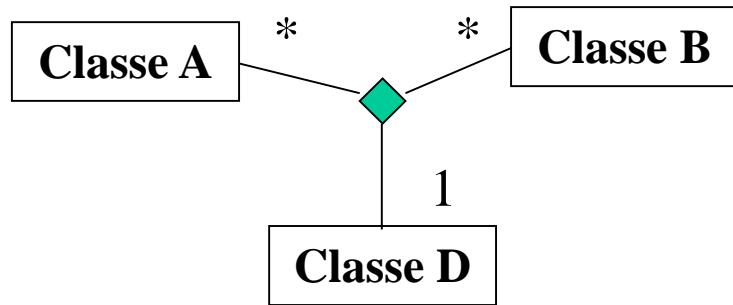
Graphe des dépendances fonctionnelles

Avant personnalisation	Après personnalisation
<p><b>Ao , Bo , Co</b> → <b>Moy. Annuelle</b> → <b>Rang annuel</b></p> <p><b>( Ao,Bo,Co) , Do</b> → <b>Note matière</b></p>	<p><b>Eo</b> → <b>Moy. Annuelle, Rang annuel</b></p> <p><b>Eo</b> → <b>Ao , Bo , Co</b></p> <p><b>Eo , Do</b> → <b>Note matière</b></p>

## 4.2.1 - La Phase d'analyse

### B11 - Personnalisation d'une association ( cas n° 2 )

Cas n° 2 :

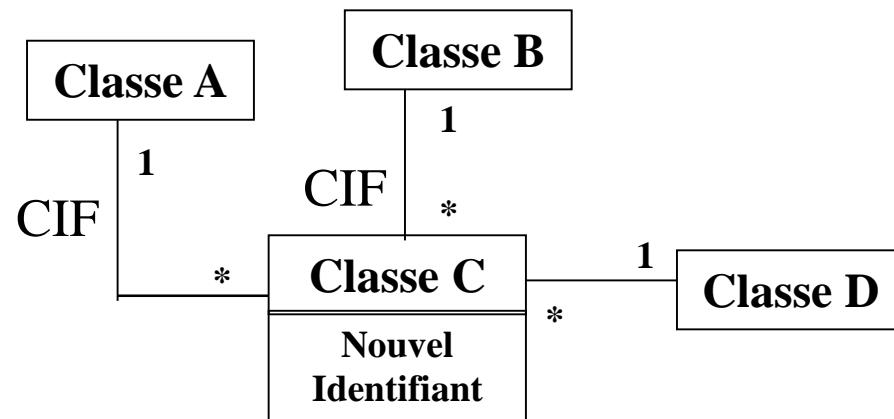


L'association ternaire cache la DF suivante :

( Ao , Bo )  $\longrightarrow$  Do

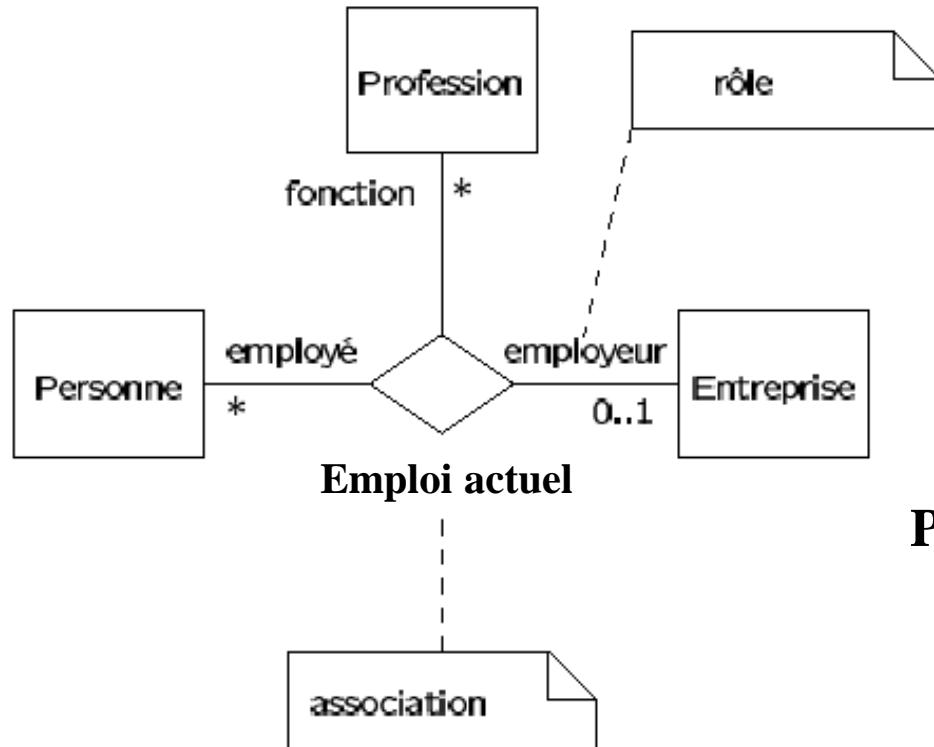
La décomposition de l'association fait apparaître une nouvelle classe C avec des contraintes d'intégrité fonctionnelles ( CIF )  
= Associations binaires hiérarchiques fortes et stables

Personnalisation

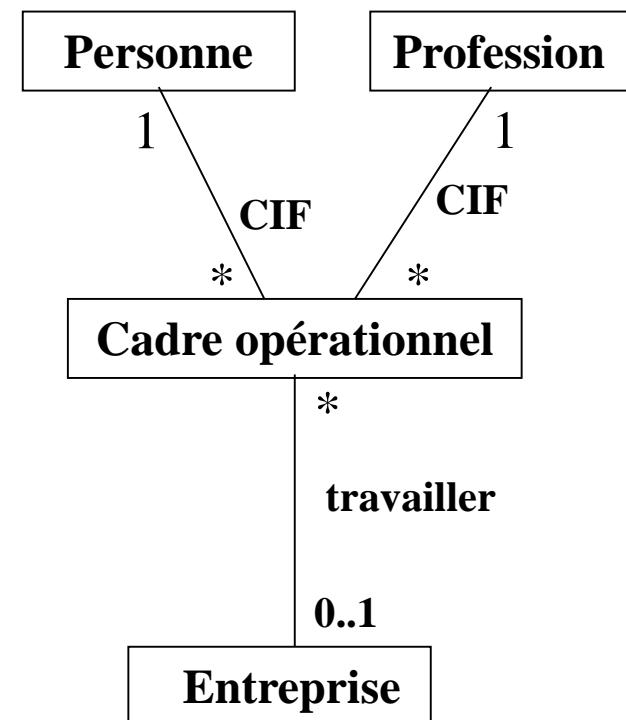


## 4.2.1 - La Phase d'analyse

### B11 - Personnalisation d'une association ( cas n° 2 ) : Exemple



Personnalisation  
→



L'association ternaire cache la DF :

(Personne , Profession ) → Entreprise

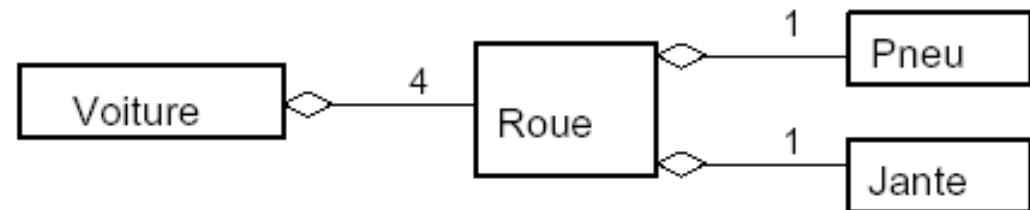
## B – Le Diagramme de Classe : Compositions et Agrégations ( Suite )

### Agrégation et composition :

Les relations d'agrégation et de composition sont 2 types particuliers d'associations permettant d'exprimer l'appartenance d'une partie à un tout .

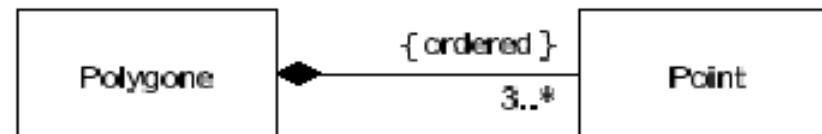
- **Agrégation**

- Association entre une classe de type « ensemble » avec plusieurs classes de type « éléments »



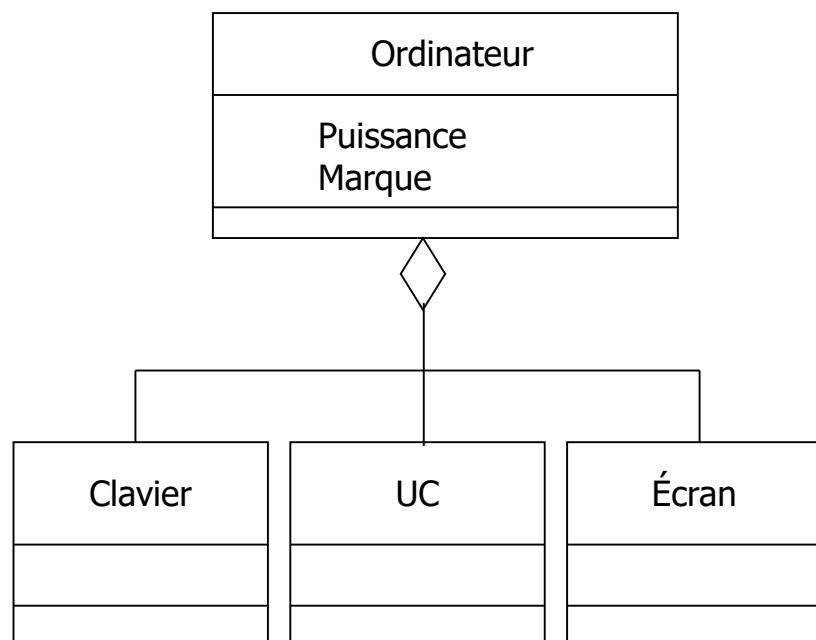
- **Composition**

- Agrégation avec une contrainte de durée de vie
  - La suppression de la classe « composé » implique la suppression des classes « composant »



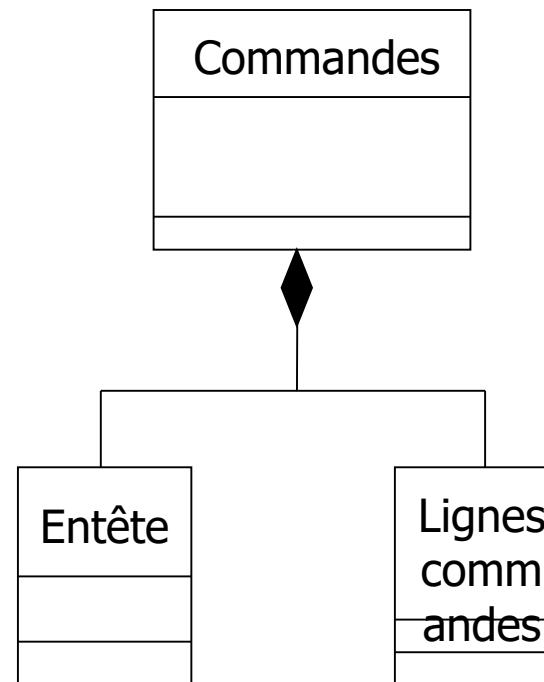
# L'AGREGATION

- L'**agrégation** est une relation de type composé – composant, maître – esclave ou un lien de type « ensemble » comprenant des « éléments ».



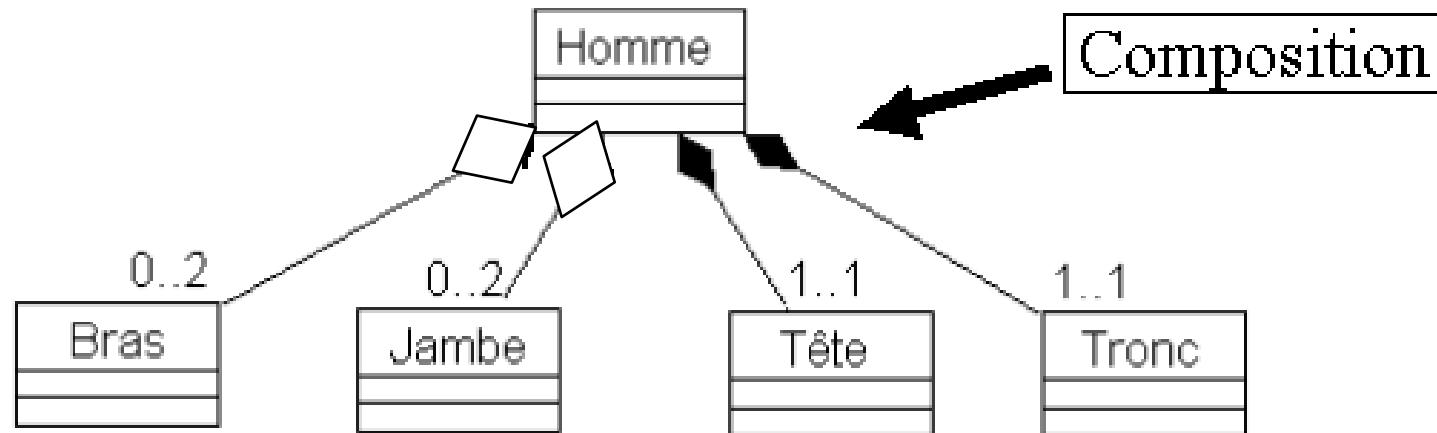
# LA COMPOSITION

- La **composition** est une relation d'agrégation dans laquelle il existe une contrainte de durée de vie entre la classe « Composé » et la ou les classes « Composant ». Autrement dit la suppression de la classe « Composé » implique la suppression de la ou des classes « Composant »

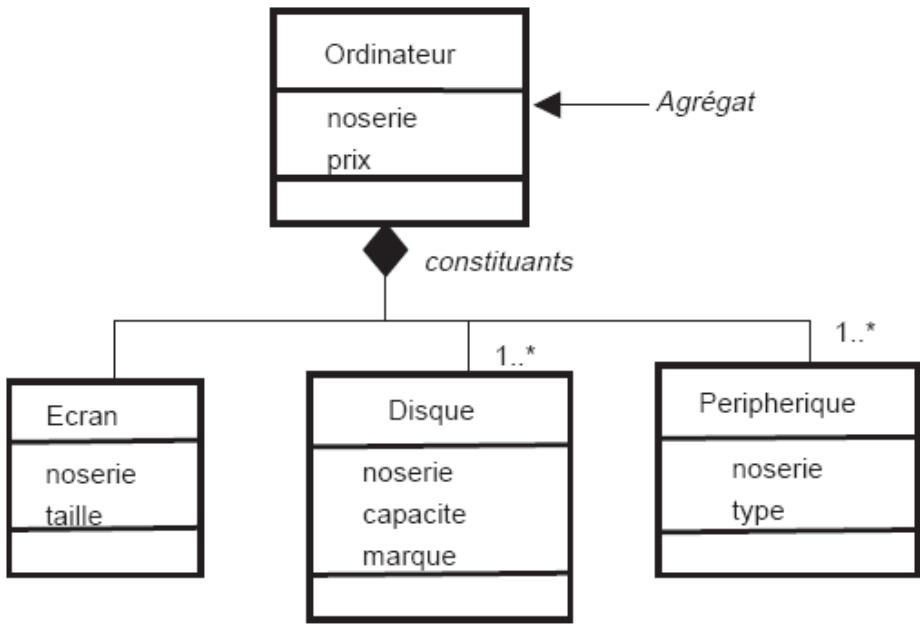


## B – Le Diagramme de Classe : Compositions et Agrégations ( Suite )

### Autres Exemples

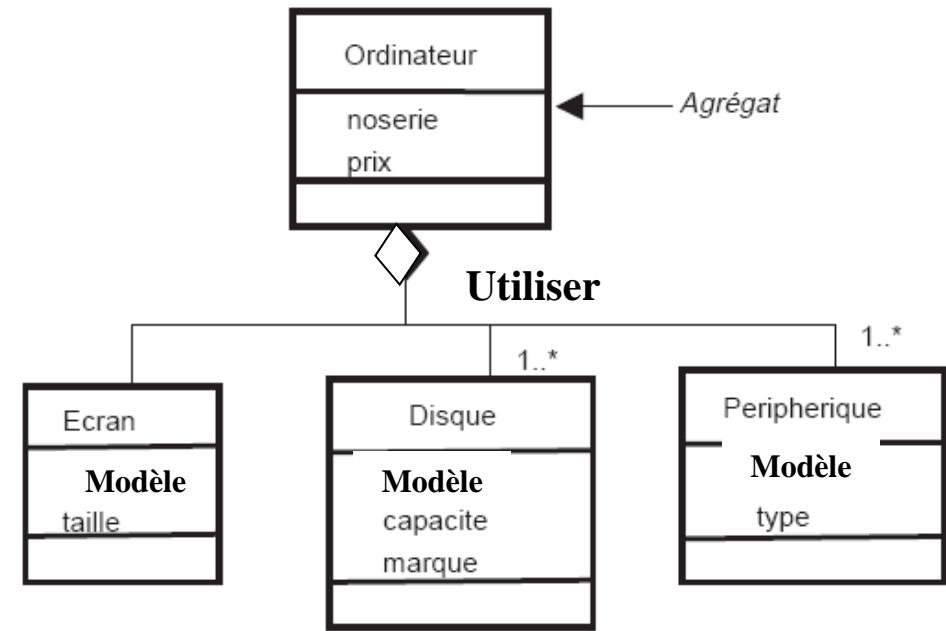


## B12 – Associations spéciales ( suite 5 ) : La composition et l’agrégation



Composition

L’écran , le disque et les périphériques sont des pièces réelles. Elles sont toutes identifiées par un numéro de série et sont les constituants d’un ordinateur à un instant donné .



Agrégation

L’écran , le disque et les périphériques sont des modèles de pièces. Ces modèles peuvent être employés dans l’assemblage de plusieurs ordinateurs .

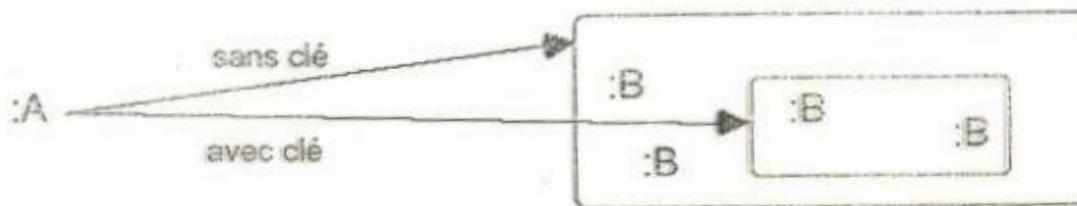
## **Relations entre classes : Association qualifiée**

- Restreindre la portée d'une association à quelques éléments ciblés (comme un ou plusieurs attributs) de la classe. Les éléments ciblés sont déterminés par un qualificatif.
- Un qualificatif agit toujours sur une association dont la multiplicité est plusieurs (avant que l'association ne soit qualifiée) du côté cible.
- L'association est appelée association qualifiée.
- Représenter par un rectangle positionné entre la terminaison d'association et la classe cible.

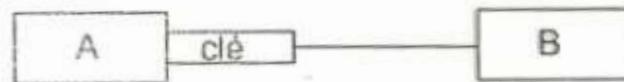
## 4.2.1 - La Phase d'analyse

### B6 – Type d'associations binaires : Association de qualification

Une association qualifiée ou « qualification » consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association



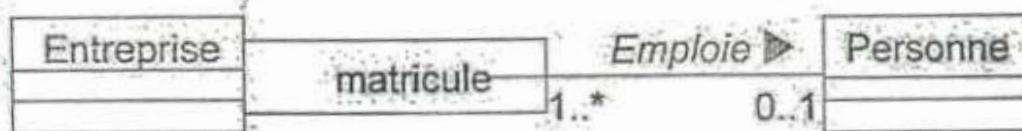
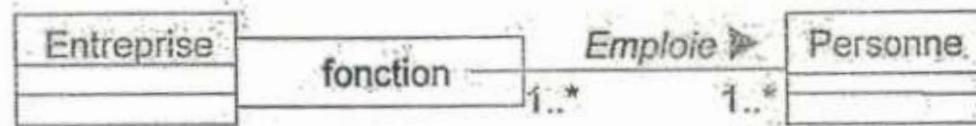
Cas général :



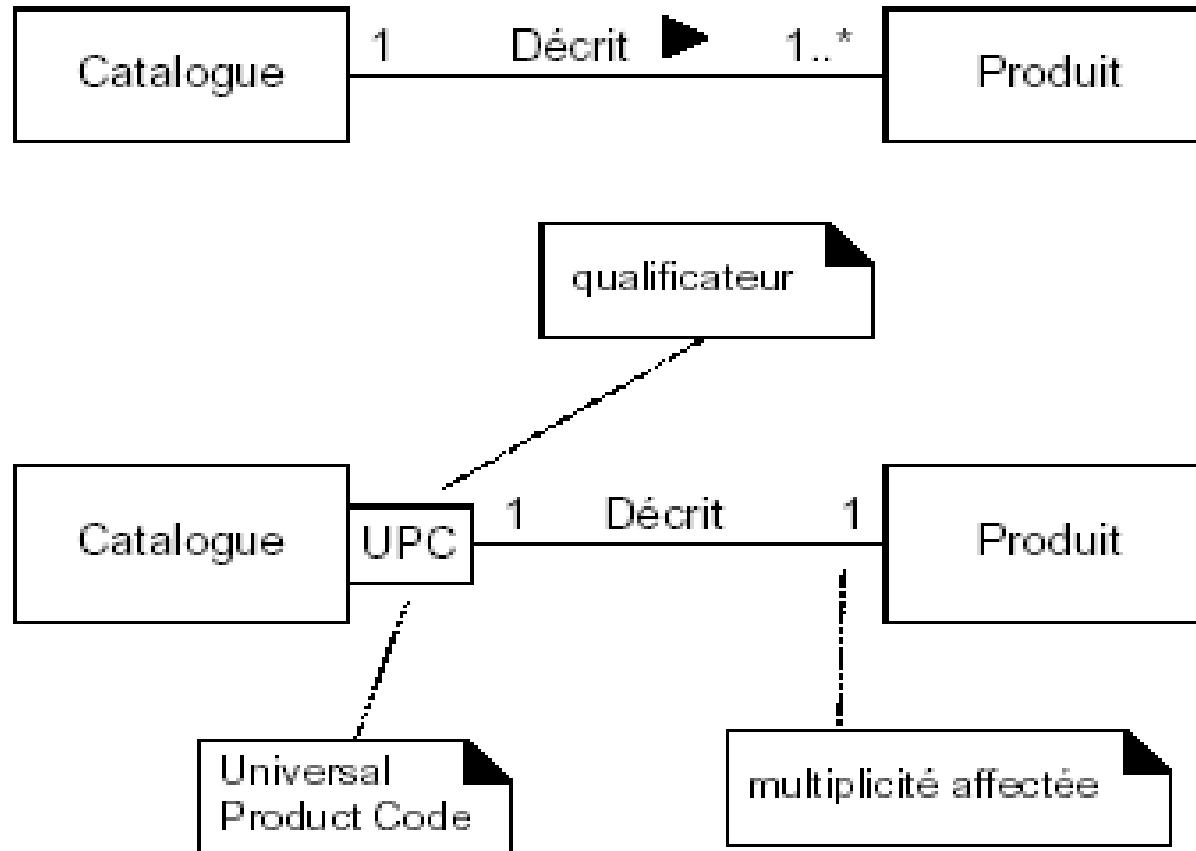
Exemples :



combinaison d'une ligne et d'une colonne pour identifier une case de l'échiquier



# Associations : associations qualifiées

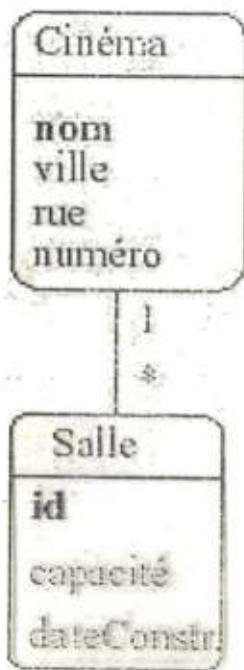


## 4.2.1 - La Phase d'analyse

### B7 – Type d'associations binaires : Association relative

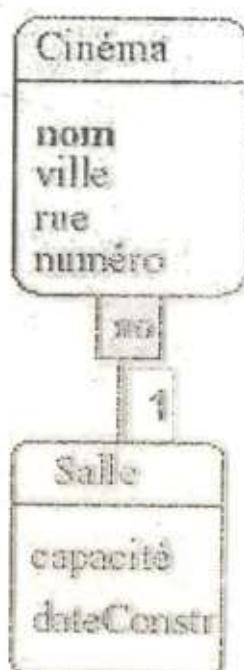
Dans UML, la qualification est utilisée pour représenter une association relative

Entité	Identifiant absolu
Cinéma	Nom cinéma
Salle	Id. salle



(a)

Entité	Identifiant absolu
Cinéma	Nom cinéma
Salle	Nom cinéma + N° Salle



(b)

#### Association hiérarchique standard

Salle = Entité forte

( identifiée de manière absolue  
avec l'attribut « Id » croissant )

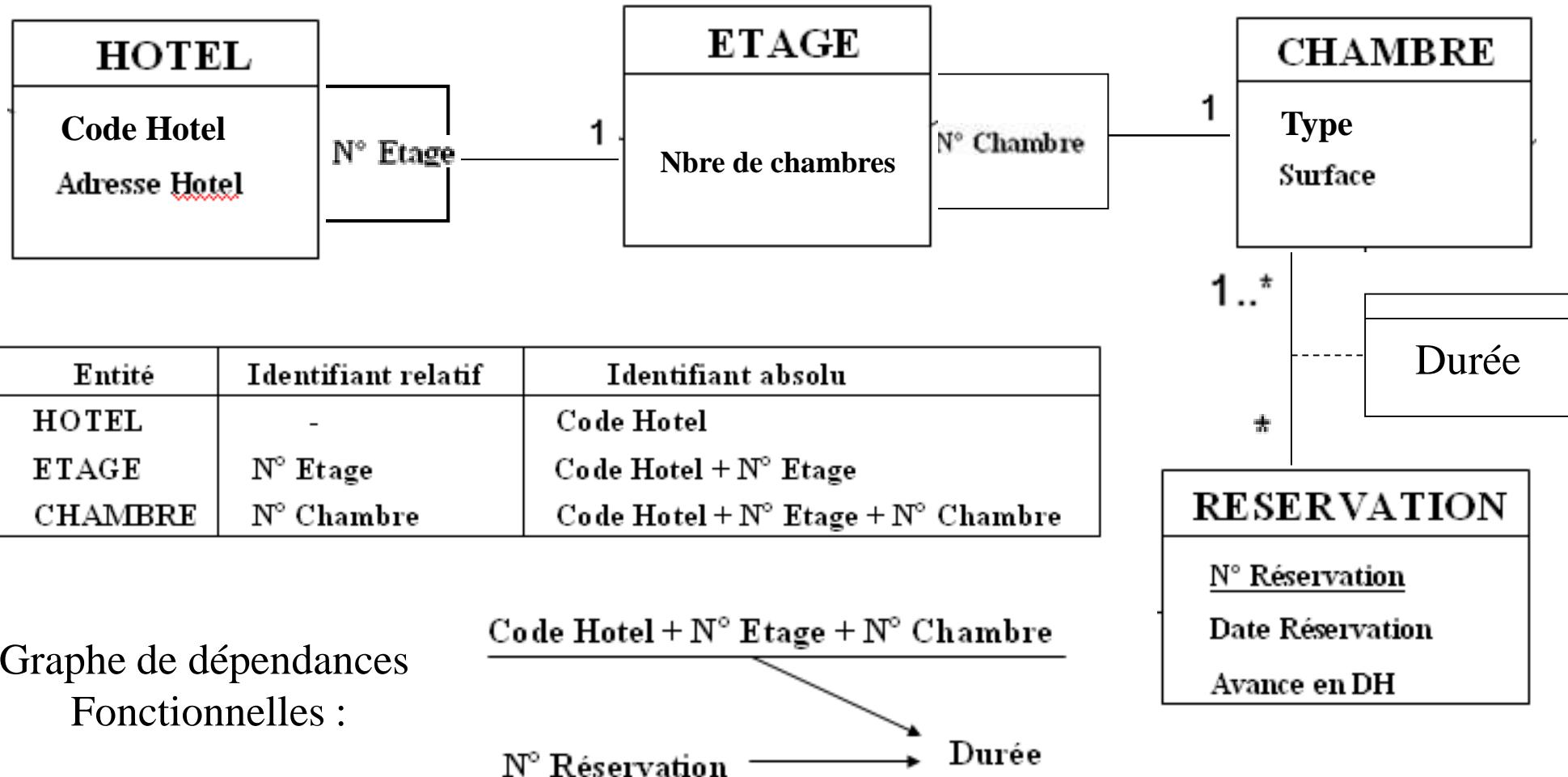
#### Association relative ( modélisée par une qualification )

Salle = Entité faible

( identifiée de manière relative par  
avec l'attribut « No » par rapport à un cinéma )

### **4.2.1 - La Phase d'analyse**

## **B7 – Type d’associations binaires : Associations relatives ( suite )**

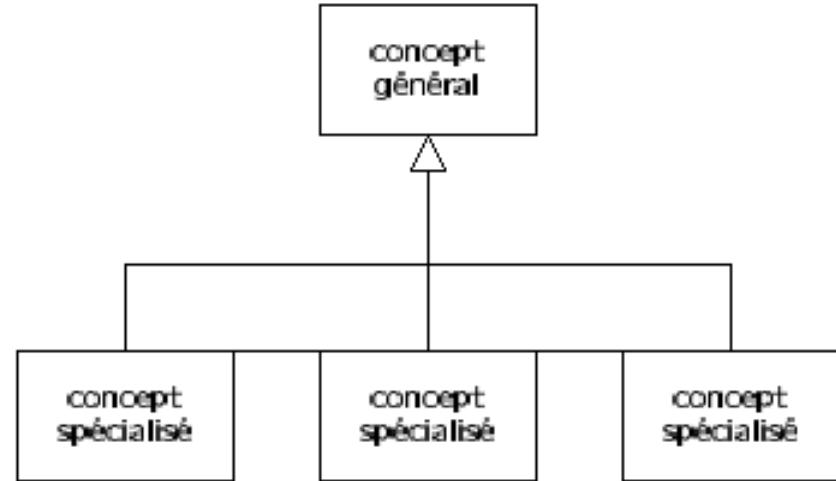
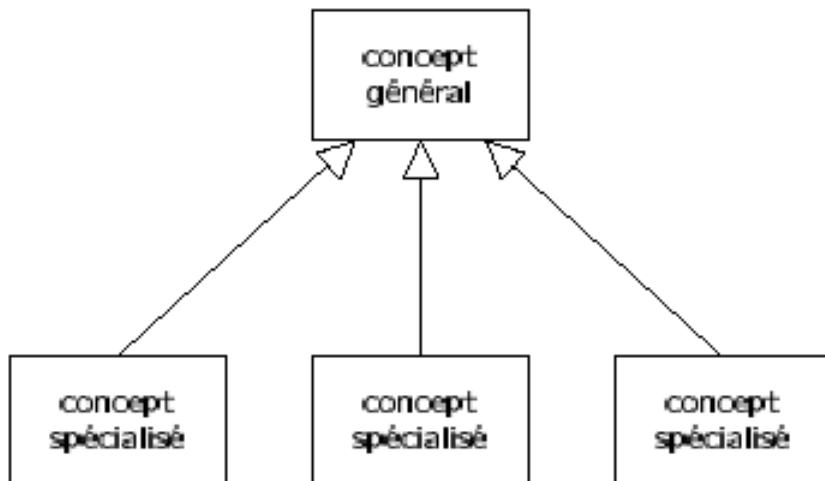


## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : La généralisation / spécialisation des concepts

Il arrive qu'une classe ( concept ) soit essentiellement identique à une autre , les seules différences résidant dans un certain nombre de caractéristiques supplémentaires ( des attributs , des méthodes ou des associations avec d'autres concepts ) . On dit alors que le 1er concept est une **spécialisation** du second et que le second est une **généralisation** du 1er .

- **Généralisation** = relation entre un élément plus général et un élément plus spécifique qui est entièrement conforme avec le premier élément, et qui ajoute de l'information supplémentaire
- **Spécialisation** = mécanisme par lequel des éléments plus spécifiques incorporent la structure et le comportement d'éléments plus généraux (notion **d'héritage**).



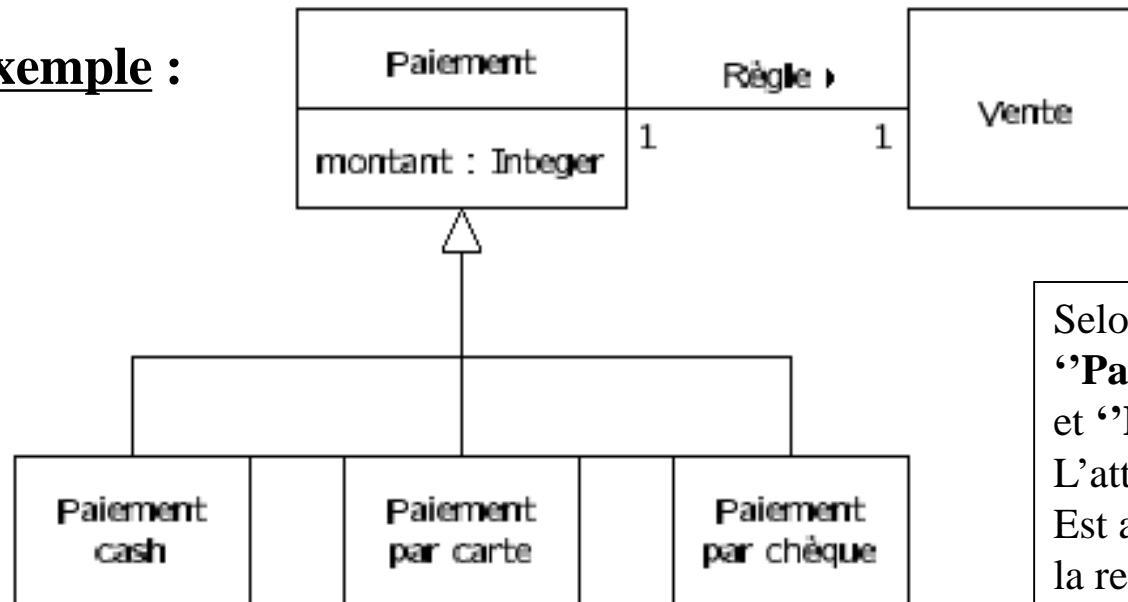
## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : l'héritage

Les spécialisations d'un concept héritent de toutes les caractéristiques de ce concept .

Cet héritage concerne les attributs , les méthodes ( opérations ) et les associations éventuelles avec d'autres concepts .

Exemple :



Selon ce modèle , les concepts “**Paiement cash**” , “**Paiement par carte**” et “**Paiement par chèque** ” possèdent L'attribut “**Montant**” , et chacun d'eux Est associé au concept “**Vente**” par la relation “**Règle**” .

Principe de substitution ( Liskov - 1987 ) :

Si 2 concepts sont liés par une relation de spécialisation , alors ils doivent satisfaire la règle suivante :

**“ Toute instance du concept spécialisé doit pouvoir jouer le rôle d'une instance du concept général ”**

Cette règle peut être vérifiée en formant la phrase : <concept spécialisé> est un <concept général> ?

Cas de notre exemple : Un paiement cash , par carte ou par chèque **est toujours** un paiement .

## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : la Classification multiple

Un concept général peut souvent être spécialisé de plusieurs façons différentes .

#### Exemple :

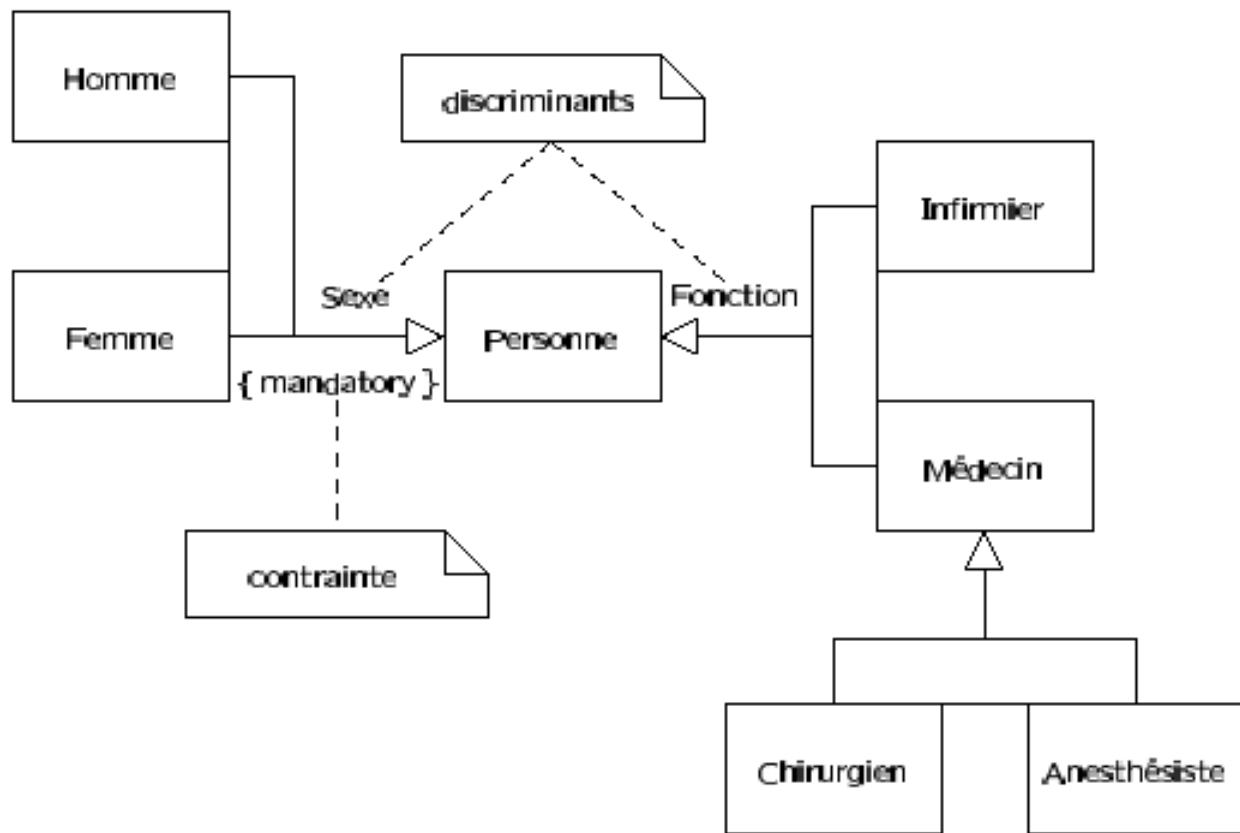
Dans un hôpital , chaque médecin ou infirmier est aussi un homme ou bien une femme .

Il est donc possible qu'une instance d'un concept appartienne simultanément à plusieurs spécialisations de ce concept .

Sur le diagramme de classe , on associe aux relations de généralisation des discriminants qui précisent quelles combinaisons de spécialisation sont autorisées .

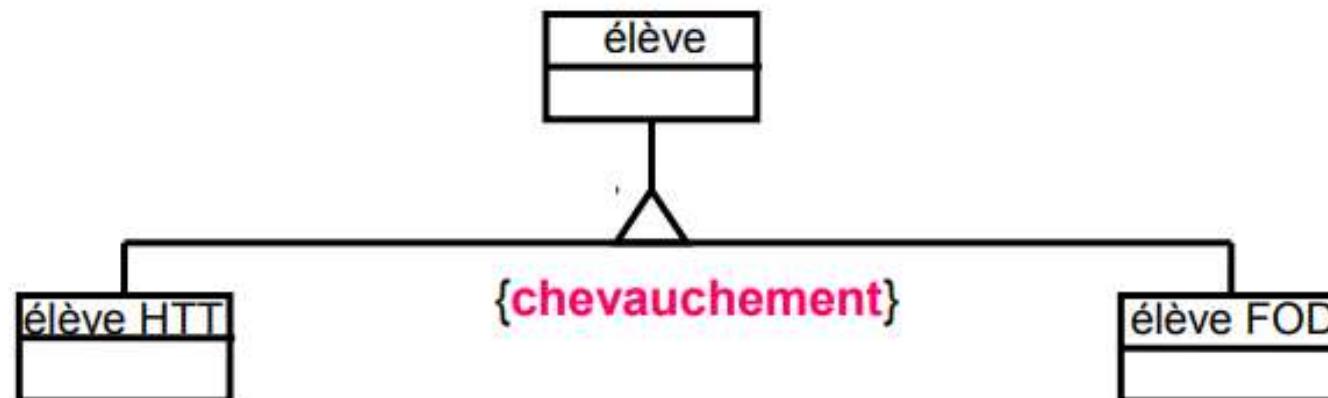
#### Règles :

- Une instance d'un concept général ne peut être simultanément une instance de 2 concepts spécialisés associés au même discriminant ( une personne peut être un infirmier ou un médecin ou ni l'un ni l'autre )
- Si la contrainte "**Mandatory**" accompagne un discriminant , alors toute instance du concept général doit obligatoirement être une instance d'un et d'un seul concept spécialisé associé à ce discriminant . ( une personne est **obligatoirement** soit un homme , soit une femme ) .



## Héritage avec recouvrement

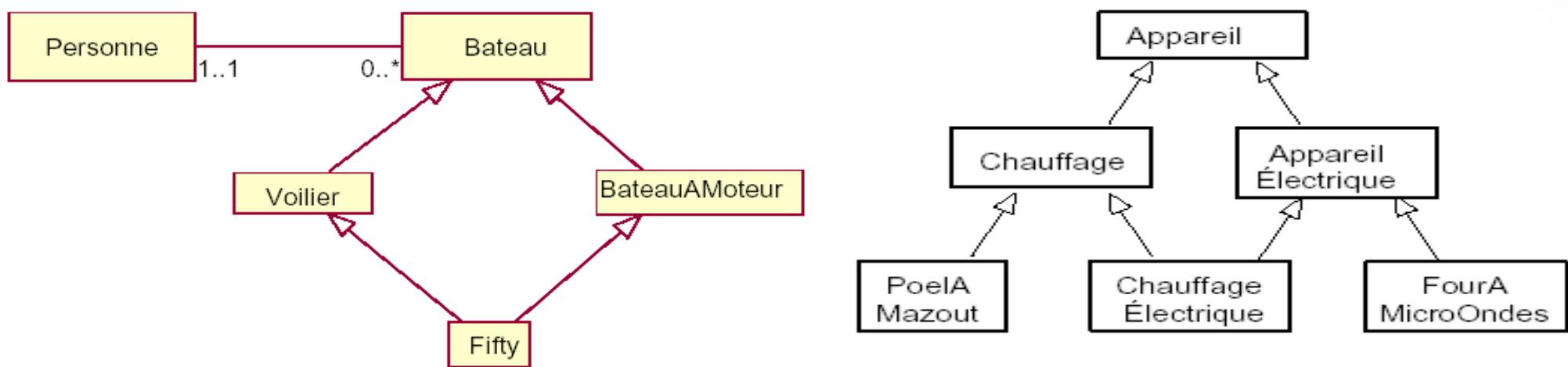
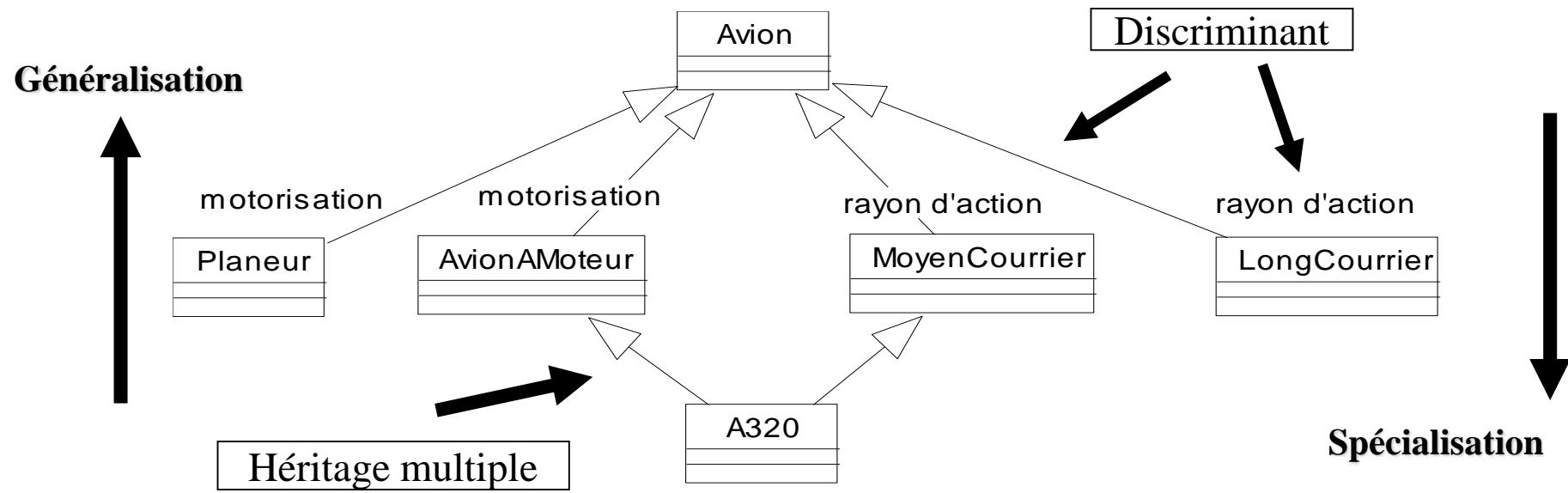
- Chevauchement : deux sous classes avec des instances identiques
- Disjoint : les instances d'une sous classe ne peuvent être dans une autre sous classe
- Complète : la généralisation ne peut être étendue (**Mandatory**)
- Incomplète : la généralisation peut être étendue



## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : l'héritage multiple

Un même concept spécialisé peut être associé à plusieurs concepts généraux et hériter du comportement de chacun d'eux .



# Agrégation ou association ?

# Agrégation ou généralisation ?

- Agrégation ou association :

si 2 objets sont étroitement associés par une relation de type «composé-composant», c'est une **agrégation**

si 2 objets sont habituellement indépendants, bien que parfois associés, c'est une **association**.

- Agrégation ou généralisation :

Dites vous «ou»  
ou dites-vous «et» ?

Exemples :

Un moteur est diesel ou à essence  
**(généralisation)**

Un moteur est composé de cylindres et de pistons  
**(agrégation)**

Dans les assertions suivantes, préciser s'il s'agit d'une association, d'une agrégation ou d'une généralisation :

- Une pièce de théâtre classique comporte des actes et des scènes.
- Chez les artistes de music-hall, il y a des compositeurs, des interprètes et des paroliers.
- Un coureur automobile participe à des courses pour une écurie
- Un programmeur écrit des programmes.
- Un programme COBOL est composé de divisions, de sections et de paragraphes.
- Les modems, les écrans, les souris et les claviers sont des organes d'entrée-sortie.
- Un ordinateur comprend une caisse, une alimentation, des composants électroniques, un disque dur, des organes d'entrée-sortie.

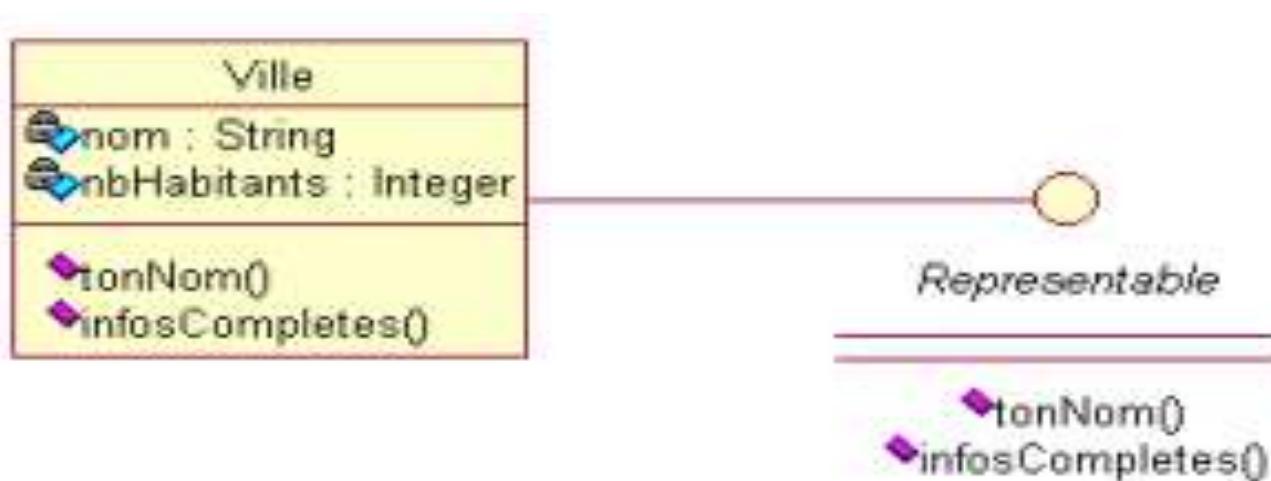
## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : les Interfaces

Une interface décrit de manière abstraite le comportement d'une classe , d'un composant , d'un sous-système , d'un paquetage ou de toute autre classificateur .

Exemple :

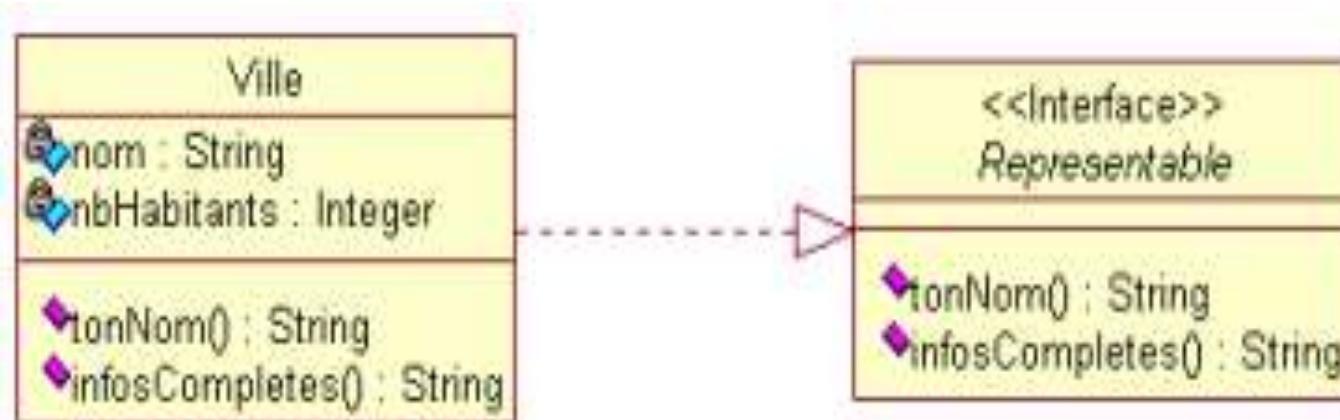
1ère Notation



Classe avec implémentation  
des méthodes

Classe générique sans implémentation  
des méthodes

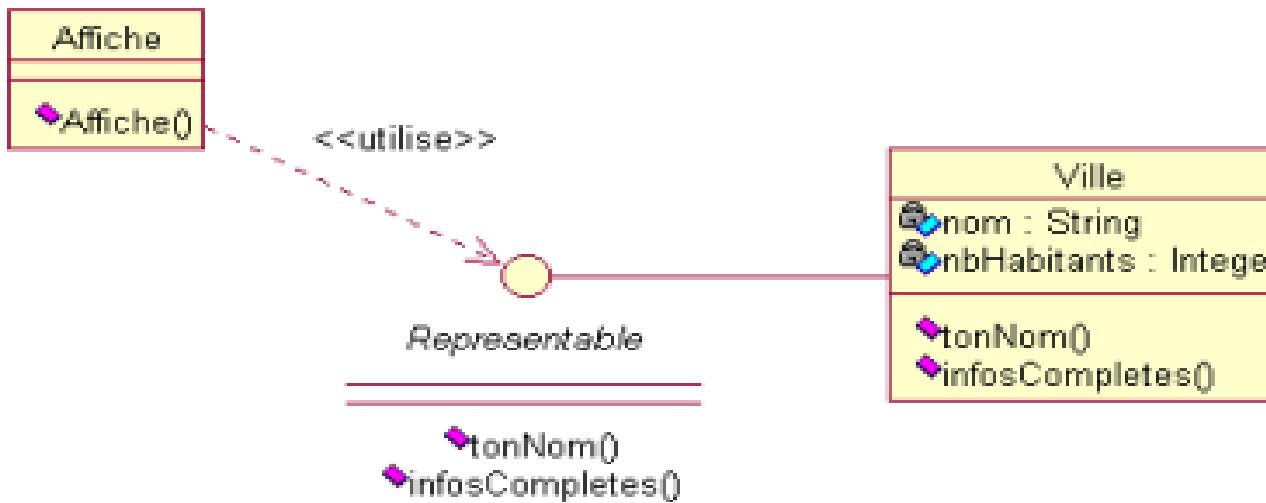
2ème Notation



## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : les Interfaces ( suite )

*Une interface possède uniquement la spécification d'un comportement visible, sous forme d'un ensemble d'opérations (pas d'attributs et d'associations), et ne fournit aucune implémentation de ses services.*

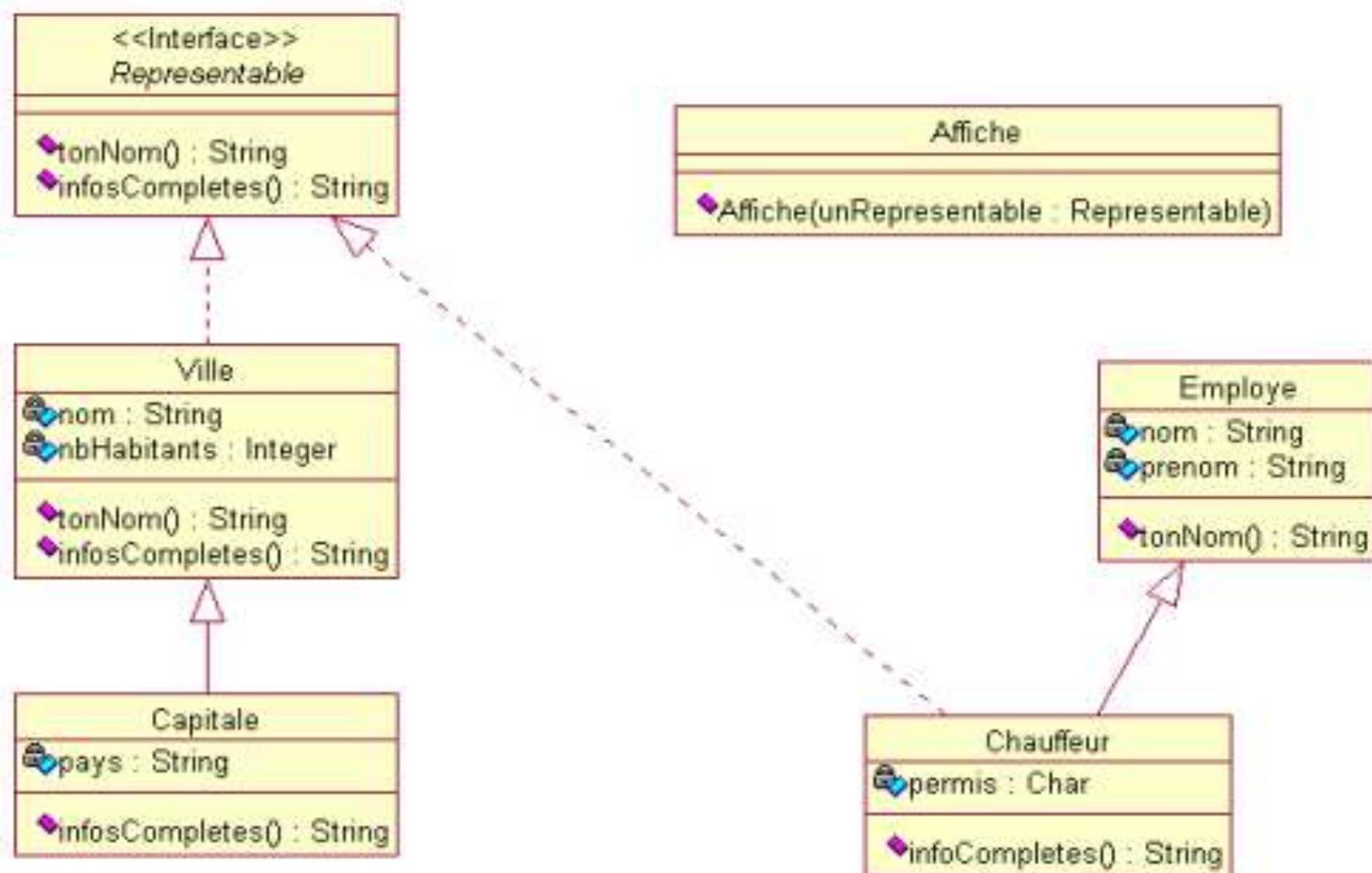


La classe VILLE implémente l'interface REPRESENTABLE ( elle comporte le code associé aux méthodes qui sont décrites dans l'interface ).

La classe AFFICHE utilise l'interface REPRESENTABLE ( déclenche l'exécution de ses méthodes ) pour une instance de la classe VILLE ou de toute autre classe implémentant l'interface .

## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : les Interfaces ( suite )



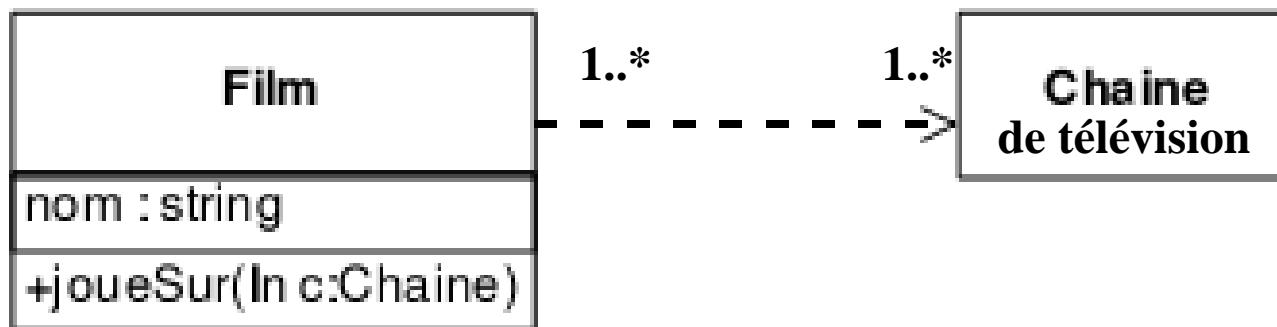
## 4.2.1 - La Phase d'analyse

### B12 – Associations spéciales

#### Association de dépendance ( ou d'utilisation ) :

= Relation sémantique entre 2 éléments selon laquelle un changement apporté à l'un peut affecter la sémantique de l'autre

Ce genre d'association apparaît dans un schéma dans le cas où le comportement dynamique de la 1<sup>ère</sup> classe nécessite la présence de la 2<sup>ème</sup> classe ( exemple ci-dessous )



# Modélisation des classes et des états

## Diagrammes de classe : exercice d'application 1

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

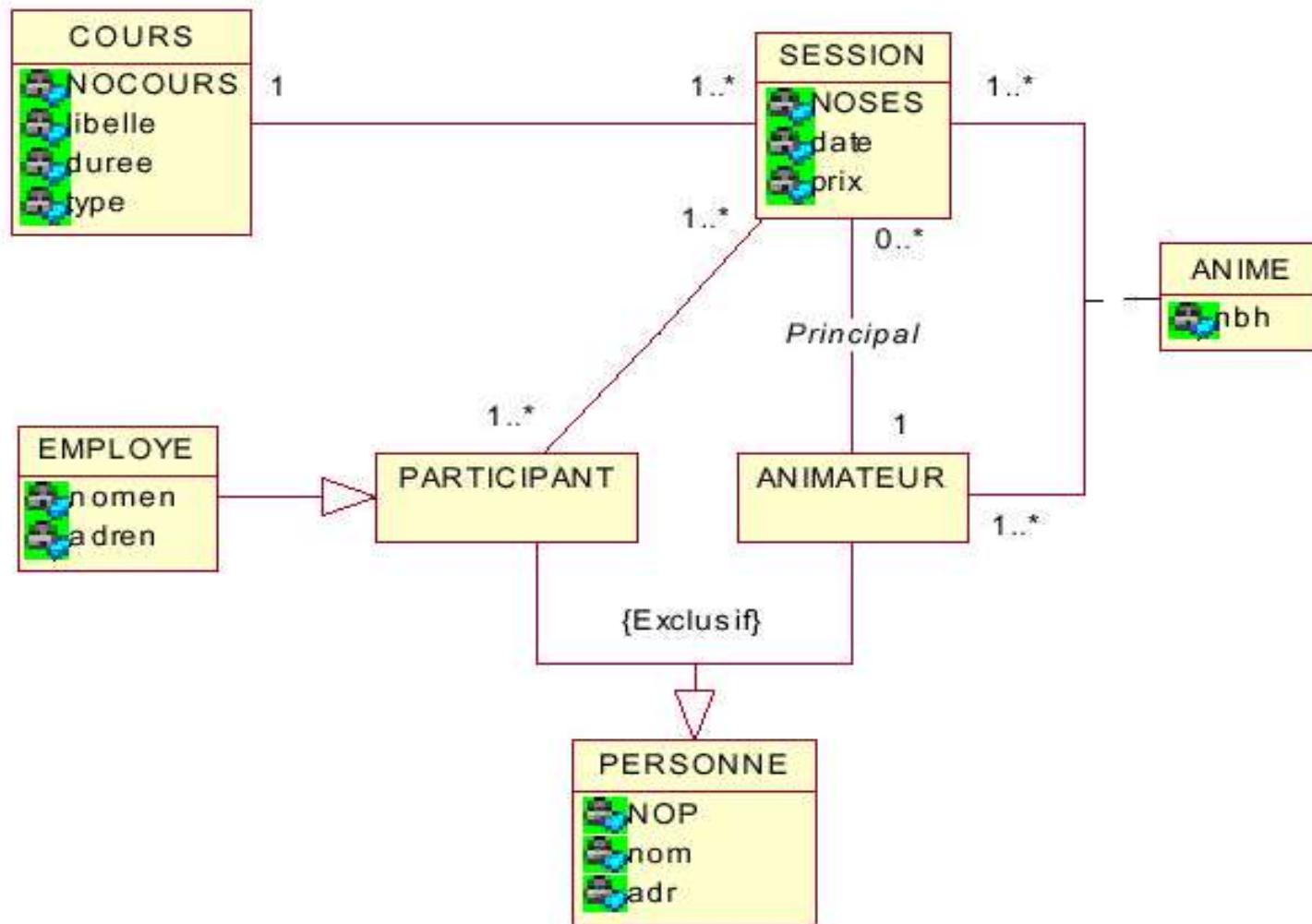
Il s'agit d'établir le schéma des données pour la gestion des formations d'un institut privé. Un cours est caractérisé par un numéro de cours (NOCOURS), un libellé (LIBELLE), une durée en heures (DUREE) et un type (TYPE). Un cours peut faire l'objet dans l'année de plusieurs sessions identiques. Une 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.

# Modélisation des classes et des états

## Diagrammes de classe : exercice d'application 1



# Modélisation des classes et des états

## Diagrammes de classe : exercice d'application 2

### **Exercice 2 : 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, 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.

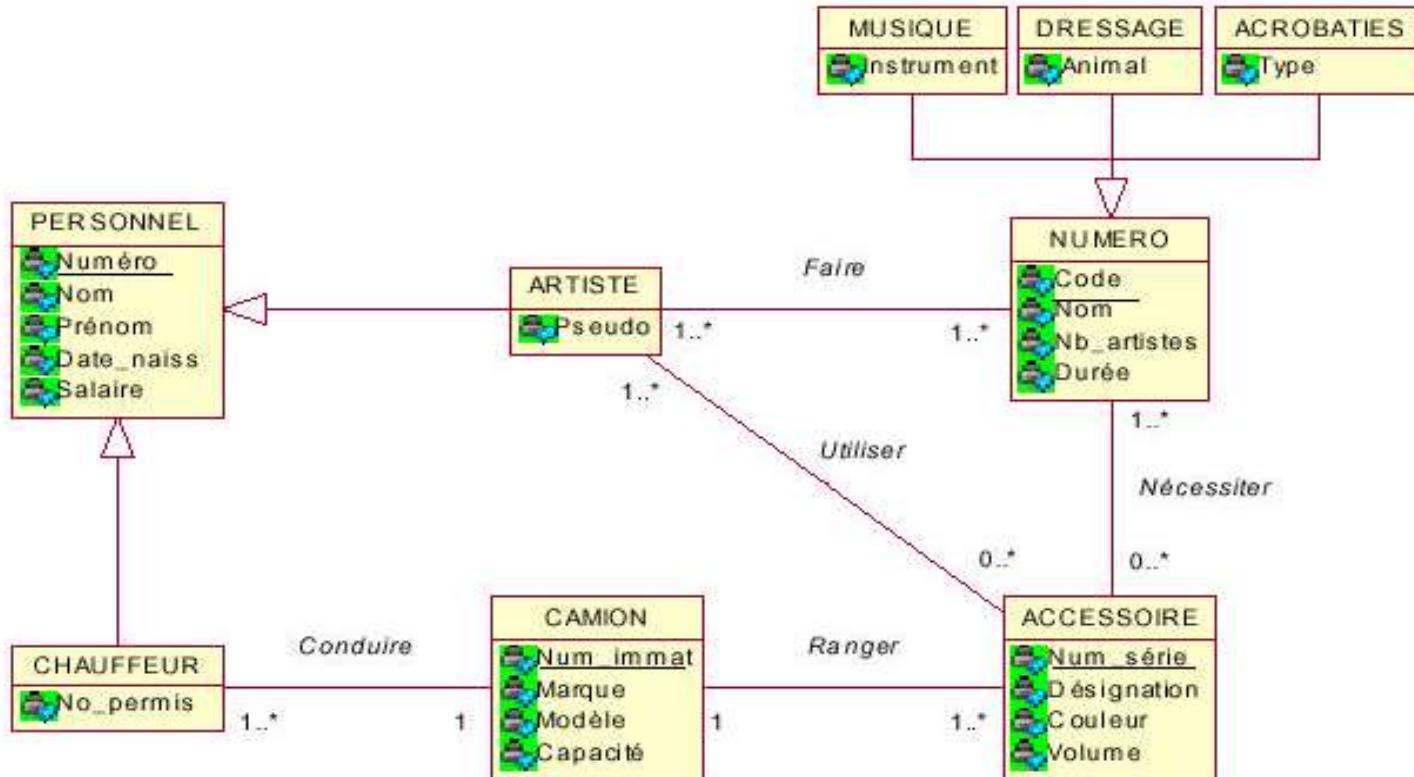
Les artistes sont susceptibles d'assurer plusieurs numéros, chaque numéro étant caractérisé par un code, son nom, le nombre d'artistes présents sur scène et sa durée. De plus, on souhaite savoir l'instrument utilisé pour les numéros musicaux, l'animal concerné par les numéros de dressage et le type des acrobaties (contorsionnisme, équilibriste, 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).

# Modélisation des classes et des états

## Diagrammes de classe : exercice d'application 2



## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations

- **Contrainte** = Relation sémantique entre éléments de modèle qui spécifie des conditions ou propositions devant être respectées **pour que le modèle soit valide**

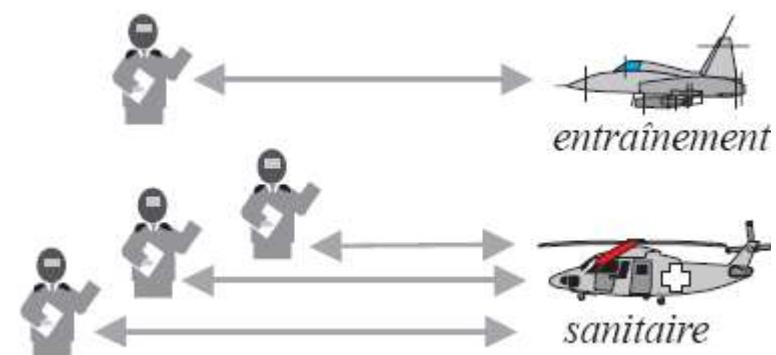
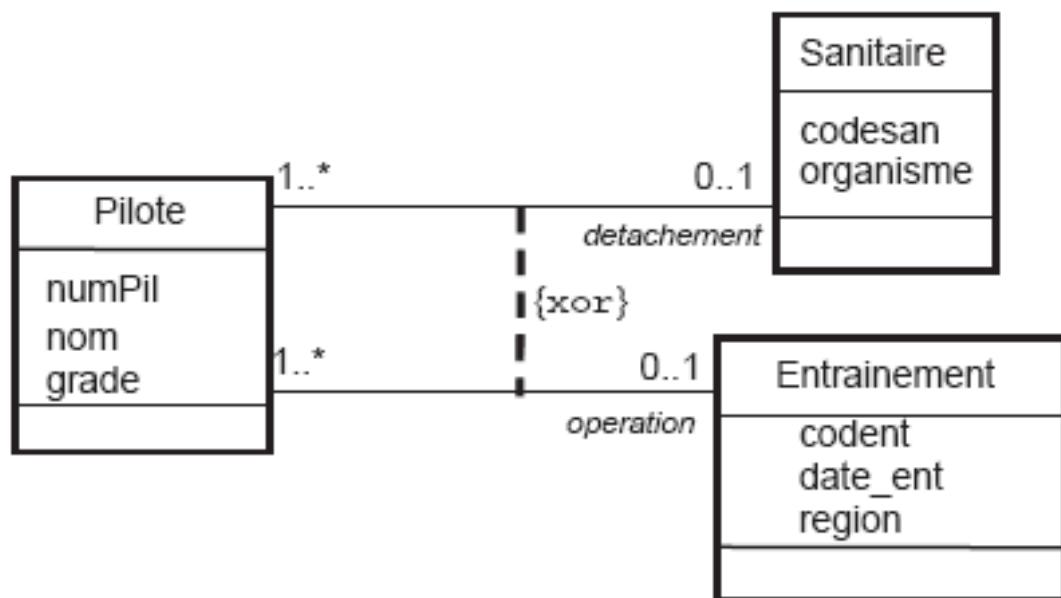
Notation : { contrainte }

*Contrainte de partition* : { xor } ou {partition}

---

Selon la contrainte de *partition*, toutes les occurrences d'une entité (ou classe) participent à l'une des deux associations, mais pas aux deux, ni à aucune des deux.

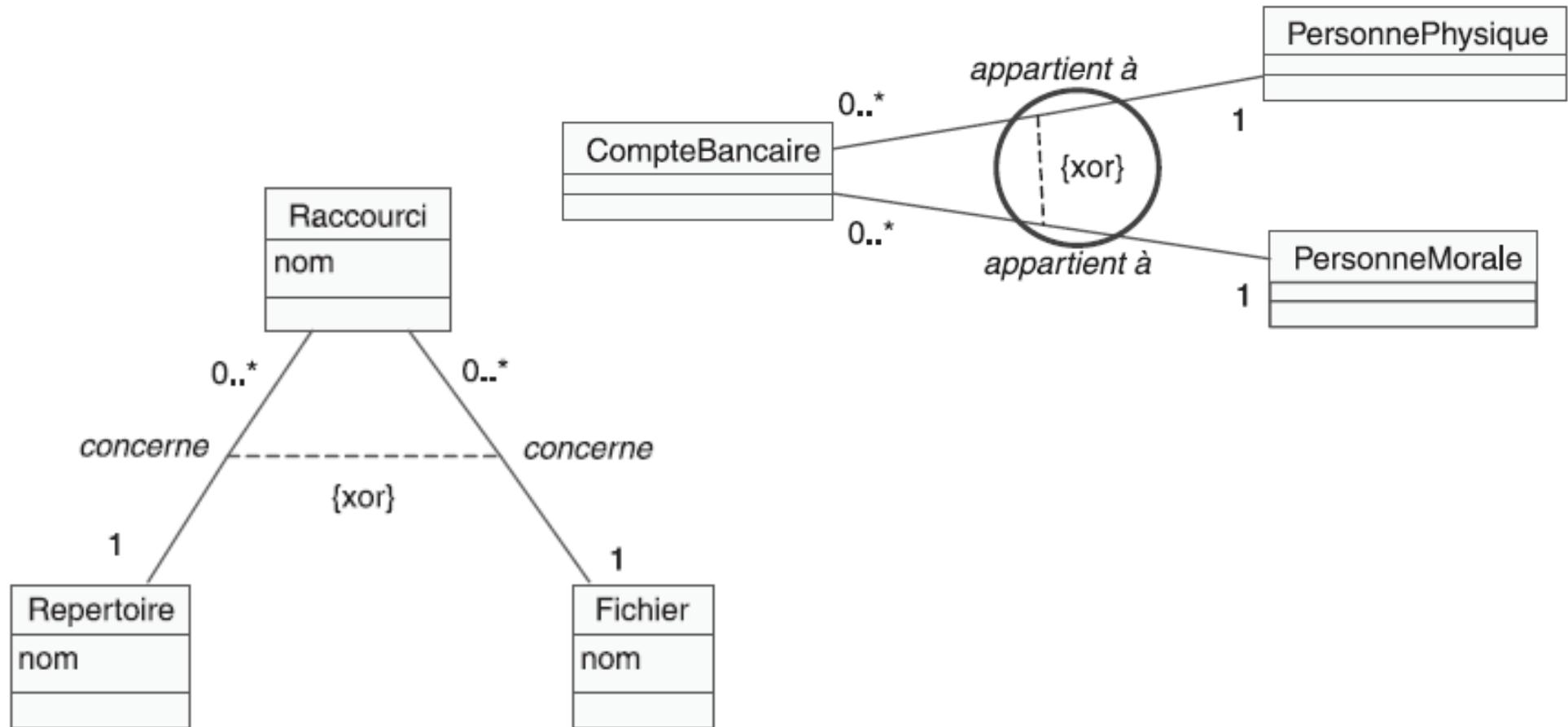
---



## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 1 )

#### Contrainte de partition ( Autres exemples )



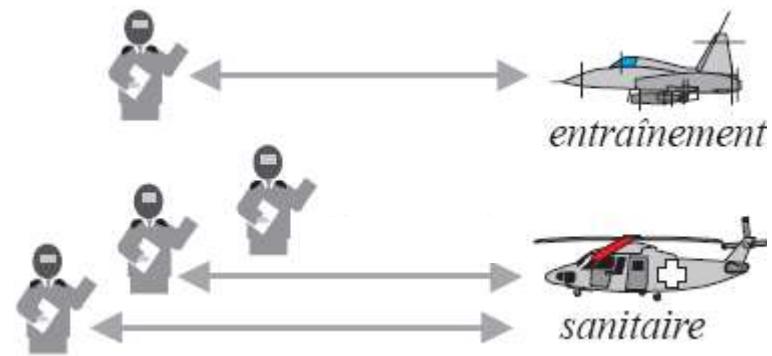
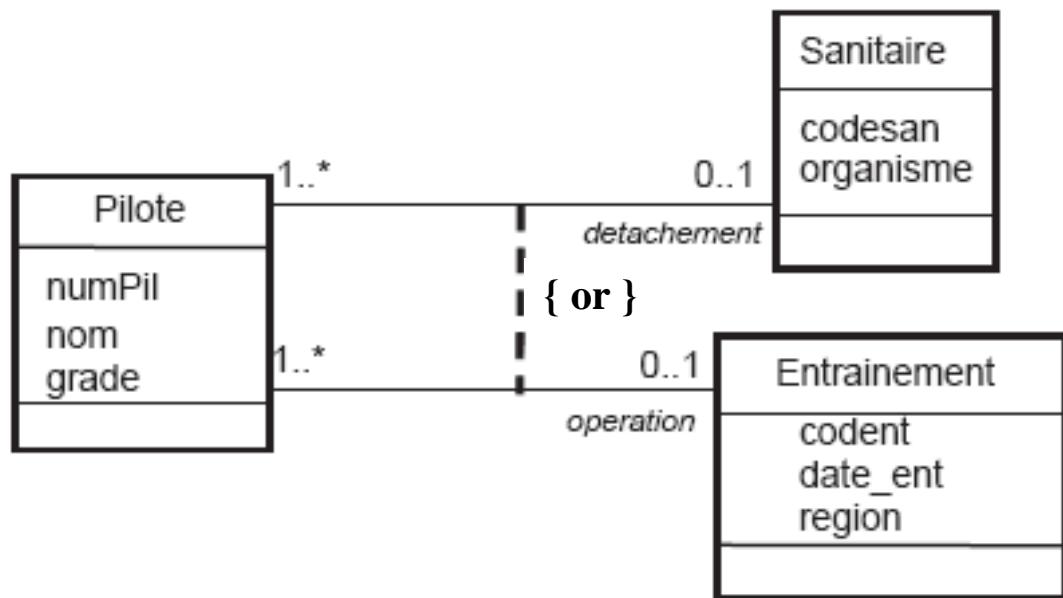
## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 2 )

#### *Contrainte d'exclusivité : { or } ou { exclusivité }*

Selon la contrainte *d'exclusivité*, toutes les occurrences d'une entité (ou classe) peuvent participer à l'une des deux associations, mais pas aux deux à la fois.

Un pilote peut être au repos (il n'est affecté à aucune mission). Si un pilote est affecté à un exercice d'entraînement, alors il ne peut pas être affecté à une mission sanitaire et réciproquement.



## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 3 )

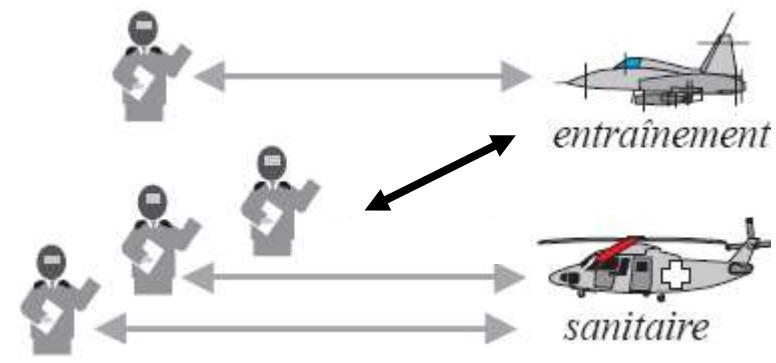
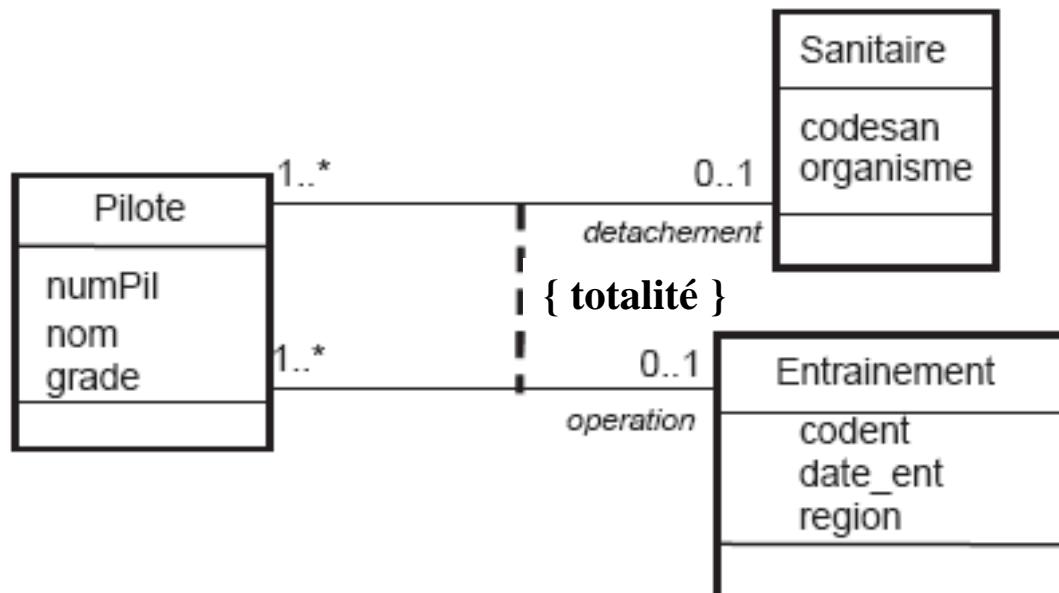
#### *Contrainte de totalité : { totalité }*

Selon la contrainte de *totalité*, toutes les occurrences d'une entité (ou classe) participent au moins à une association.

Considérons :

- qu'un pilote peut être affecté simultanément à une mission sanitaire et à un exercice d'entraînement ;
- que tous les pilotes participent au moins à une mission.

On peut dire ainsi que les pilotes forment la totalité du contingent.

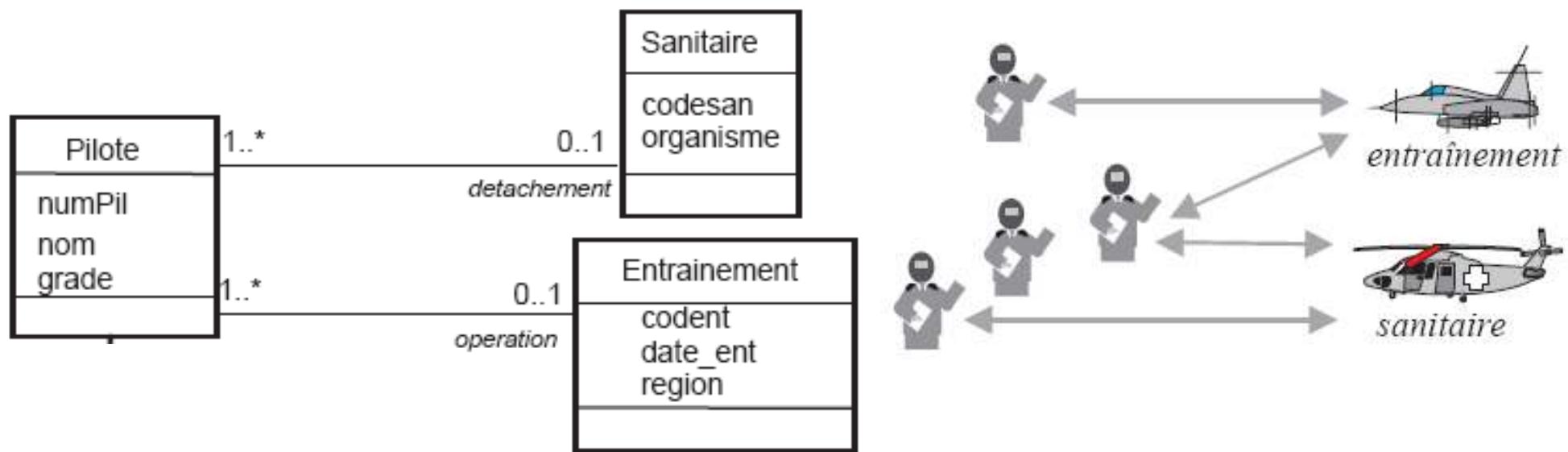


## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 4 )

#### *Absence de contrainte*

Dans notre exemple, l'absence de contrainte (illustrée à l'exemple      ) signifie qu'il peut exister des pilotes n'étant affectés à aucune mission ou participant simultanément à une mission sanitaire et à un exercice d'entraînement.



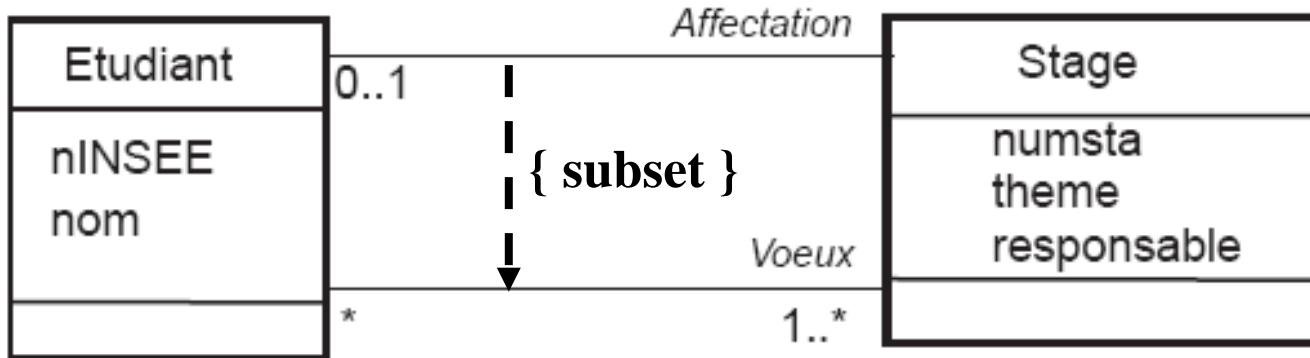
## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 5 )

#### *Contrainte d'inclusion : { inclusion } ou { subset }*

Selon la contrainte *d'inclusion*, toutes les occurrences d'une association doivent être incluses dans les occurrences d'une autre association.

Des étudiants émettent des vœux concernant des stages sachant qu'ils doivent suivre un seul stage. Un stage peut intéresser aucun ou plusieurs étudiants. Pour modéliser le fait que le stage effectué par un étudiant doit être un stage figurant dans ses vœux, il faut ajouter une contrainte d'inclusion.



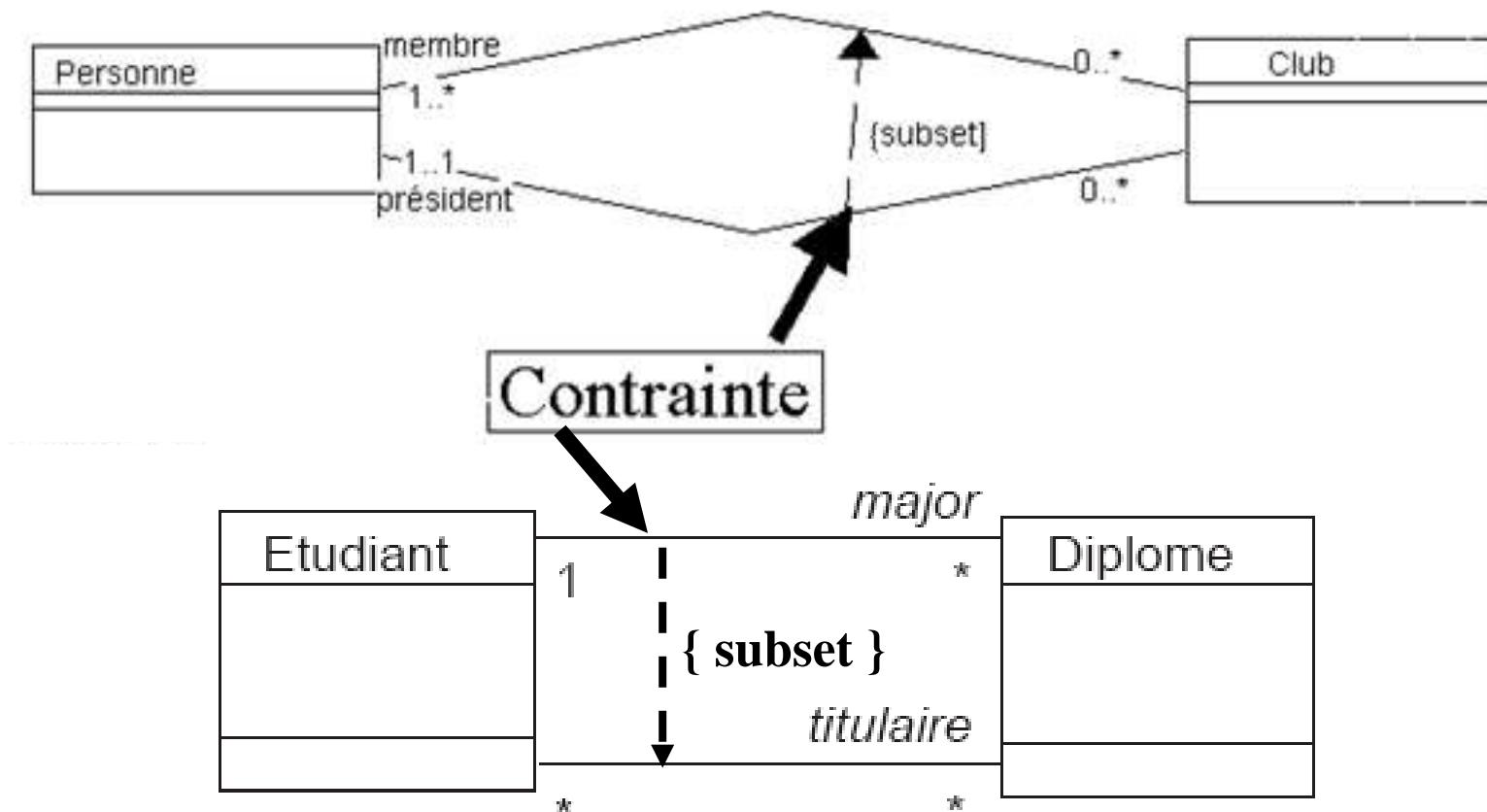
**Les instances de l'association « Affectation » sont incluses dans l'ensemble des instances de l'association « Vœux »**

## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 6 )

*Contrainte d'inclusion : { inclusion } ou { subset }*

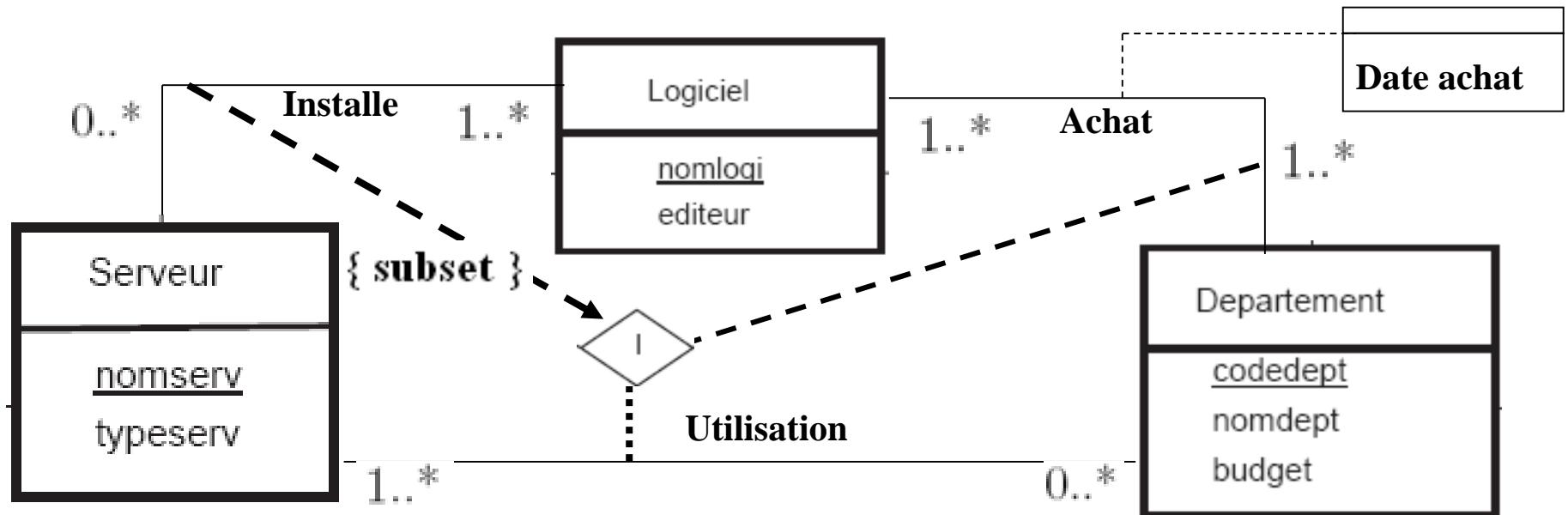
Autres exemples :



## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 7 )

*Contrainte d'inclusion : { inclusion } ou { subset }* entre 3 associations binaires



Représentation tabulaire des occurrences

Achat	
Dept.	Logiciel
D1	L1
D1	L2
D1	L4
D2	L1

Utilisation	
Dept.	Serveur
D1	S1
D1	S2
D2	S3

Jointure entre **Achat** et **Utilisation**

Logiciel	Serveur
L1	S1
L1	S2
L2	S1
L2	S2
L4	S1
L4	S2
L1	S3

Occurrences possibles pour l'association « **Installe** »

Installe	
Logiciel	Serveur
L1	S1, S2 ou S3
L4	S1 ou S2

## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 8 )

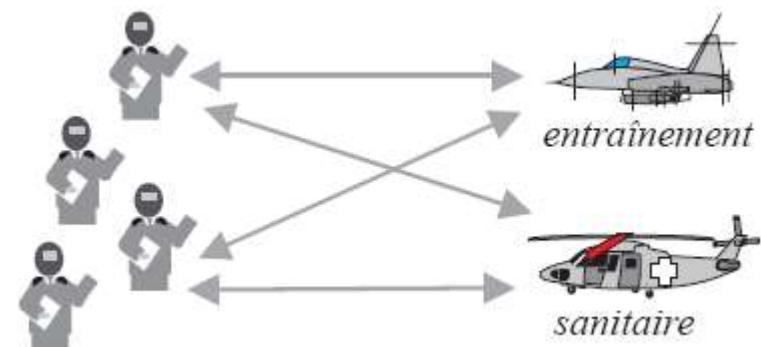
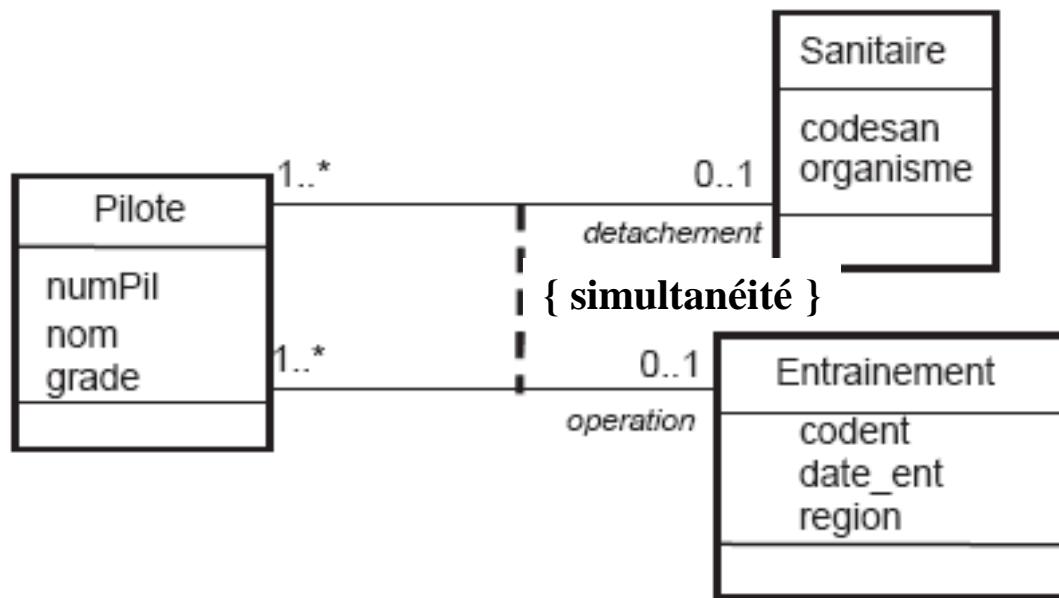
#### *Contrainte de simultanéité* : { simultanéité }

---

Selon la contrainte de *simultanéité* entre plusieurs associations, toute occurrence d'une entité (ou classe) liée à une association participe également aux autres.

---

Comme l'illustre l'exemple      un pilote peut être au repos, ou, s'il est affecté à un exercice d'entraînement, il doit aussi être affecté à une mission sanitaire et réciproquement.

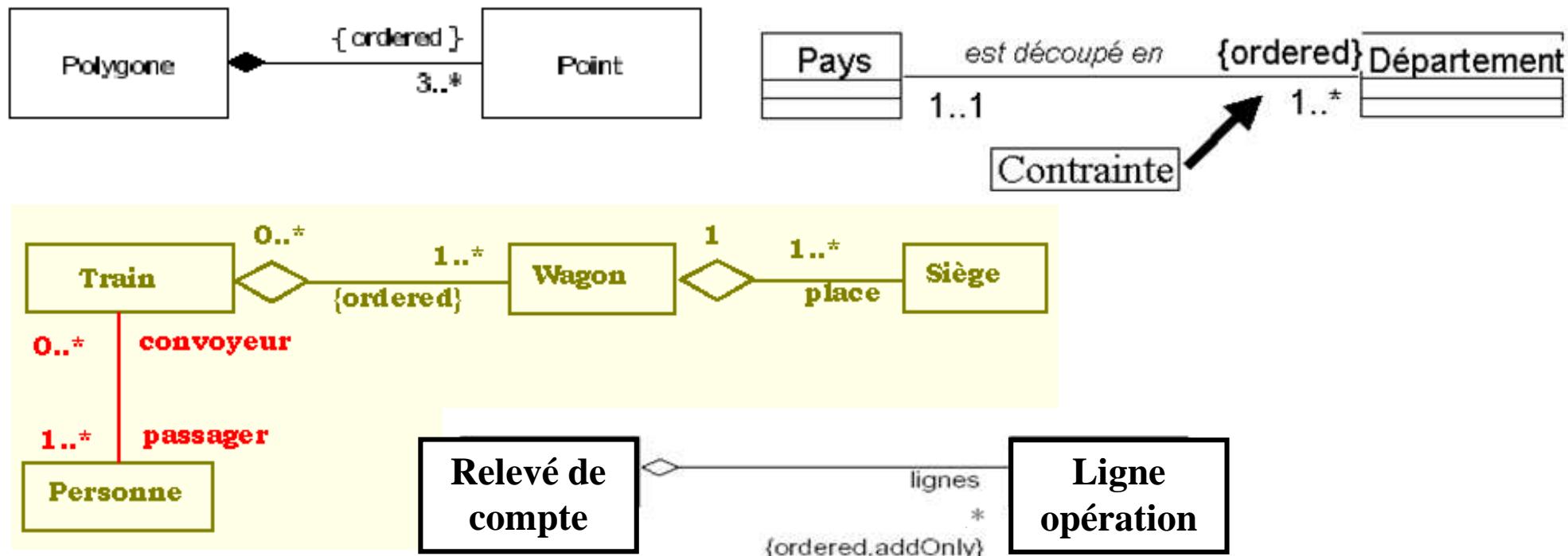


## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 9 )

Contraintes prédéfinies sur les associations.

- { frozen } : fixé lors de la création de l'objet, ne peut pas changer
- { ordered } : les éléments de la collection sont ordonnés
- { addOnly } : impossible de supprimer un élément

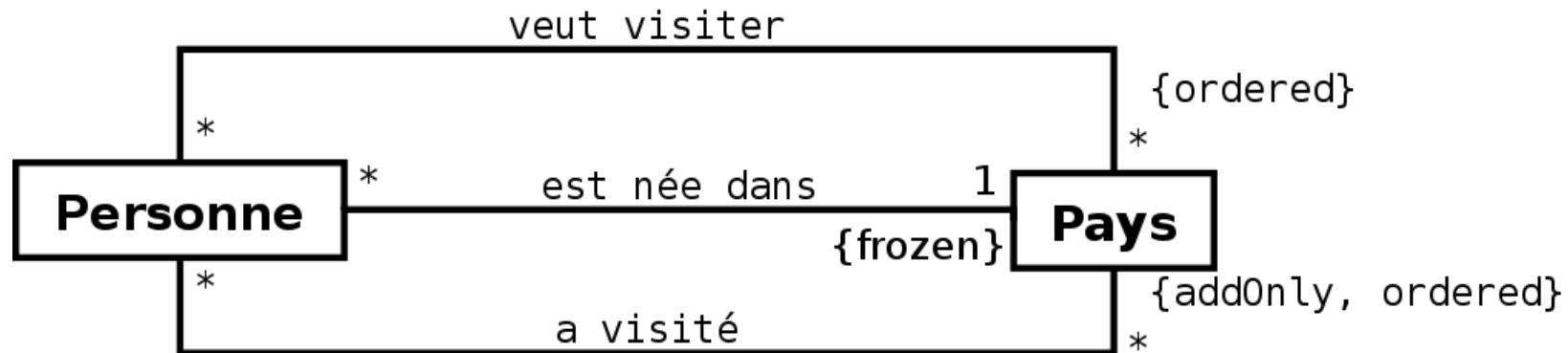


## 4.2.1 - La Phase d'analyse

### B13 – Les Contraintes sur associations ( suite 10 )

Contraintes prédéfinies sur les associations.

---



Ce diagramme exprime que :

- une personne est née dans un pays, et que cette association ne peut être modifiée
- une personne a visité un certain nombre de pays, dans un ordre donné, et que le nombre de pays visités ne peut que croître ;
- une personne aimeraient encore visiter toute une liste de pays, et que cette liste est ordonnée (probablement par ordre de préférence).

## 4.2.1 - La Phase d'analyse

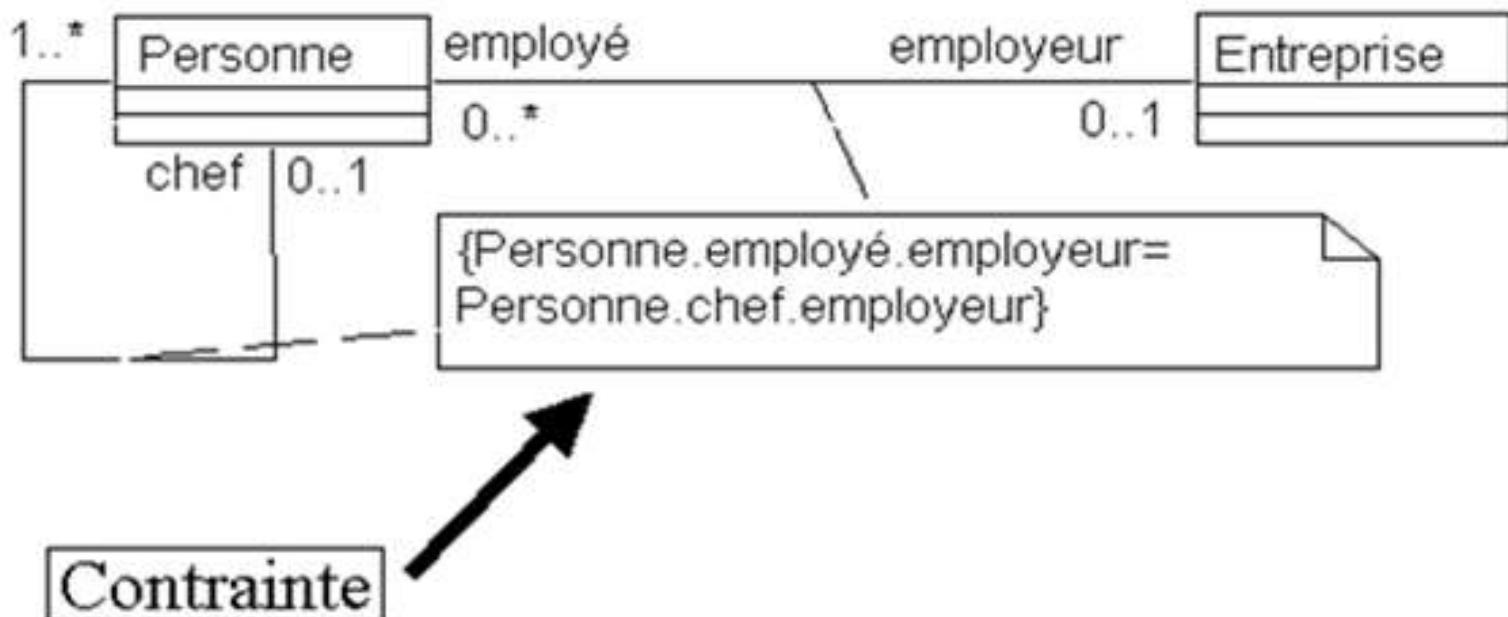
### B13 – Les Contraintes sur associations ( suite 11 )

#### Contraintes personnalisées sur les associations

Elles sont exprimées en OCL : Object Constraint Language .

OCL est un langage typé qui peut s'appliquer à la plupart des diagrammes UML et permet de spécifier des contraintes sur l'état d'un objet ou d'un ensemble d'objets

Exemple :



## 4.2 - La Phase d'élaboration

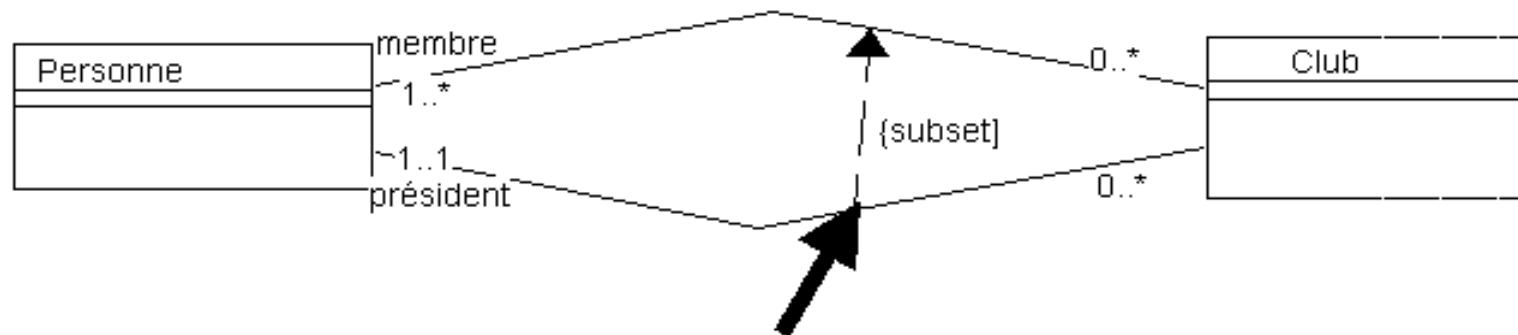
### B – Le Diagramme de Classe : les Contraintes ( Suite )

- **Contrainte** = Relation sémantique entre éléments de modèle qui spécifie des conditions ou propositions devant être respectées **pour que le modèle soit valide**

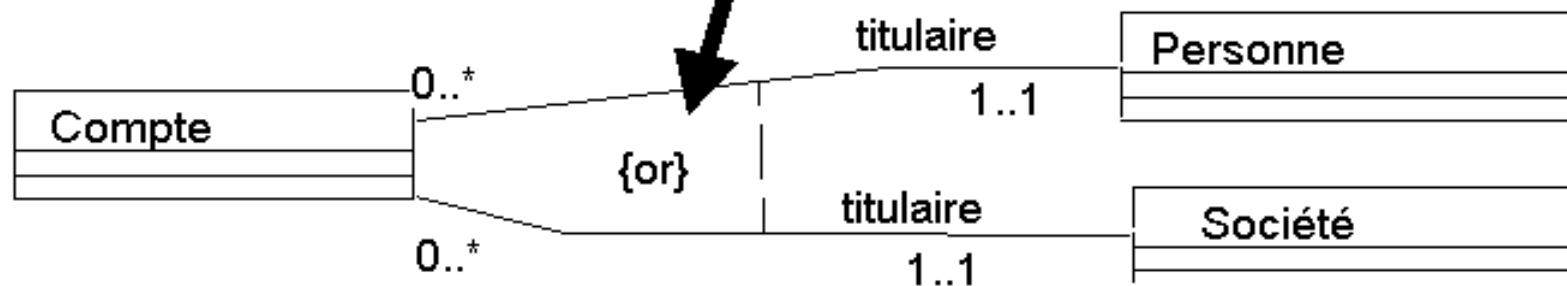
Notation : { contrainte }

- Contraintes Pré-Définies

- **Inclusion**



- **Exclusion**

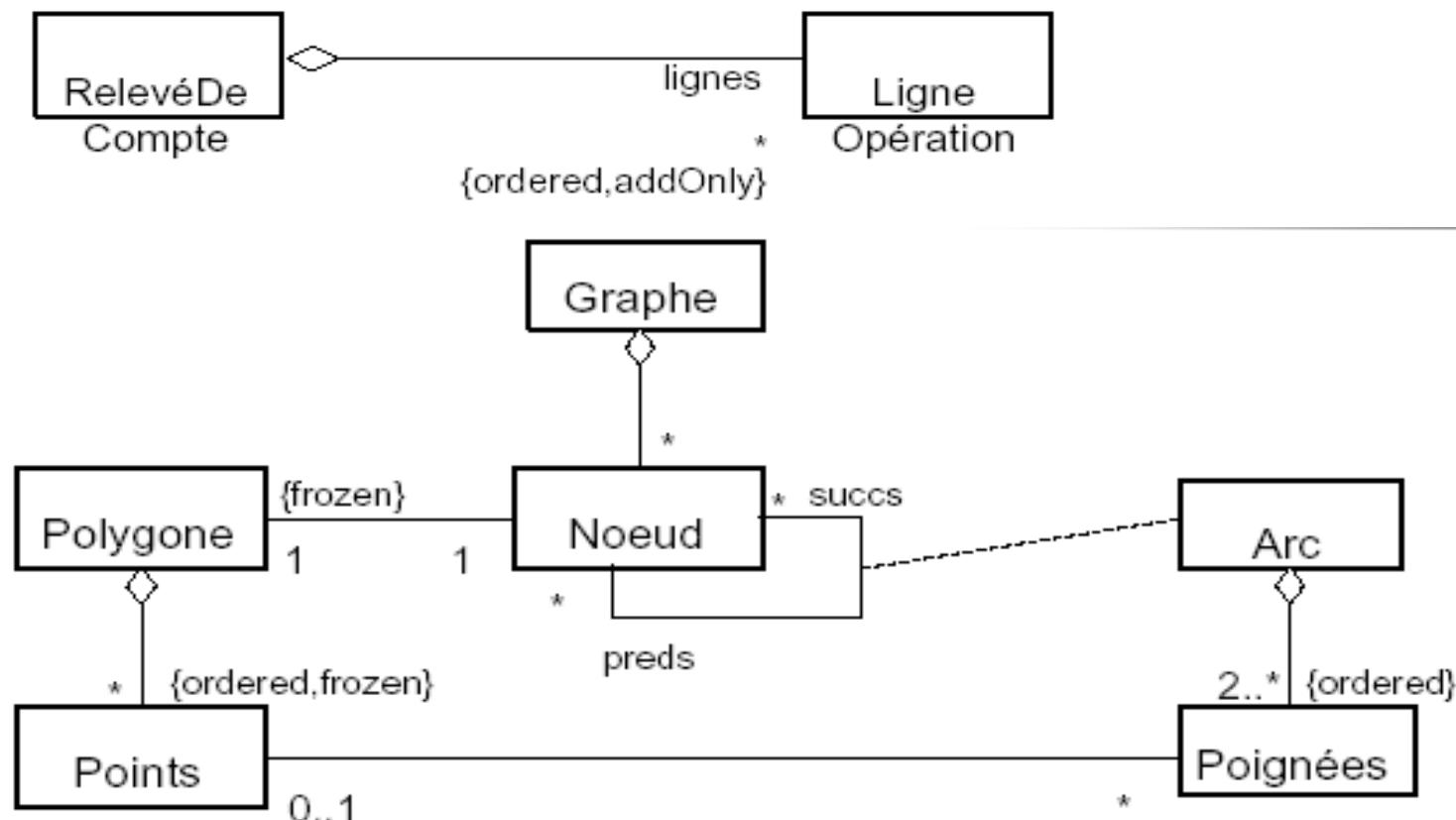


## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : les Contraintes ( Suite )

Contraintes prédéfinies sur les associations. Par exemple :

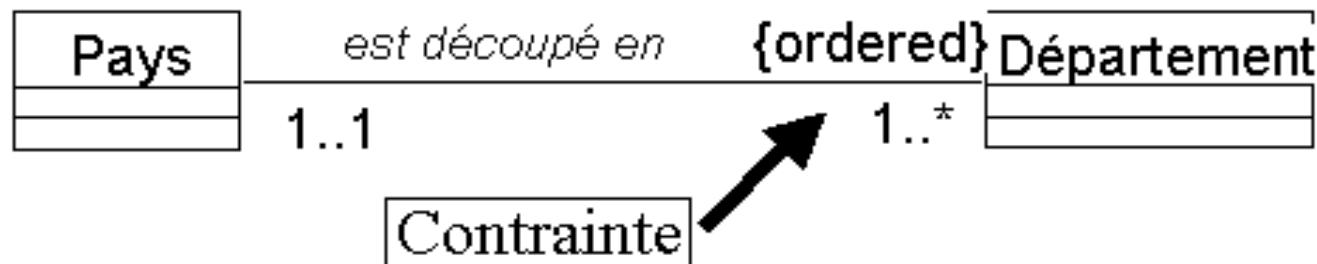
- { frozen } : fixé lors de la création de l ’objet, ne peut pas changer
- { ordered } : les éléments de la collection sont ordonnés
- { addOnly } : impossible de supprimer un élément



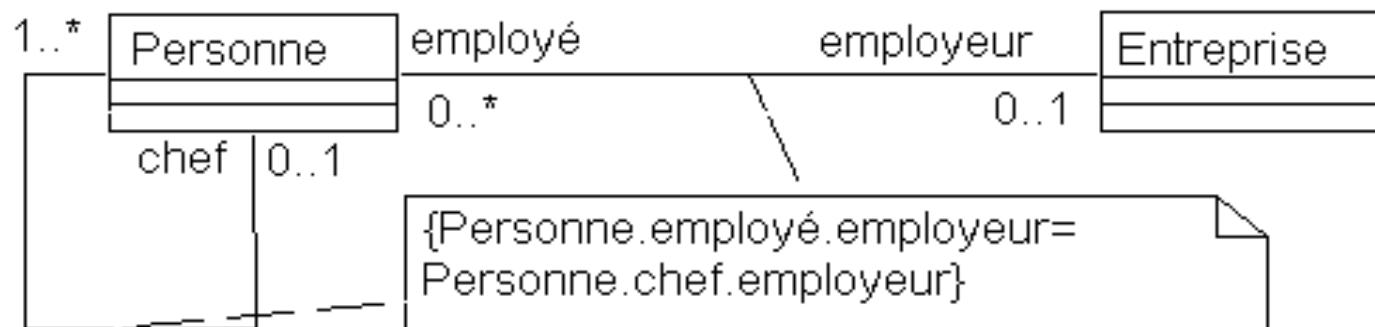
## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : les Contraintes ( Suite )

- **Ordre**



- **Chemins**

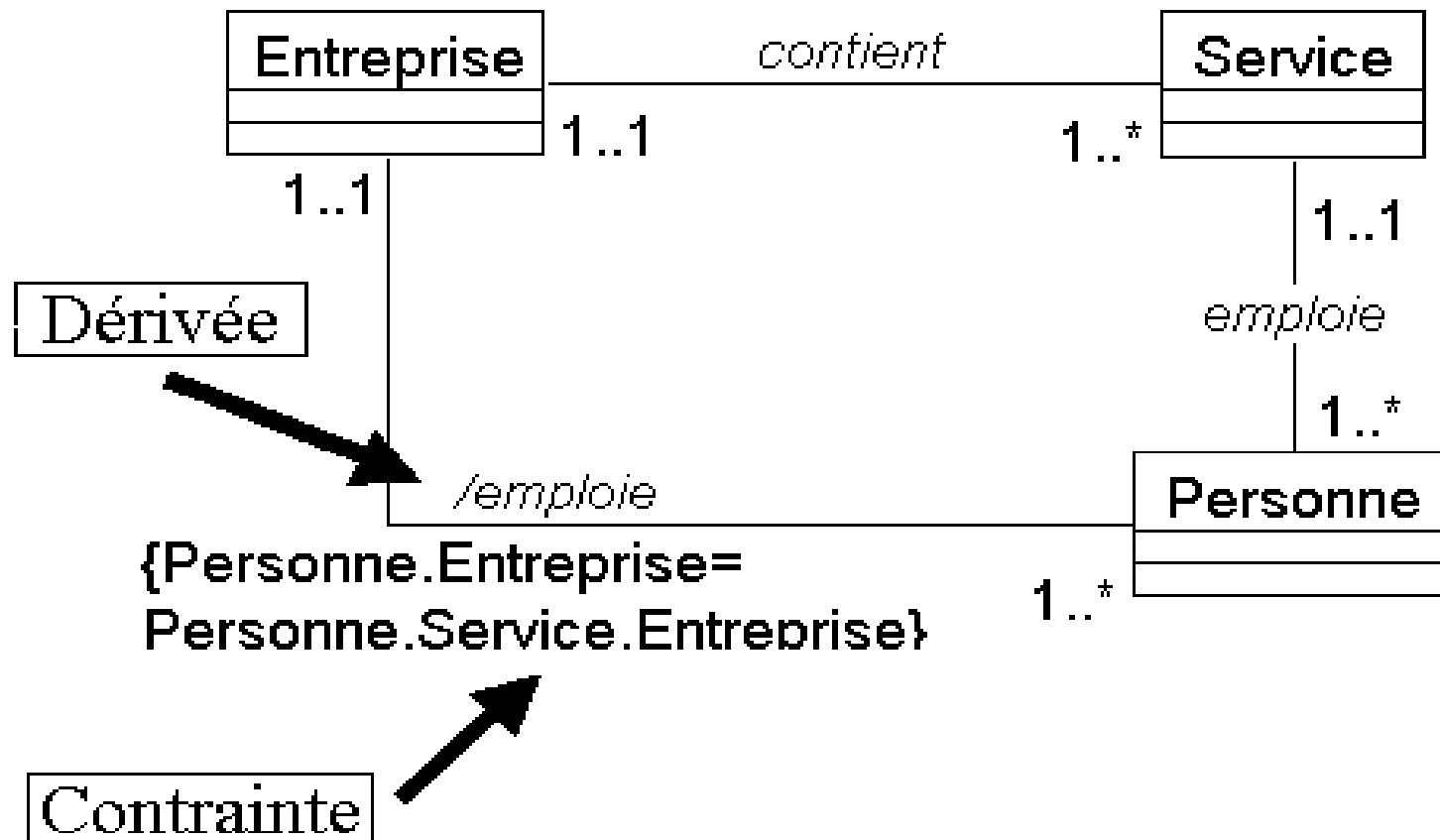


**Contrainte**

## 4.2 - La Phase d'élaboration

### B – Le Diagramme de Classe : les Associations dérivées ( Suite )

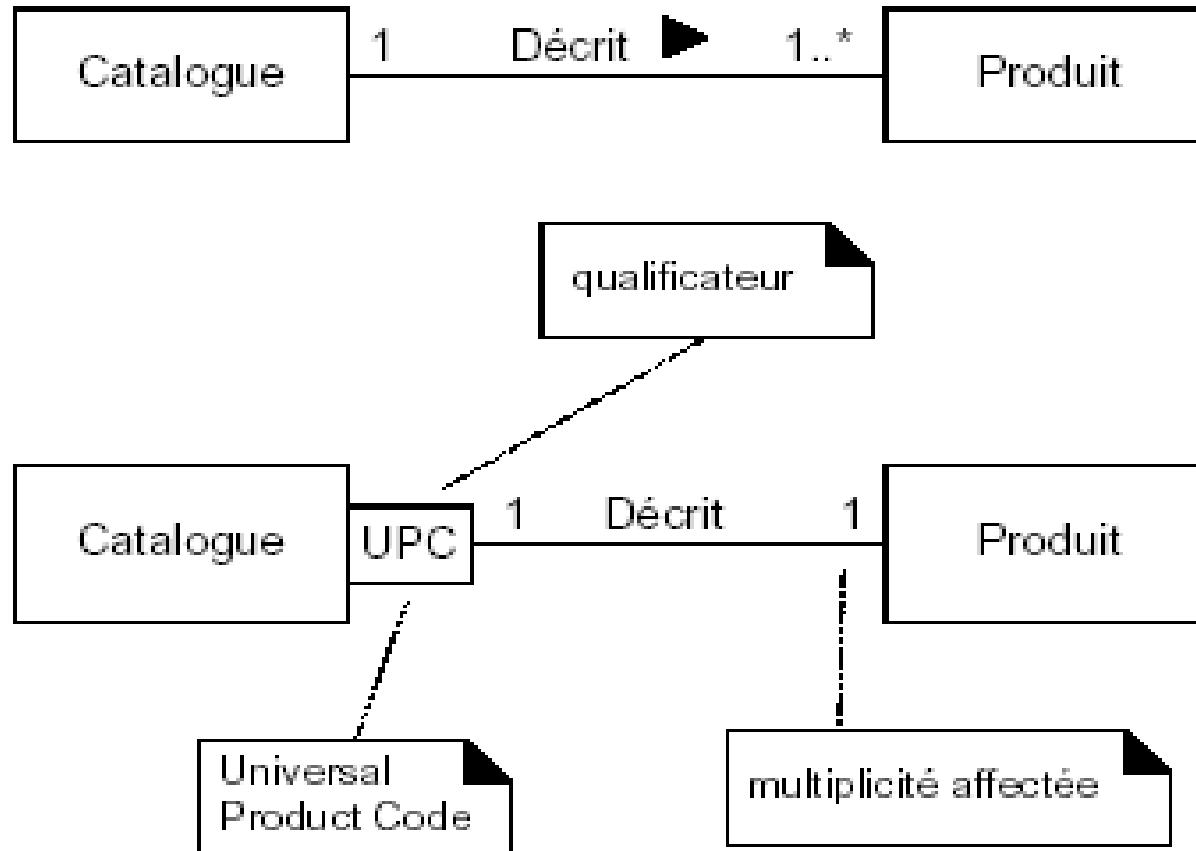
- Une **association dérivée** est une association qui peut se **déduire** des associations existantes



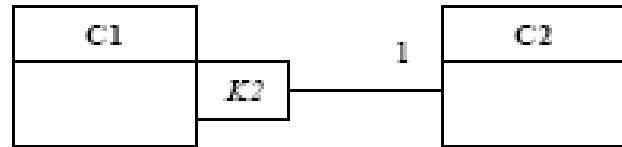
## **Relations entre classes : Association qualifiée**

- Restreindre la portée d'une association à quelques éléments ciblés (comme un ou plusieurs attributs) de la classe. Les éléments ciblés sont déterminés par un qualificatif.
- Un qualificatif agit toujours sur une association dont la multiplicité est plusieurs (avant que l'association ne soit qualifiée) du côté cible.
- L'association est appelée association qualifiée.
- Représenter par un rectangle positionné entre la terminaison d'association et la classe cible.

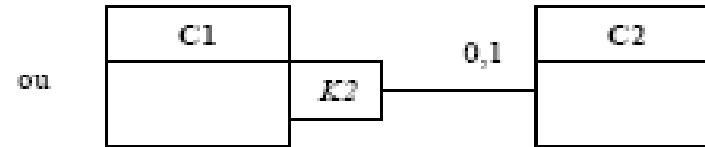
# Associations : associations qualifiées



# Associations : associations qualifiées

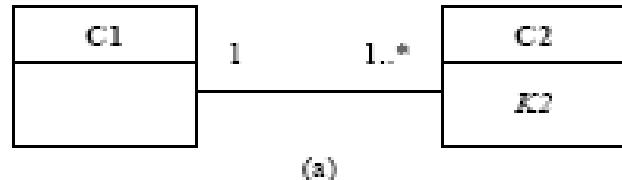


(a)

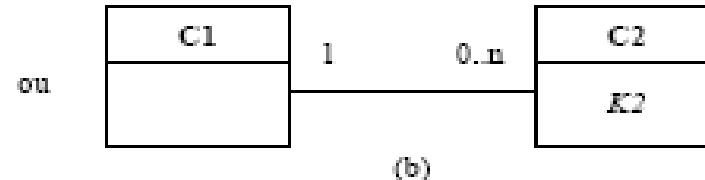


(b)

Il faut se noter que cette association sans qualification est équivalente à :



(a)

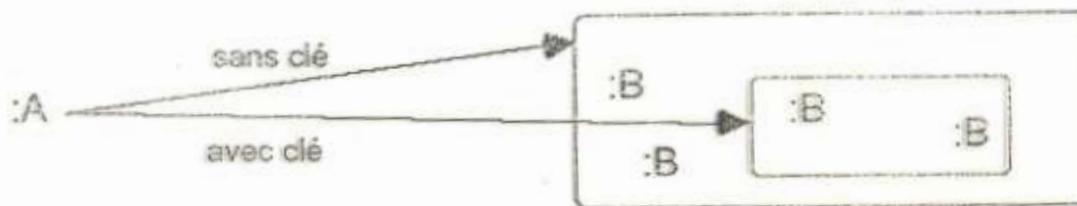


(b)

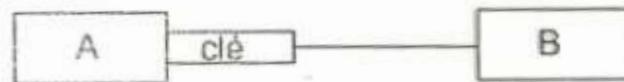
## 4.2.1 - La Phase d'analyse

### B6 – Type d'associations binaires : Association de qualification

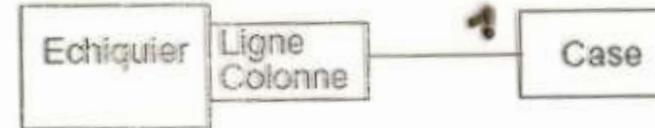
Une association qualifiée ou « qualification » consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association



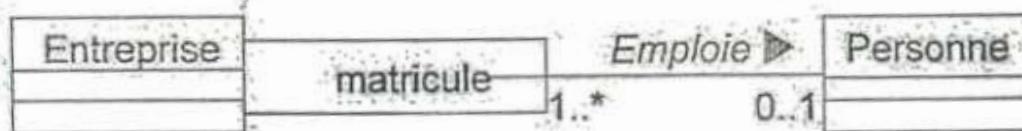
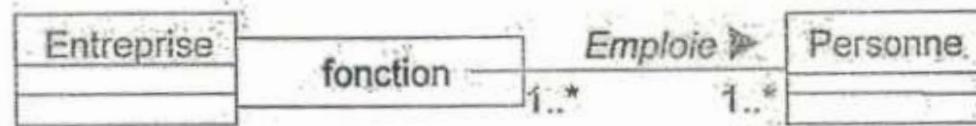
Cas général :



Exemples :



combinaison d'une ligne et d'une colonne pour identifier une case de l'échiquier

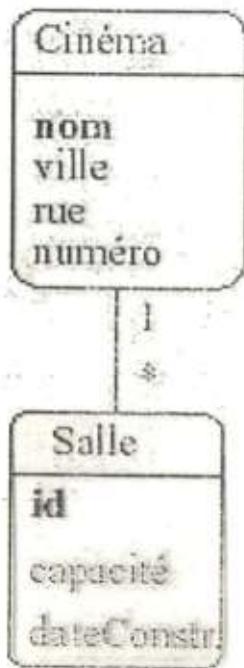


## 4.2.1 - La Phase d'analyse

### B7 – Type d'associations binaires : Association relative

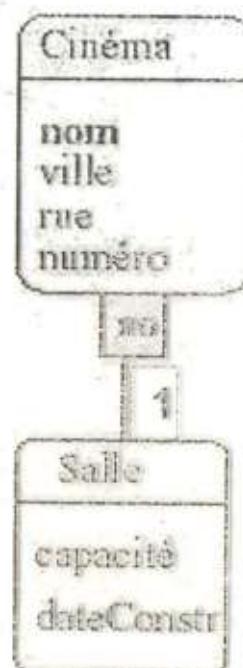
Dans UML, la qualification est utilisée pour représenter une association relative

Entité	Identifiant absolu
Cinéma	Nom cinéma
Salle	Id. salle



(a)

Entité	Identifiant absolu
Cinéma	Nom cinéma
Salle	Nom cinéma + N° Salle



(b)

#### Association hiérarchique standard

Salle = Entité forte

( identifiée de manière absolue  
avec l'attribut « Id » croissant )

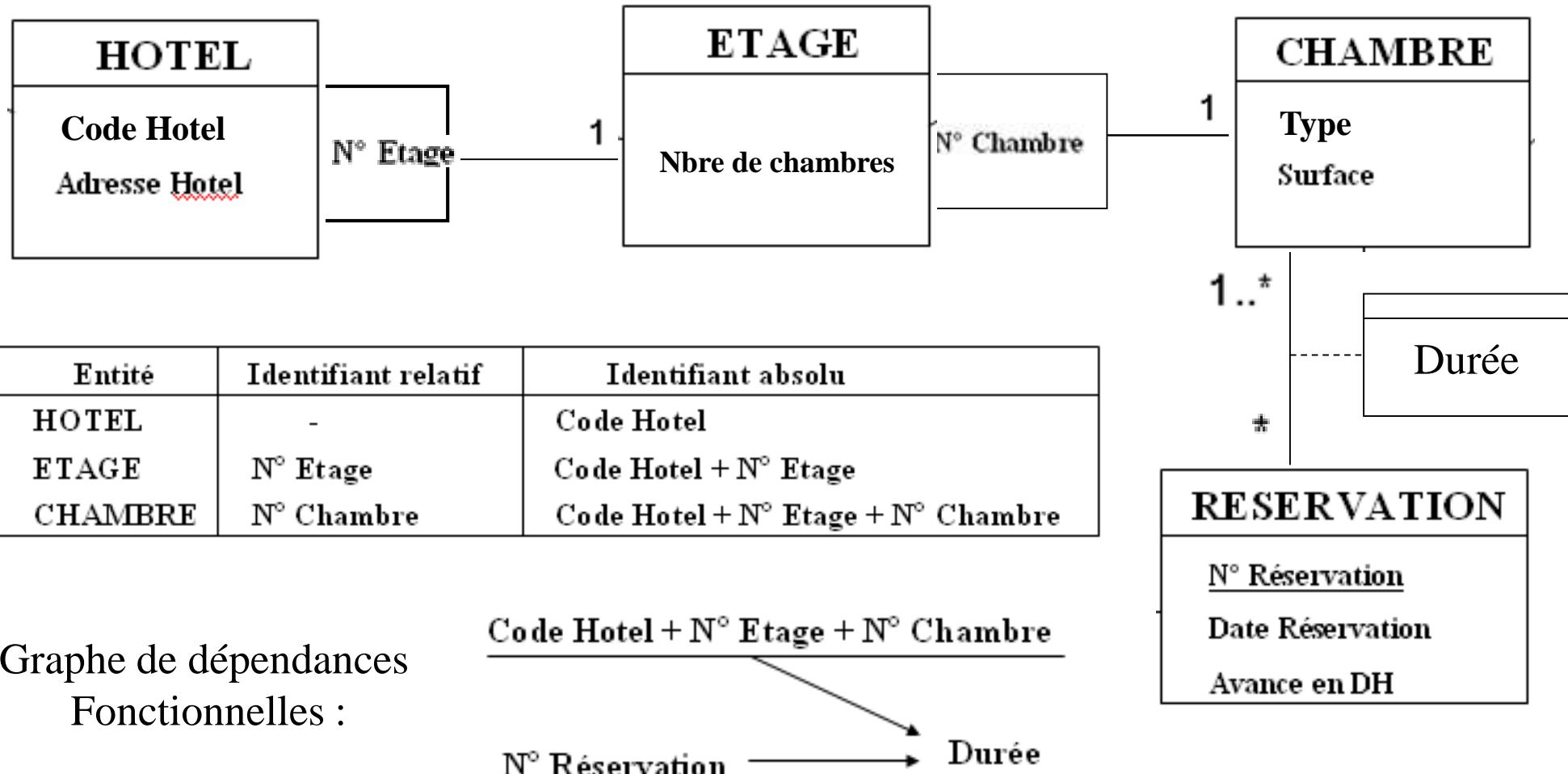
#### Association relative ( modélisée par une qualification )

Salle = Entité faible

( identifiée de manière relative par  
avec l'attribut « No » par rapport à un cinéma )

### **4.2.1 - La Phase d'analyse**

## **B7 – Type d’associations binaires : Associations relatives ( suite )**



## 4.2 - La Phase d'élaboration

### C – Le Diagramme d’objets

**Les diagrammes d’objets, ou diagrammes d’instances, montrent des objets et des liens. Comme les diagrammes de classes, les diagrammes d’objets représentent la structure statique.**

**Les diagrammes d’objets s’utilisent principalement pour montrer un contexte, par exemple avant ou après une interaction, mais également pour faciliter la compréhension des structures complexes, comme les structures récursives.**

ObjetA

L’objet « ObjetA » n’est pas classifié

ObjetB :  
ClasseB

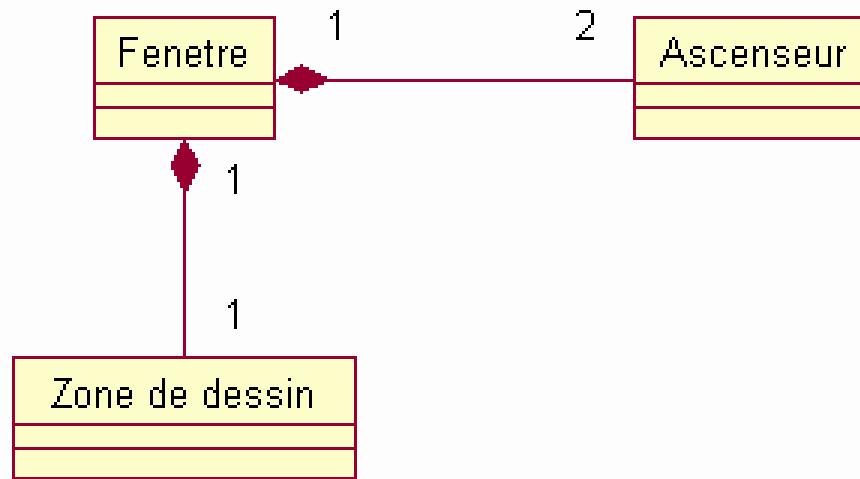
L’objet « : ObjetB » est instance de la classe « ClasseB »

: ClasseC

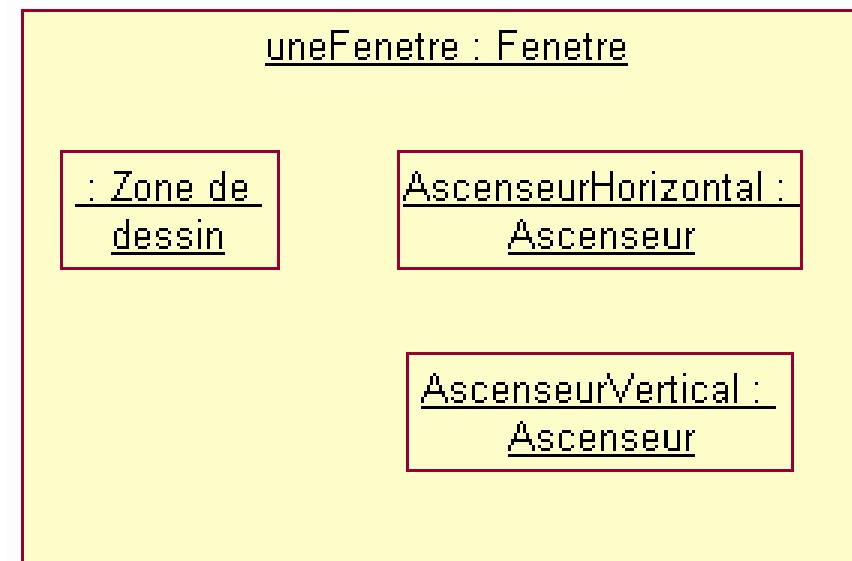
L’objet « : ClasseC » est une instance anonyme de la classe « ClasseC »

## 4.2 - La Phase d'élaboration

### C – Le Diagramme d’objets ( Suite 1 ) : Exemple d’objet composite

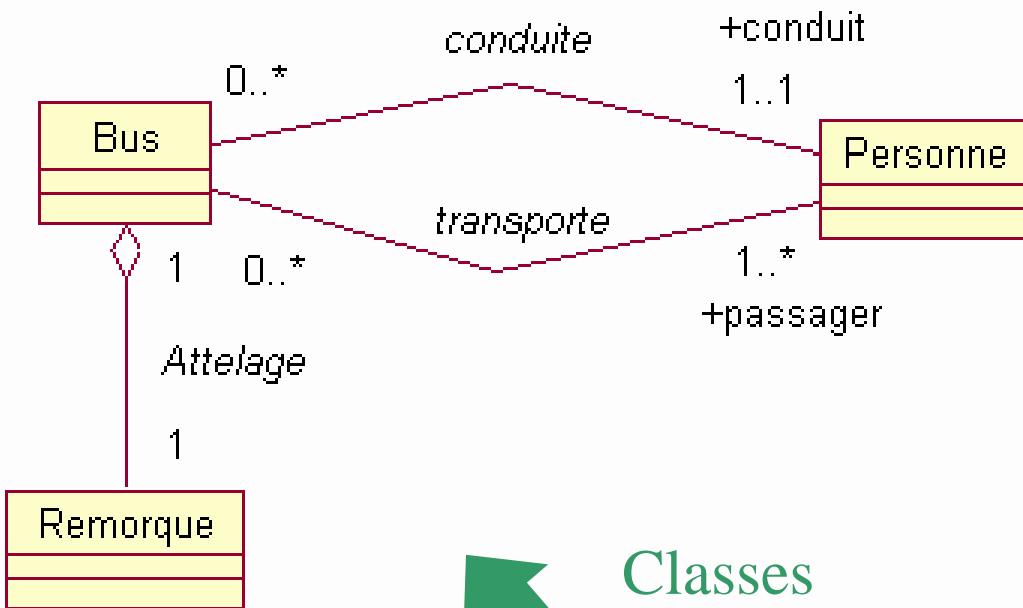


Classes  
↓  
Objets

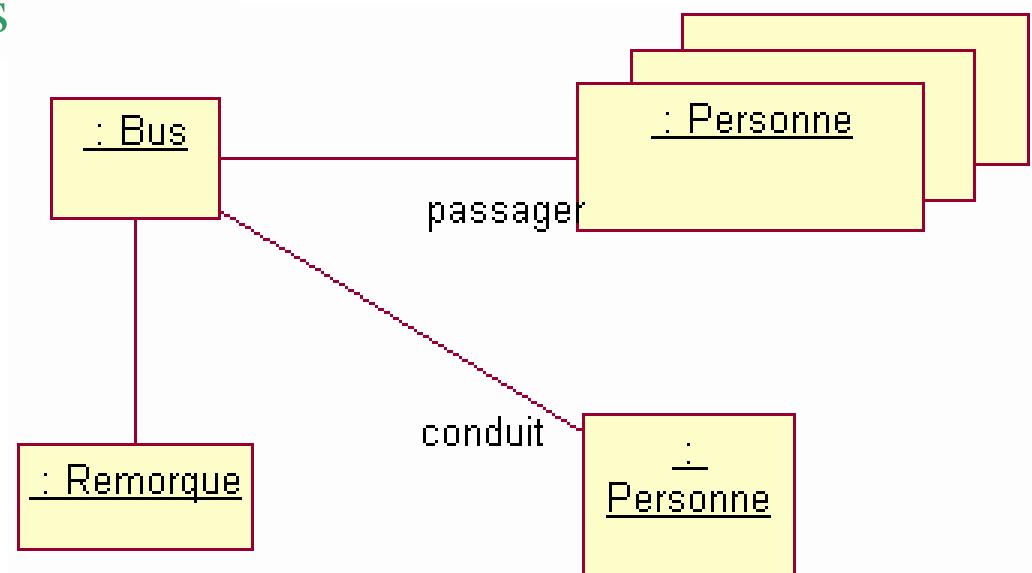


## 4.2 - La Phase d'élaboration

### C – Le Diagramme d’objets ( Suite 2 ) : Exemple d’associations multiples

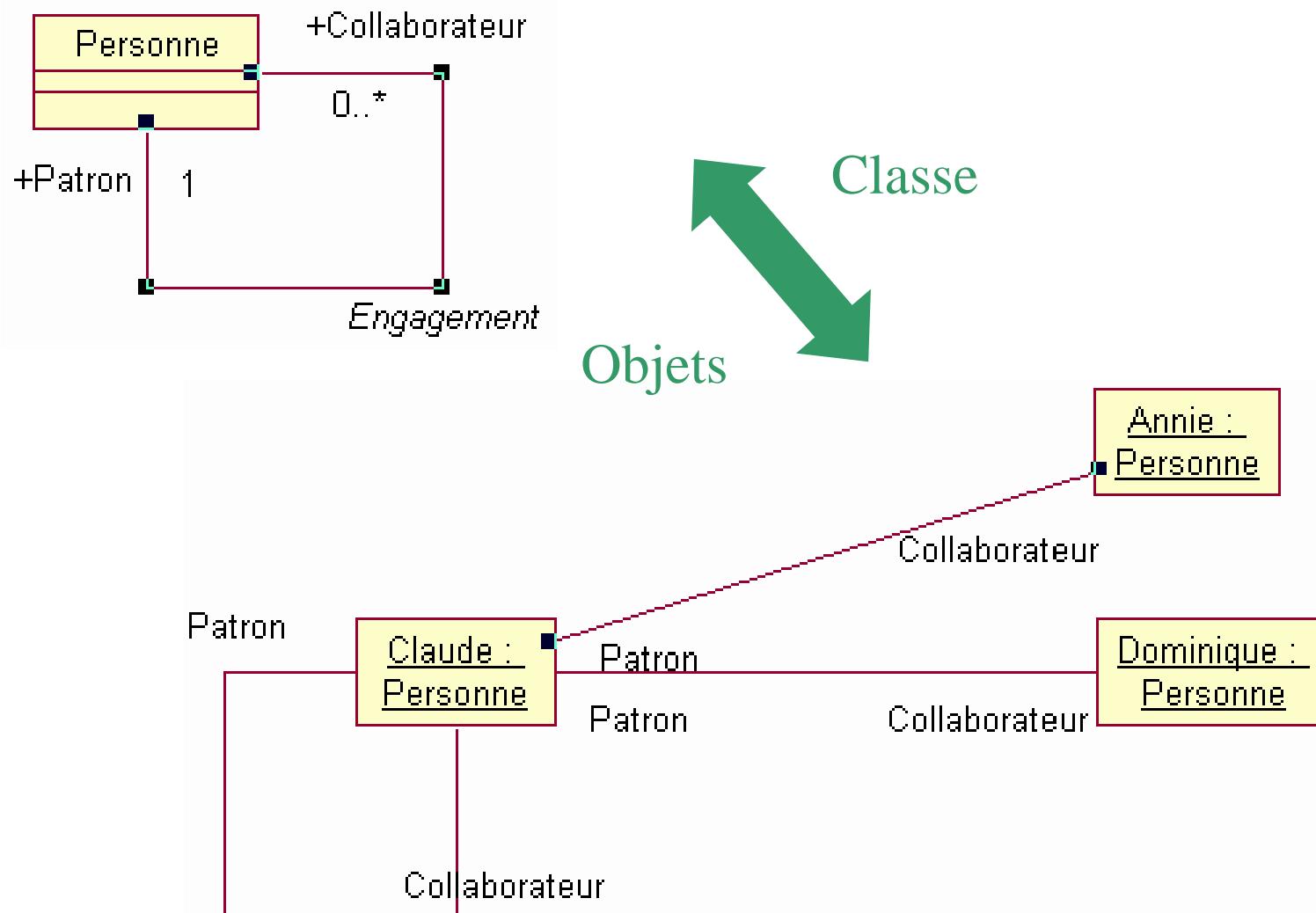


Classes  
↓  
Objets



## 4.2 - La Phase d'élaboration

### C – Le Diagramme d’objets ( Suite 3 ) : Exemple d’association réflexive



## 4.2 - La Phase d'élaboration

### D – L'axe dynamique

- Le modèle dynamique représente les séquences **d'événements, d'états et de réactions** qui peuvent survenir dans un système.
- Il est intimement lié au modèle statique ( diagrammes de classe et d'objet ) et au modèle fonctionnel ( diagramme des cas d'utilisation ) et décrit les aspects de contrôle d'un système en prenant compte le **temps, le séquencement des opérations** et les **interactions** entre objets
- 3 diagrammes fondamentaux sont utilisés dans le modèle dynamique :
  - **Diagrammes de Séquence et de collaboration**
  - **Diagramme Etats-Transitions**
  - **Diagramme d'activité**

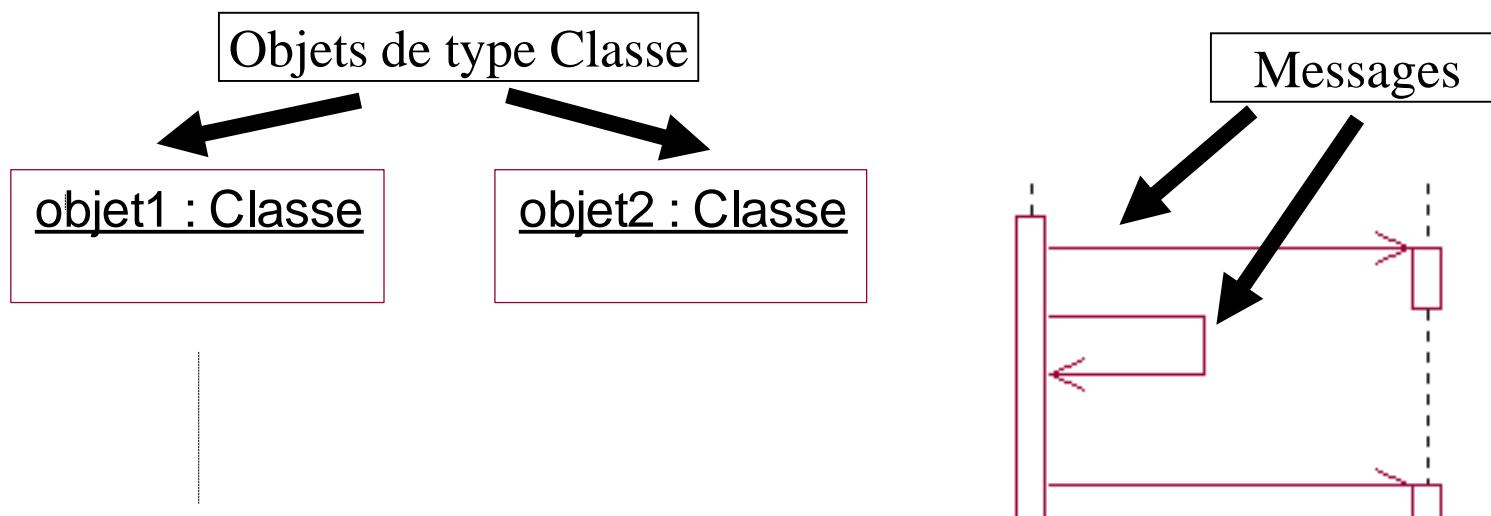
## 4.2 - La Phase d'élaboration

### E – Le Diagramme de Séquence

- **Définition :**

Le diagramme de séquence décrit le déroulement d'une interaction entre les objets dans le cadre de la réalisation d'un scénario de cas d'utilisation en mettant en évidence la dimension temporelle de cette interaction .

- Il y a autant de diagrammes de séquence qu'il y a de scénarios
- Un **Scénario** montre une séquence particulière d'interactions entre objets, dans un seul contexte d'exécution du système
- Un **scénario** peut être vu comme une des instances possibles d'un **Use Case**.
- On y fait intervenir des **objets**, des **messages** et des **événements**



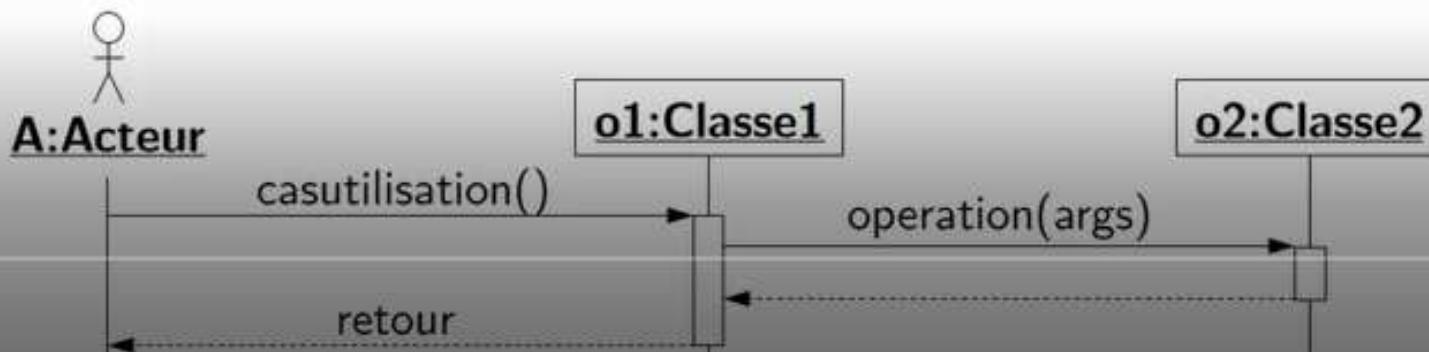
# Diagrammes de séquence (conception)

## Éléments du diagramme de séquence

- Acteurs
- Objets (instances)
- Messages (cas d'utilisation, appels d'opération)

Principes de base : Représentation graphique de la **chronologie** des échanges de messages avec le système ou au sein du système

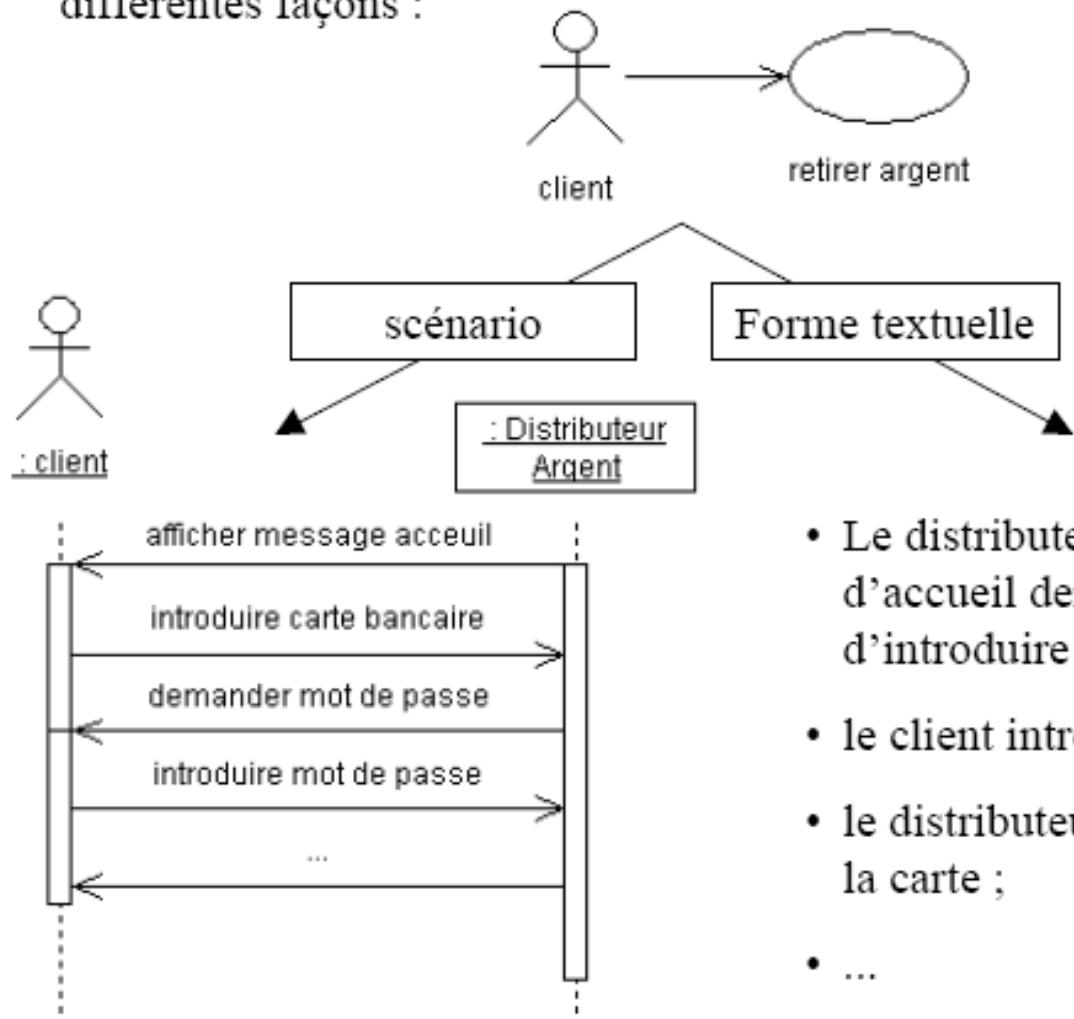
- « Vie » de chaque entité représentée verticalement
- Échanges de messages représentés horizontalement



## 4.1 - La Phase de spécification

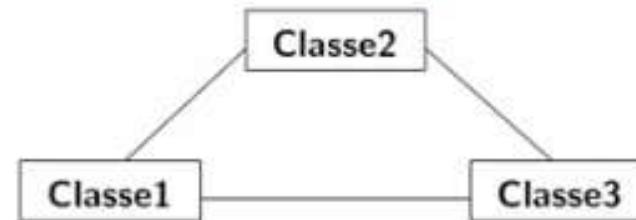
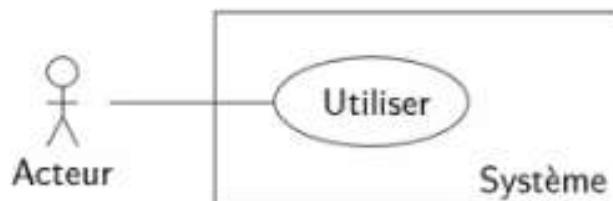
### Documentation du scénario d'un cas d'utilisation

- Les use cases peuvent être décrits sous la forme de flots d'événements de différentes façons :



## 4.2 - La Phase d'élaboration

### D – L'axe dynamique



Communication entre acteurs et système via une interface (texte, web, physique...)

## 4.2 - La Phase d'élaboration

### D – L'axe dynamique

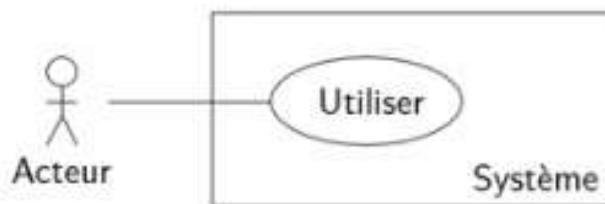


Diagramme de cas d'utilisation

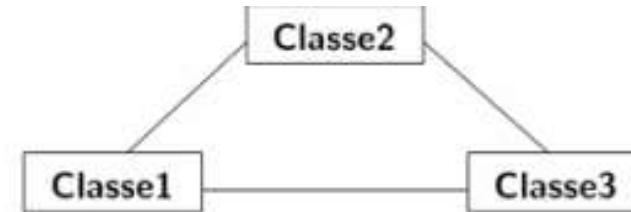
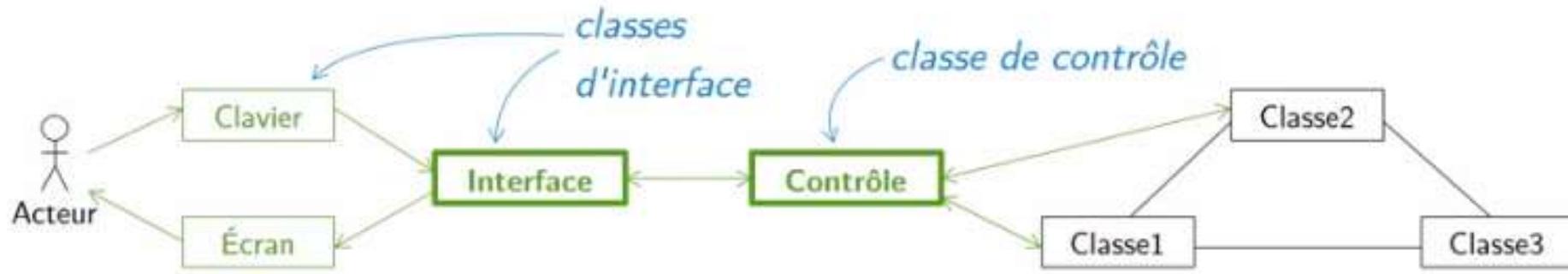


Diagramme de classes du système



**Solution :** Création de **classes de contrôle** et de **classes d'interface** qui :

- gèrent les interactions avec les acteurs
- encapsulent le résultat des opérations

## 4.2 - La Phase d'élaboration

### D – L'axe dynamique

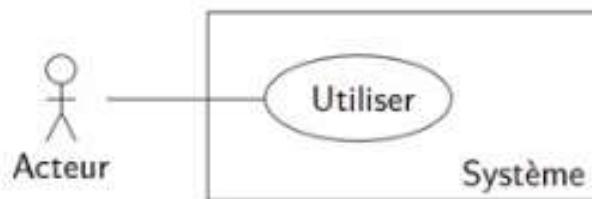


Diagramme de cas d'utilisation

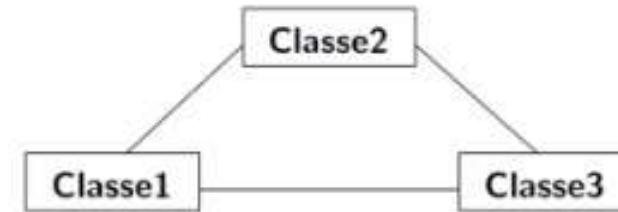
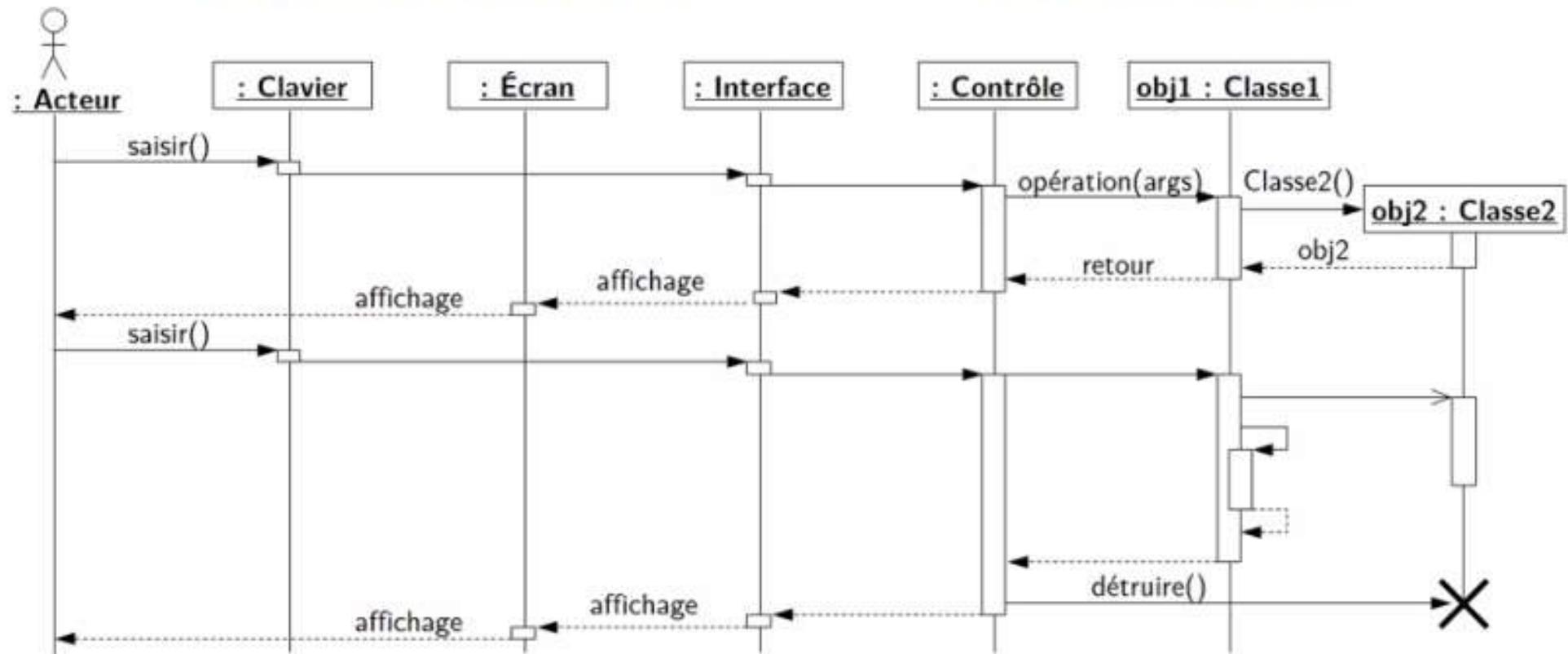


Diagramme de classes



## 4.2 - La Phase d'élaboration

### E – Le Diagramme de Séquence ( suite 1 )

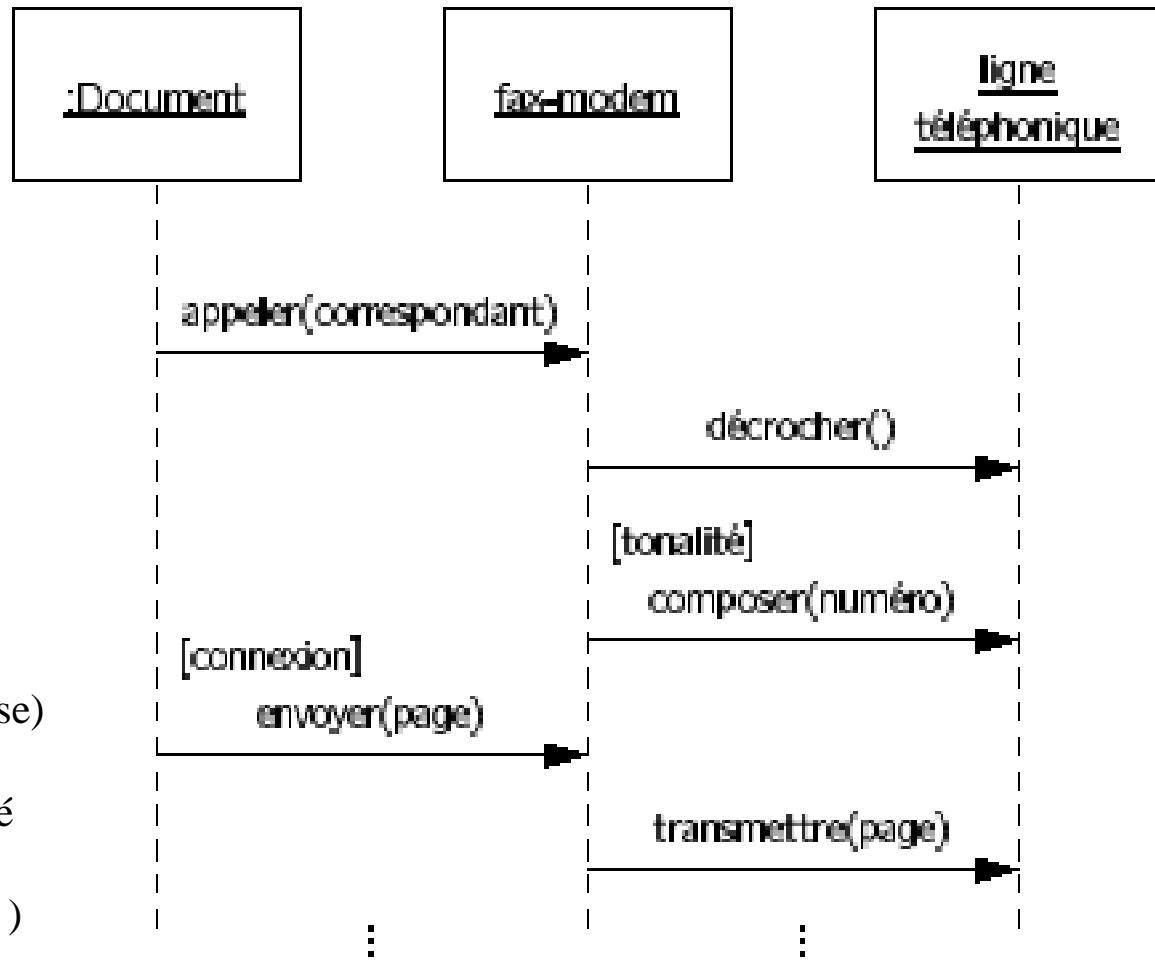
- L'envoi d'un message d'un objet à un autre se dénote par une flèche pleine reliant les lignes de vie de ces deux objets .

- La forme élémentaire de l'étiquette d'un message est la suivante :

[ garde ] message ( paramètres )

où

- **Garde** est une condition devant être satisfaite afin de pouvoir envoyer le message ( si cette condition est toujours vraie , elle peut être omise)
- **Message** est l'identificateur du message envoyé
- **Paramètres** est une liste ( éventuellement vide ) de paramètres , séparés par de virgules )

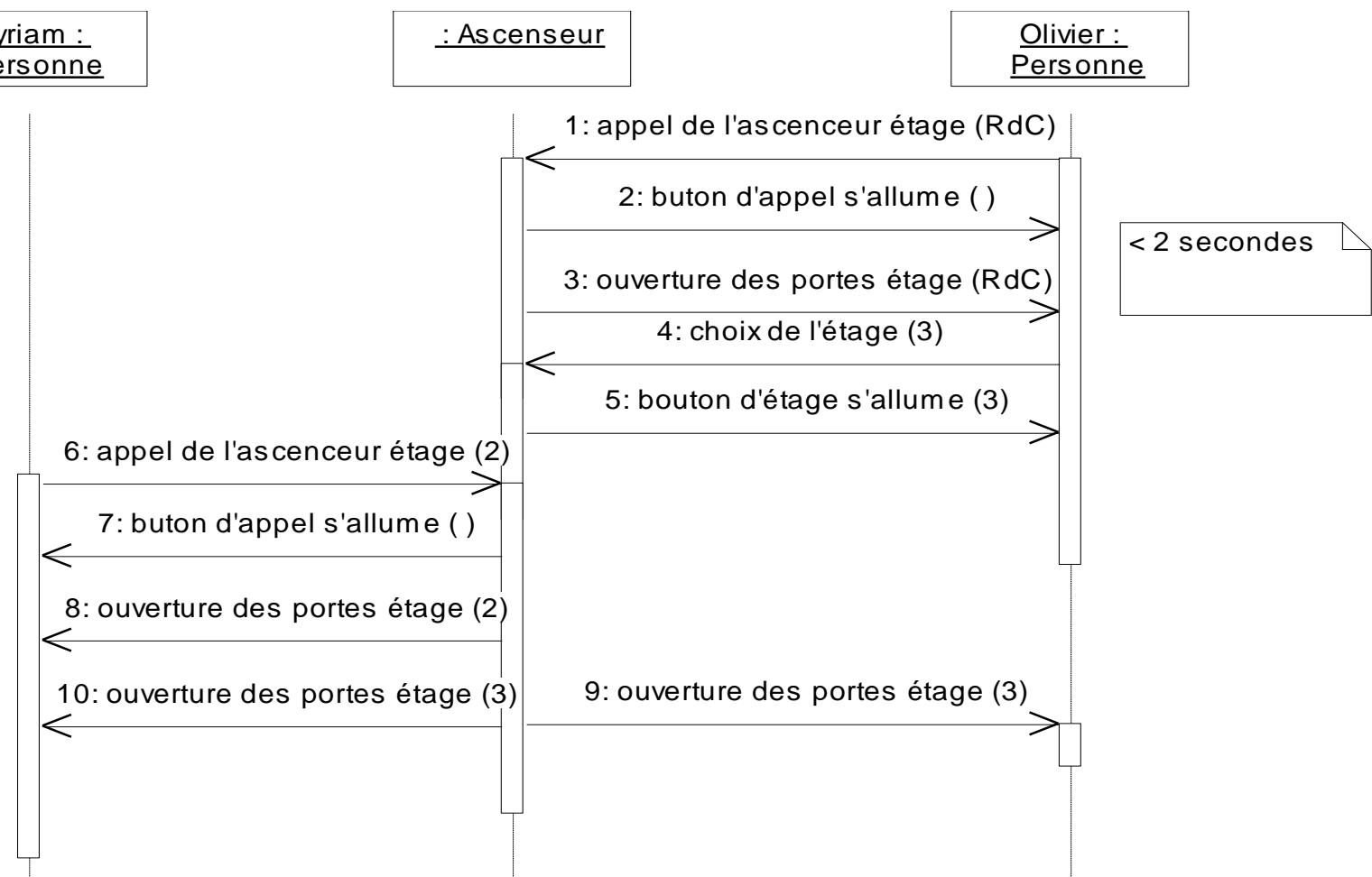


## 4.2 - La Phase d'élaboration

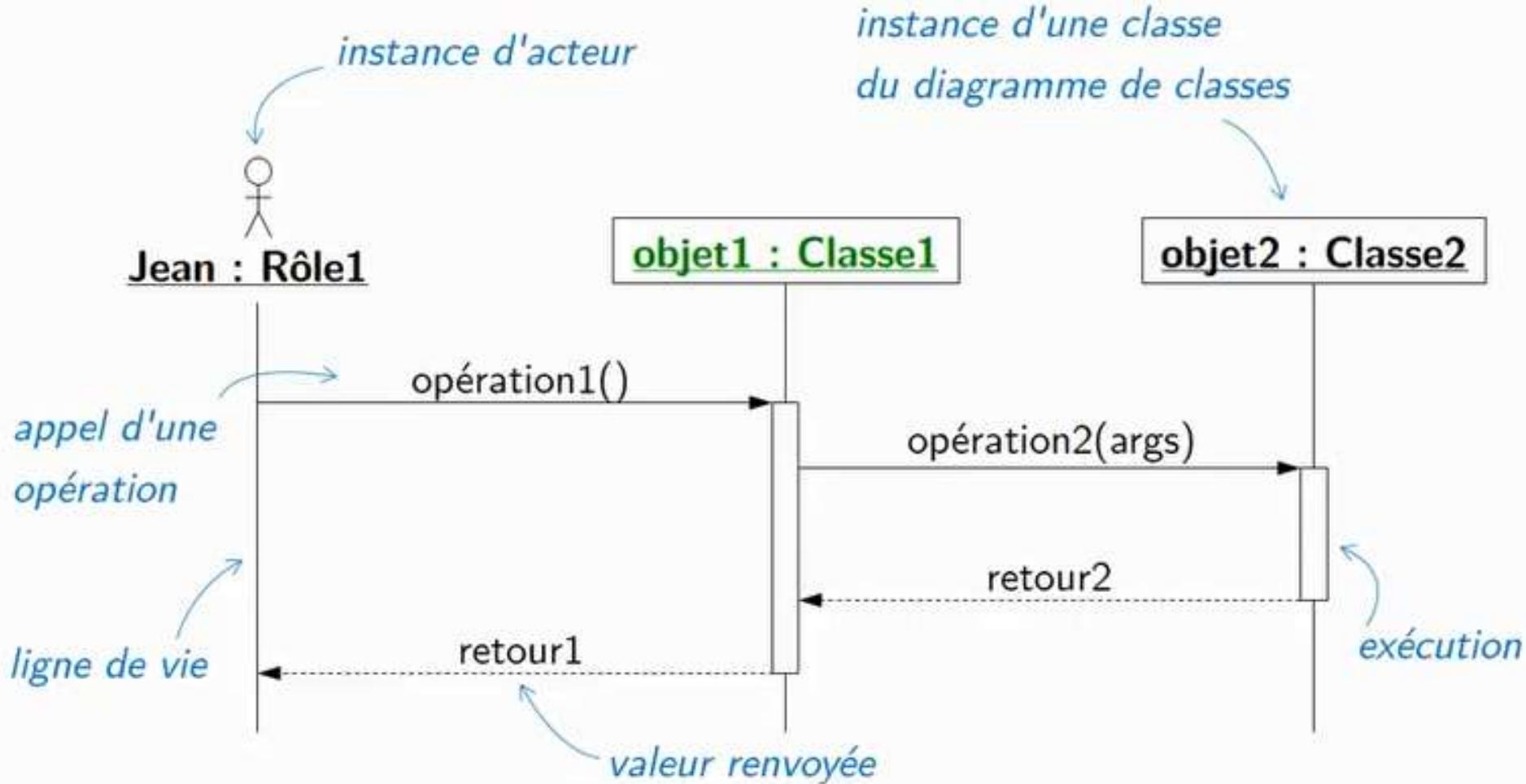
### E – Le Diagramme de Séquence ( suite 2 ) : Interation avec le diagramme de classe



### Exemple de L'ascenceur



# Éléments de base



## 4.2 - La Phase d'élaboration

### E – Le Diagramme de Séquence ( suite 3 )

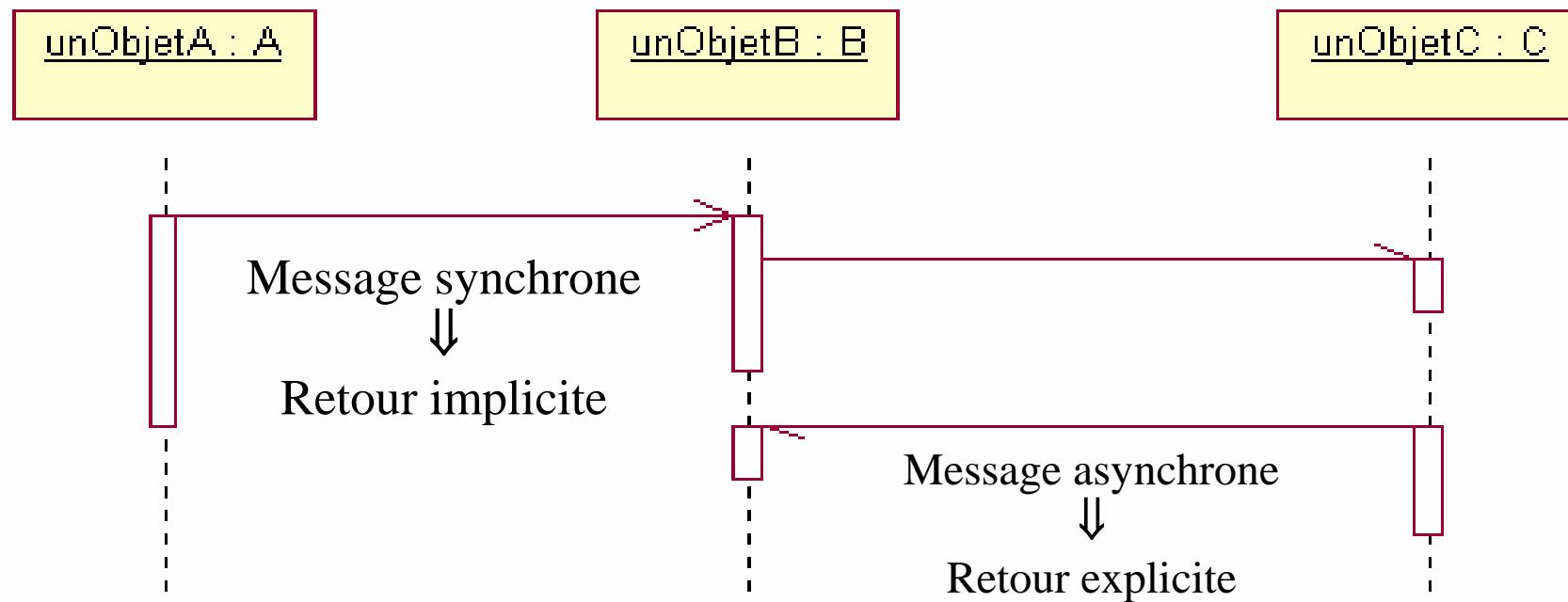
Les diagrammes de séquence distinguent deux grandes catégories d'envoi de message :

- Les envois synchrones pour lesquels l'émetteur est bloqué et attend que l'appelé ait fini de traiter le message
- Les envois asynchrones pour lesquels l'émetteur n'est pas bloqué et peut continuer son exécution.

Les diagrammes de séquence permettent de représenter les périodes d'activité des objets.

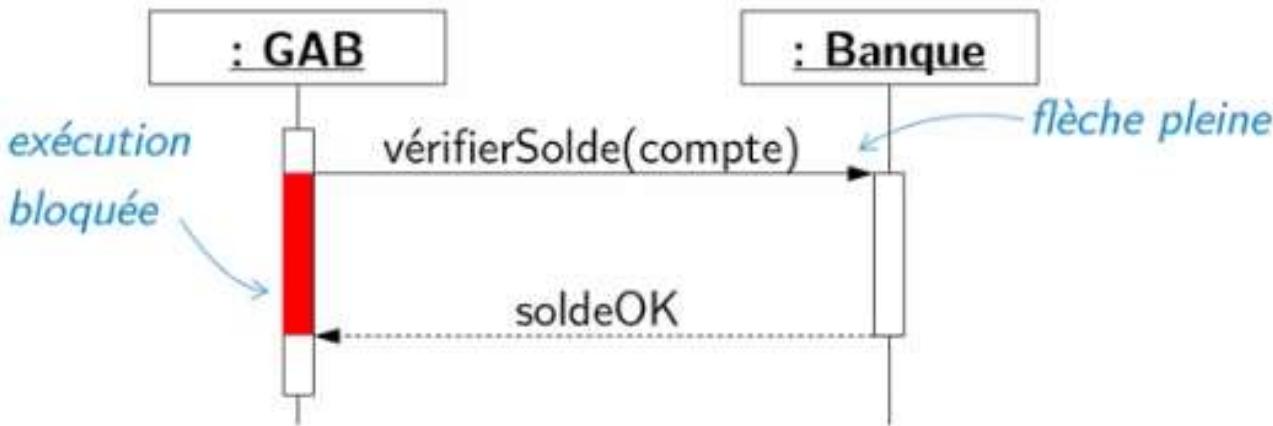
Une période d'activité correspond au temps pendant lequel un objet effectue une action, soit directement, soit par l'intermédiaire d'un autre objet qui lui sert de sous-traitant.

Les périodes d'activité se représentent par des bandes rectangulaires placées sur les lignes de vie. Le début et la fin d'une bande correspondent respectivement au début et à la fin d'une période d'activité.

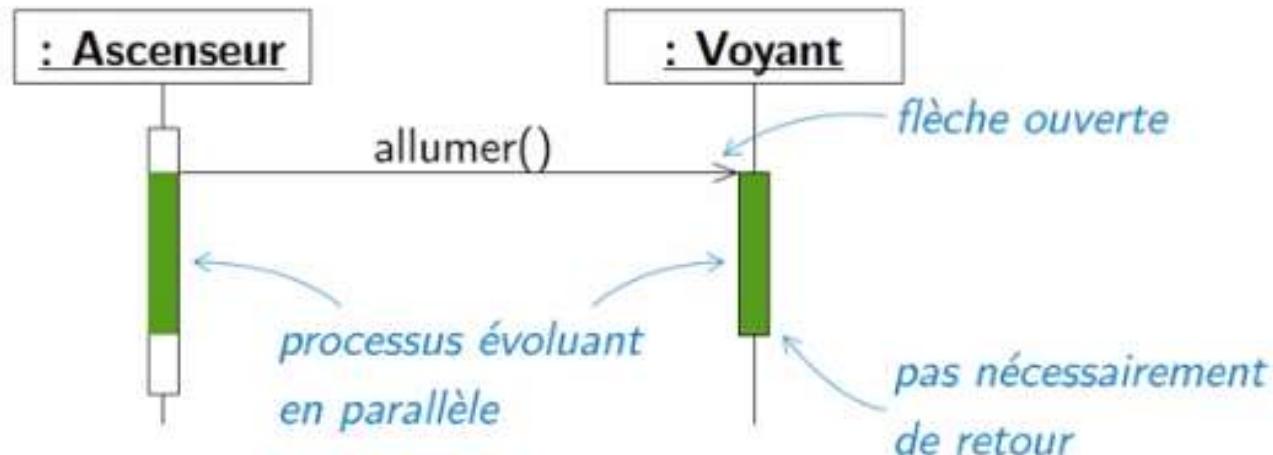


# Types de messages

Message synchrone : Émetteur bloqué en attente du retour

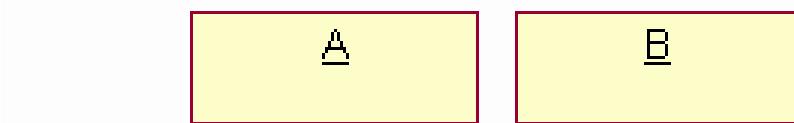
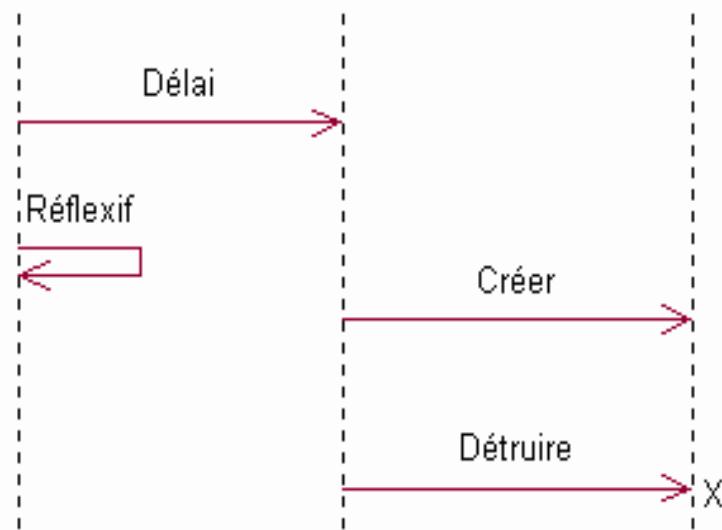
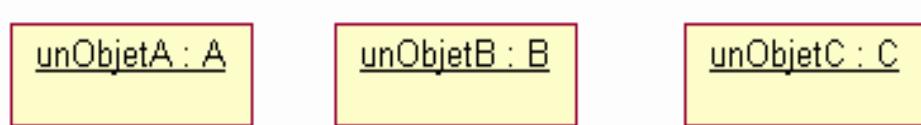


Message asynchrone : Émetteur non bloqué, continue son exécution

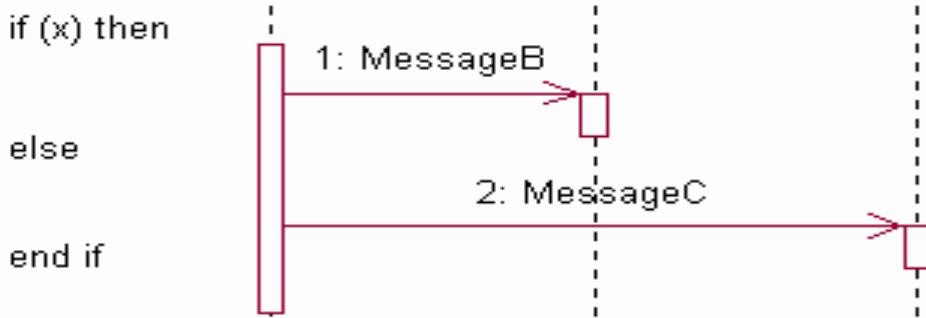
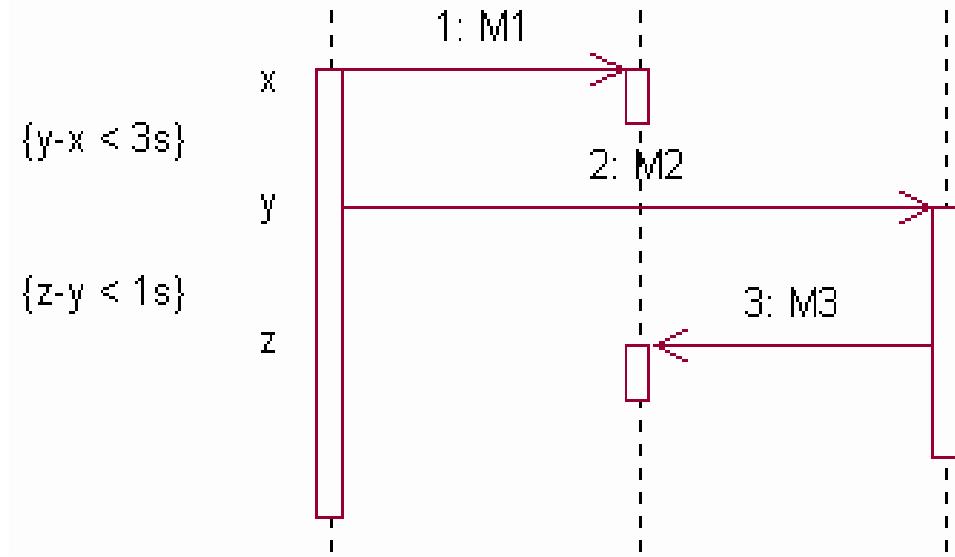
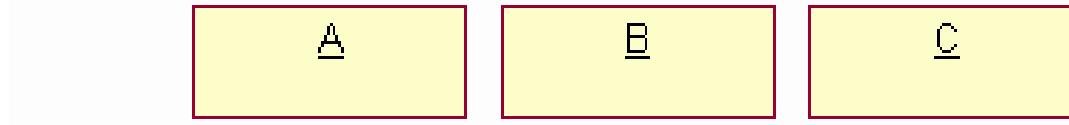


## 4.2 - La Phase d'élaboration

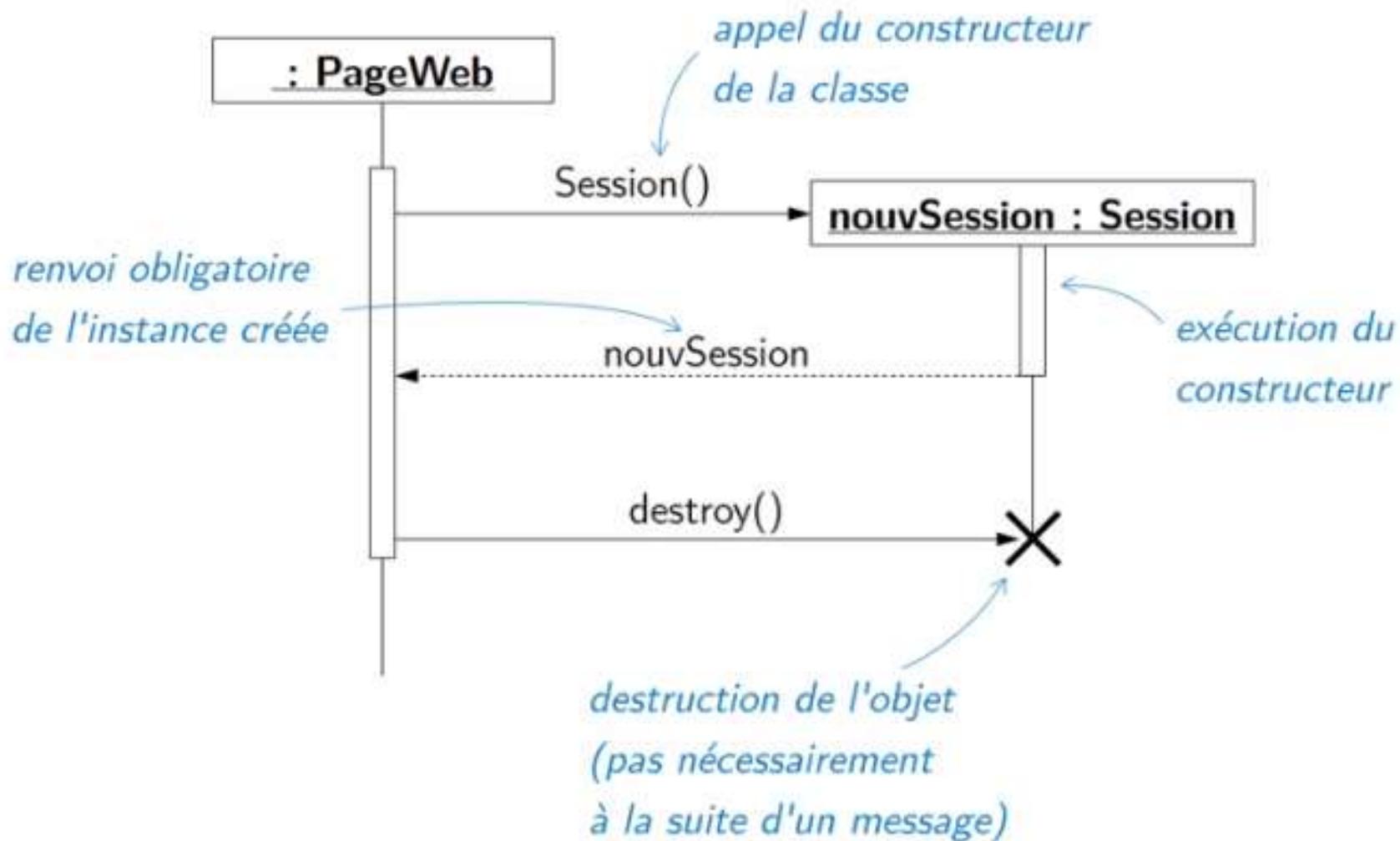
### E – Le Diagramme de Séquence ( suite 4 ) : Règles de représentations



while (x)  
loop  
end loop



# Constructeur Destructeur

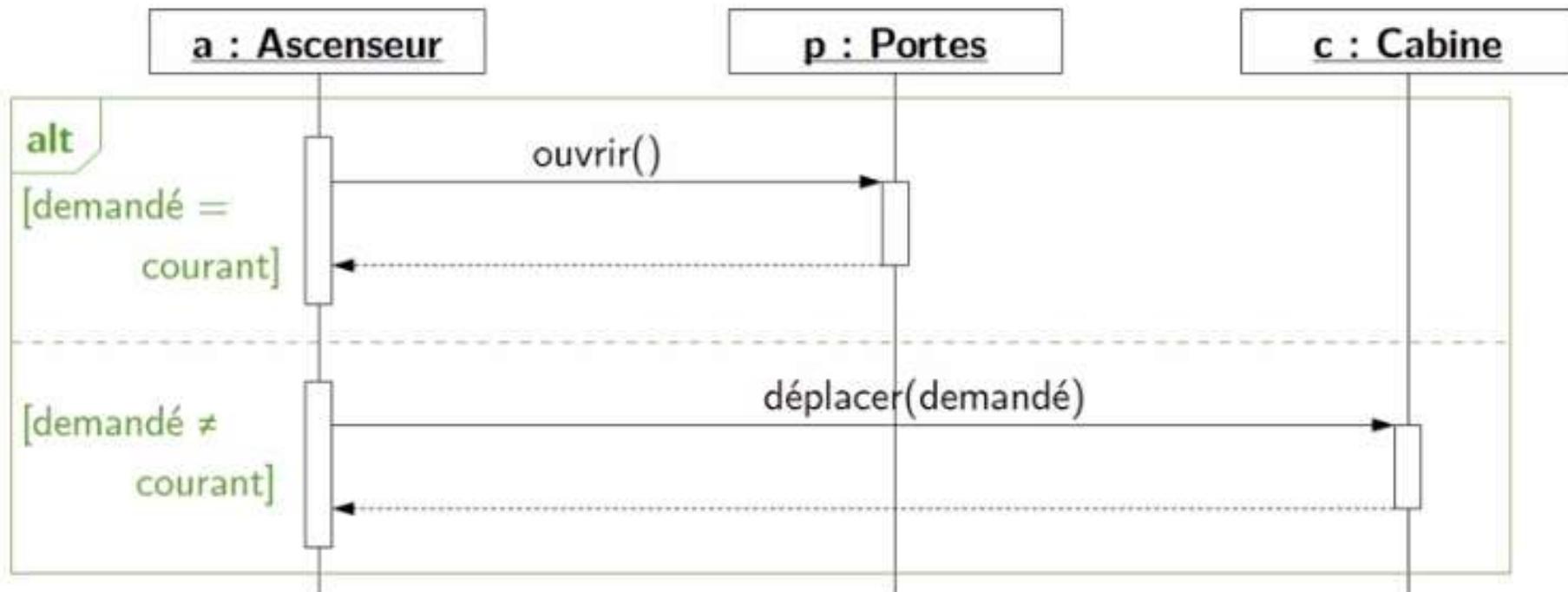


# Alternative

Principe : Condition à l'envoi d'un message

Notation :

- Deux diagrammes
- Bloc d'alternative **alt**

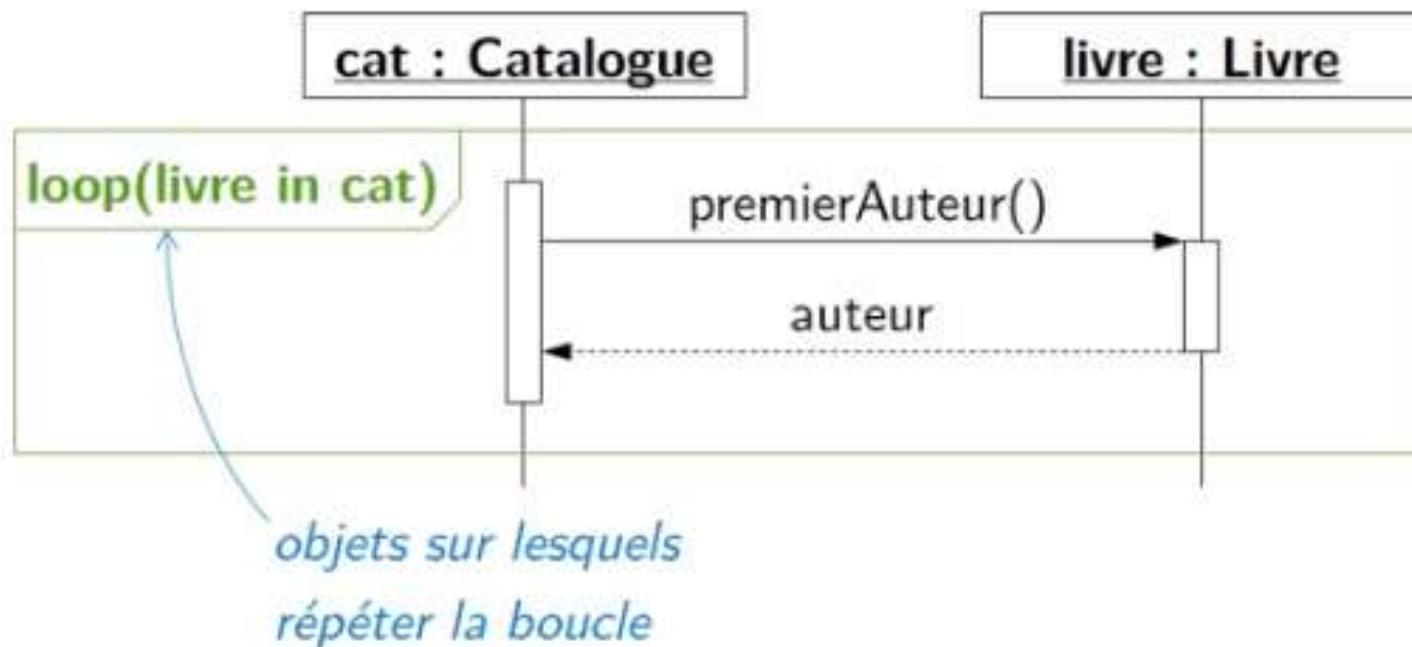


# Boucle

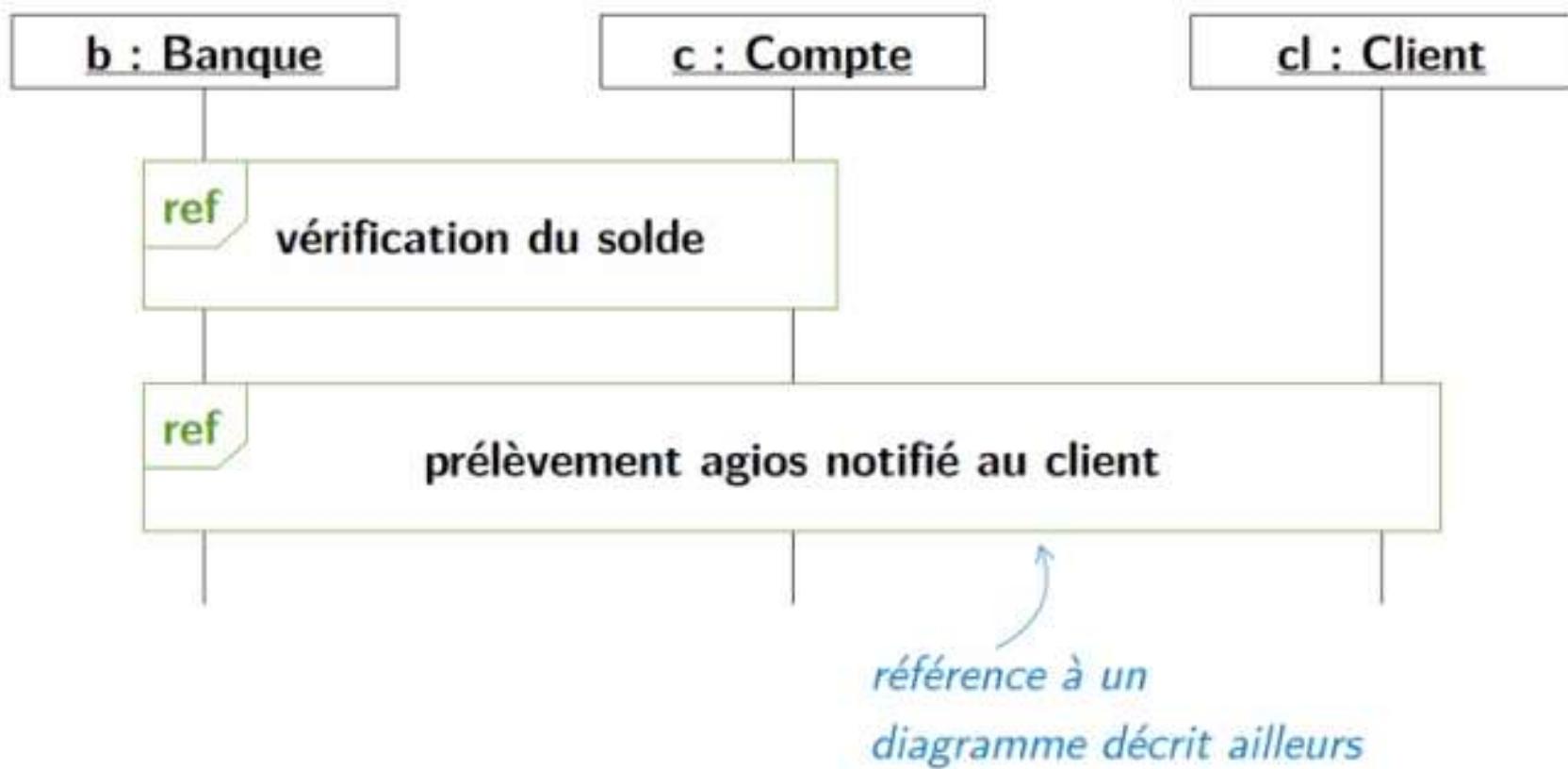
Principe : Répéter un enchaînement de messages

Notation :

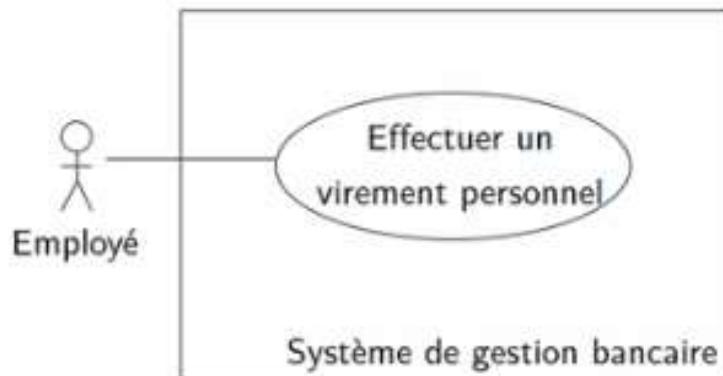
- Note
- Bloc de boucle **loop**



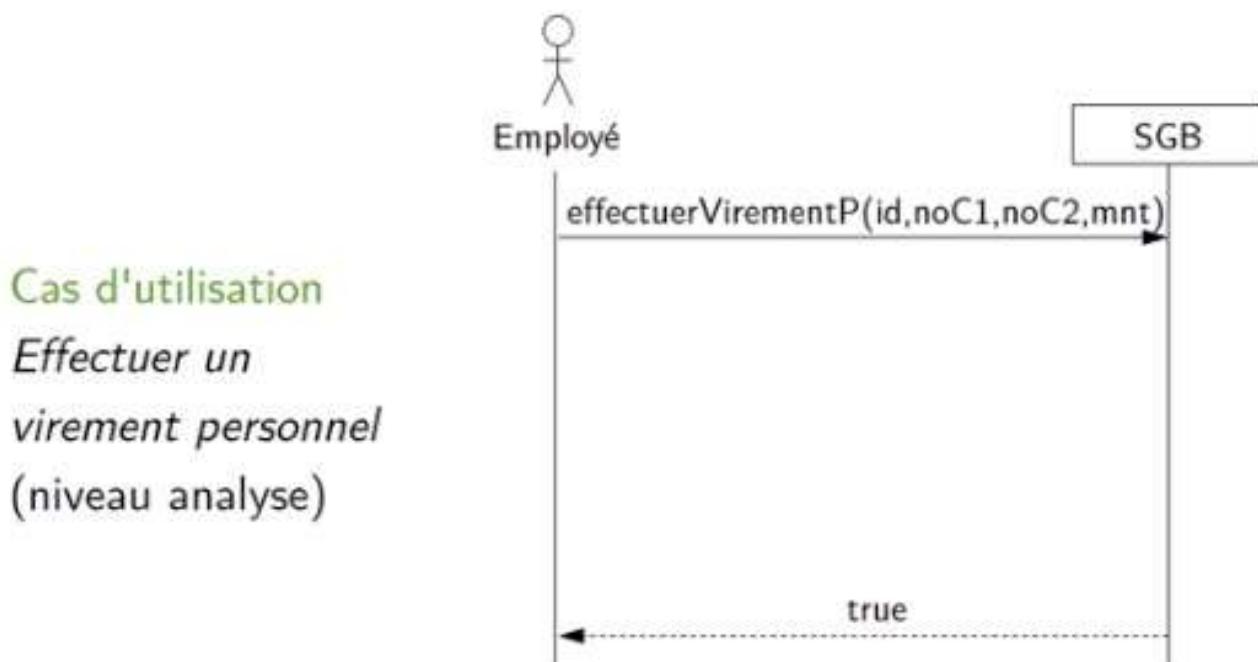
# Référence à un autre diagramme



# Exemple - Analyse



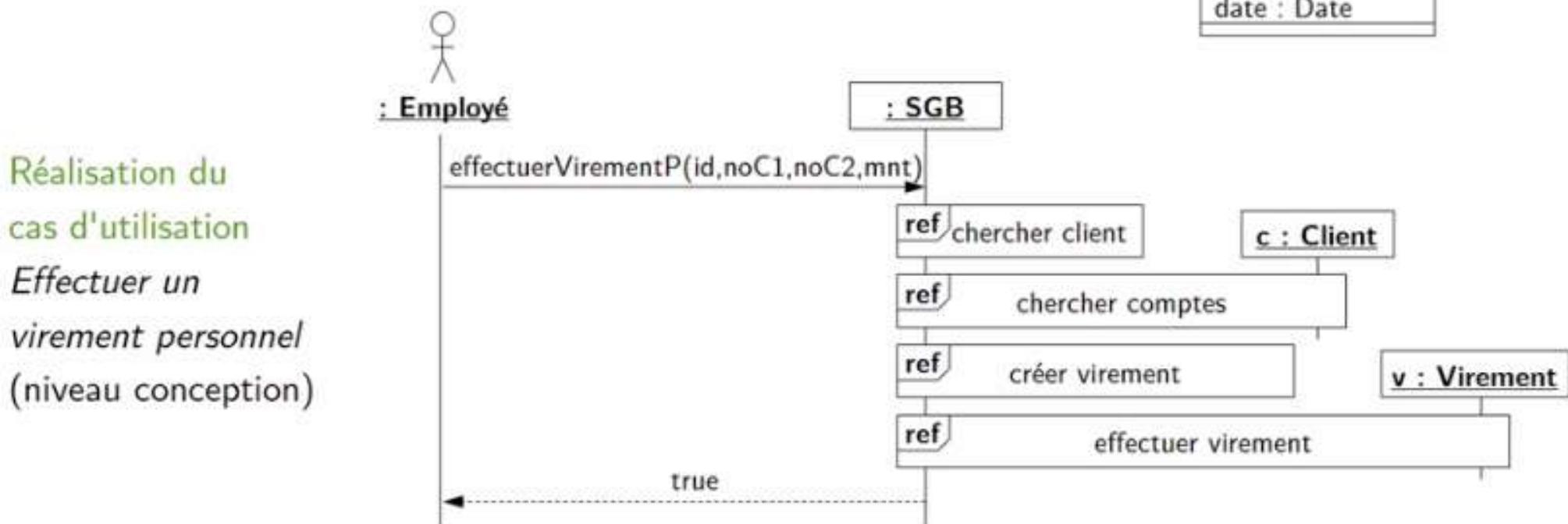
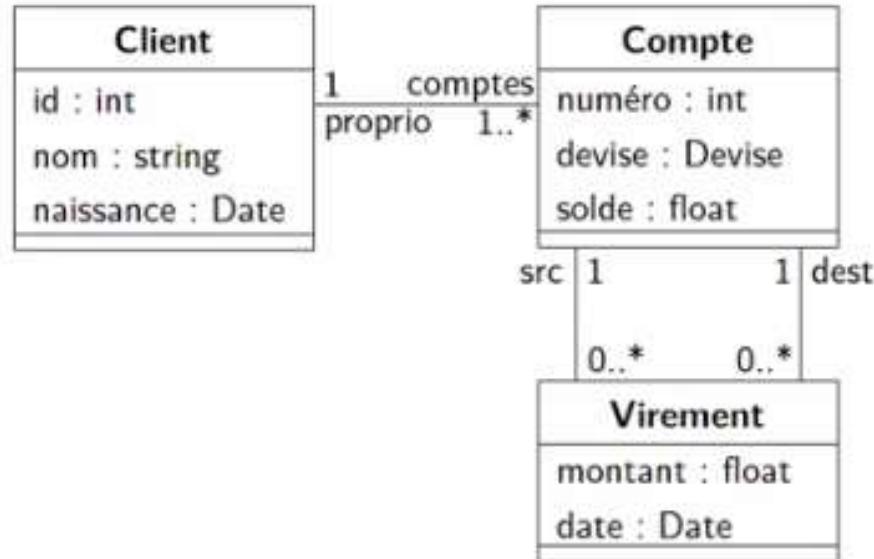
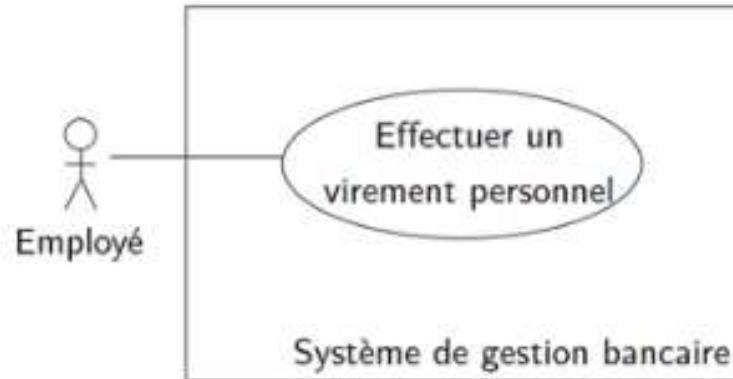
Client		Compte	
id : int	1 proprio	comptes 1..*	numéro : int
nom : string			devise : Devise
naissance : Date			solde : float
src		1	1 dest
0..*		0..*	
Virement			
montant : float			
date : Date			



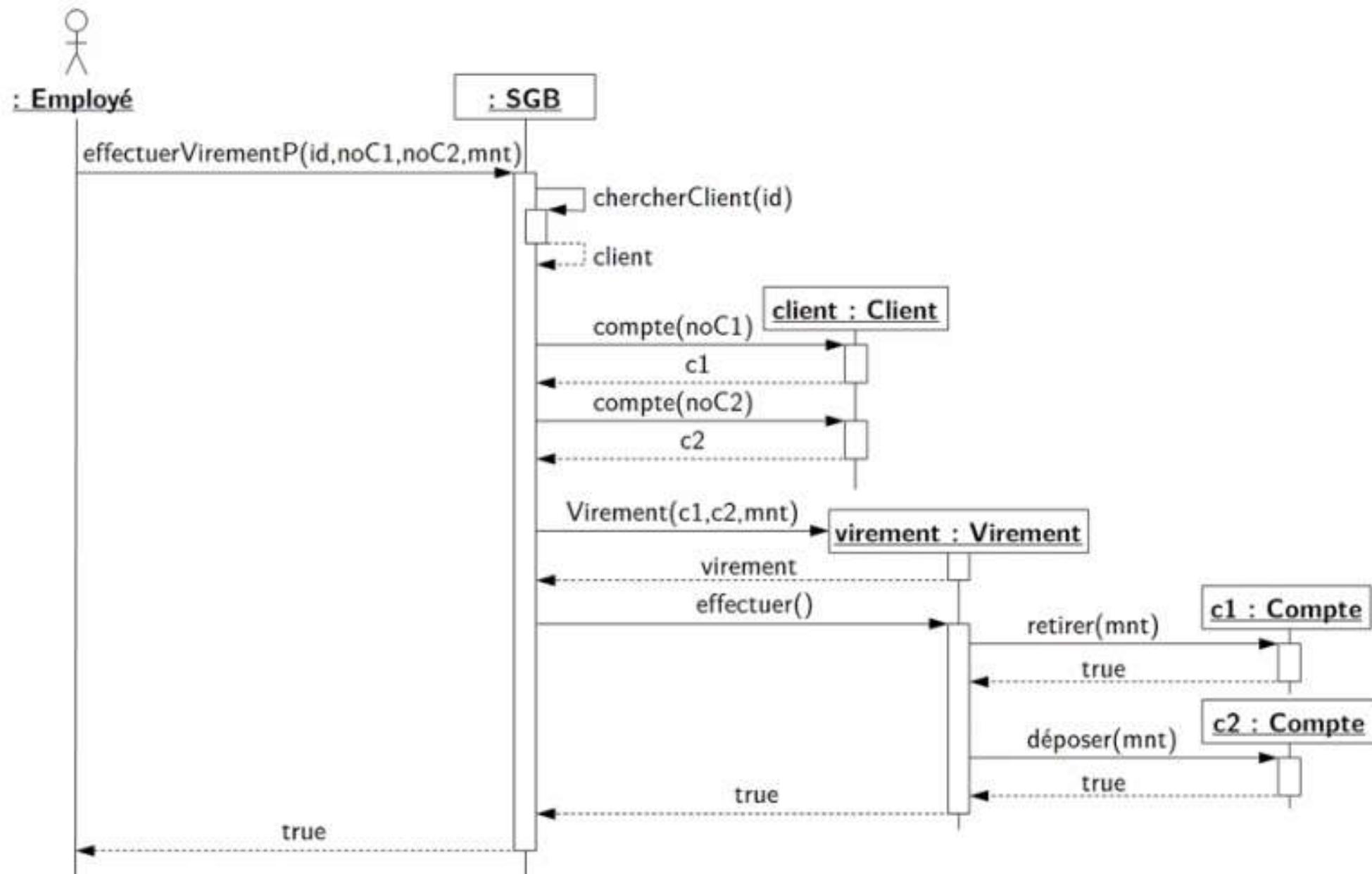
Cas d'utilisation

*Effectuer un  
virement personnel  
(niveau analyse)*

# Exemple - Conception

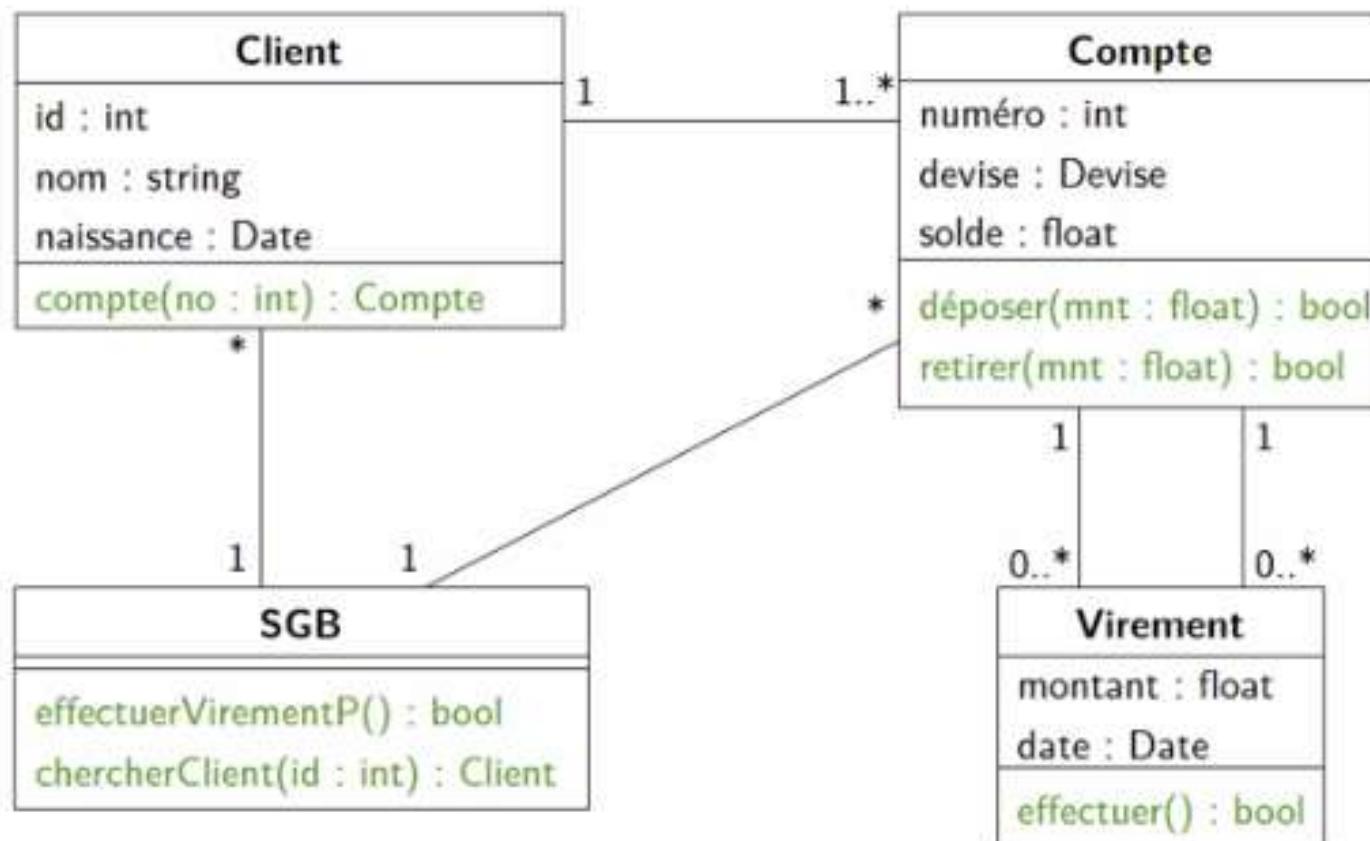


# Exemple - Conception



# Exemple - Conception

Diagramme de classes complété avec les classes techniques et les opérations nécessaires



# Quelques règles

## Messages entre acteurs et interface

- « Fausses » opérations liées au **cas d'utilisation** (même nom)
- Arguments (**saisis**) et valeurs de retour (**affichées**) **simples** : texte, nombre

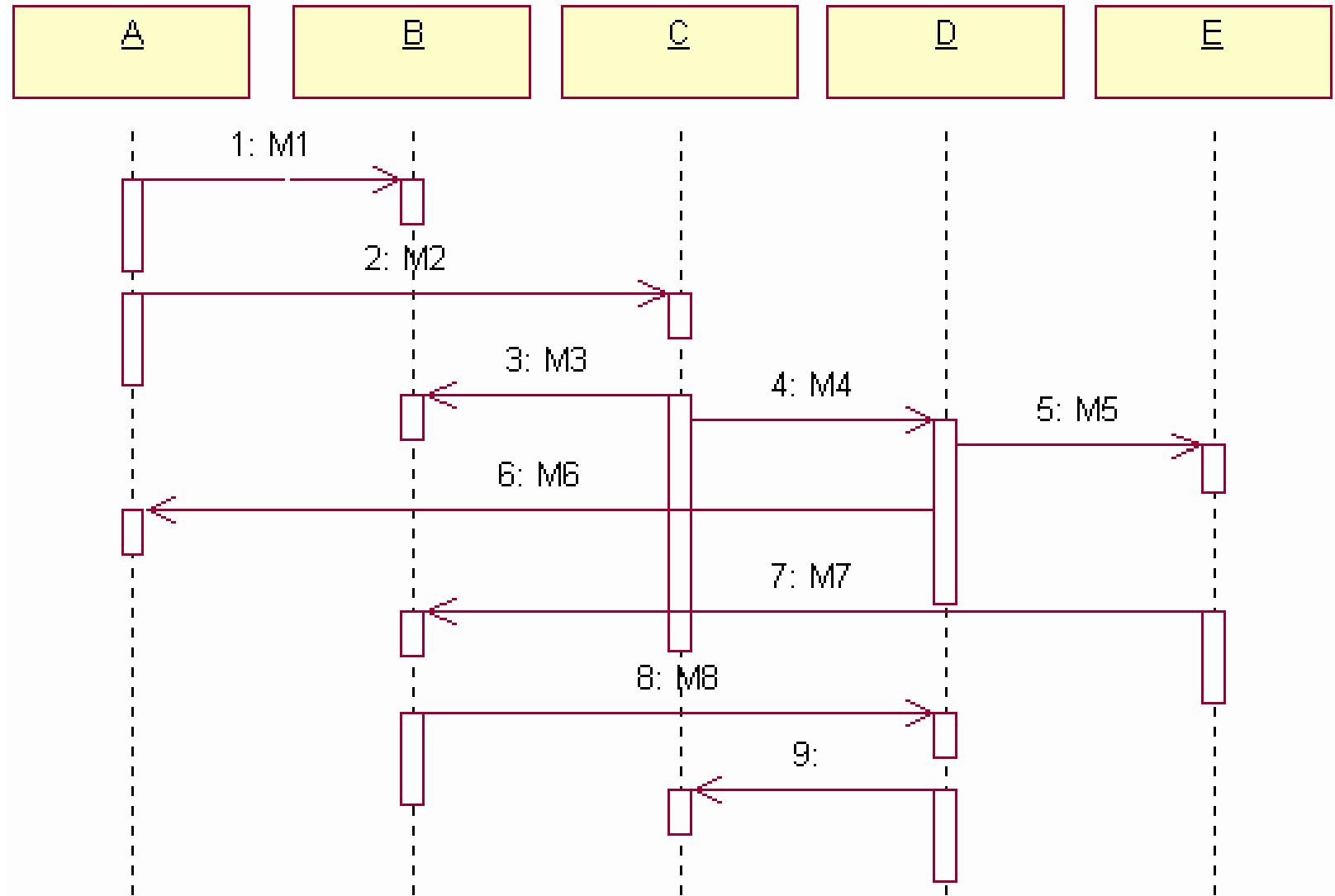
## Messages au sein du système

- **Opérations** du diagramme de classes
- Si message de **objA : ClasseA** vers **objB : ClasseB**, alors opération du message dans ClasseB

## 4.2 - La Phase d'élaboration

### E – Le Diagramme de Séquence ( suite 6 ) : Diagramme de séquence à éviter

**Diagramme  
de séquence  
à éviter**



## **Intérêt de diagramme de séquence**

Les diagrammes de séquences présentent les intérêts suivants :

- permettre de mieux comprendre le fonctionnement du système; modéliser la vie des objets dans le temps et leur chronologie ;
- représenter les interactions, les communications (par messages) entre objets : messages asynchrones (traits horizontaux avec une demi-flèche, messages synchrones (traits horizontaux avec une flèche entière) ;
- d'être très utiles dans la description des cas d'erreur et des cas limites d'utilisation du système;
- d'être une aide précieuse pour documenter les méthodes des classes

## 4.2 - La Phase d'élaboration

### F – Le Diagramme de Collaboration

**Définition :** Les diagrammes de collaboration montrent des interactions entre objets, en insistant plus particulièrement sur la structure spatiale statique qui permet la mise en collaboration d'un groupe d'objets.

Les diagrammes de collaboration expriment à la fois le contexte d'un groupe d'objets (au travers des objets et des liens) et l'interaction entre ces objets (par la représentation de l'envoi de messages).

Les diagrammes de collaboration sont une extension des diagrammes d'objet.

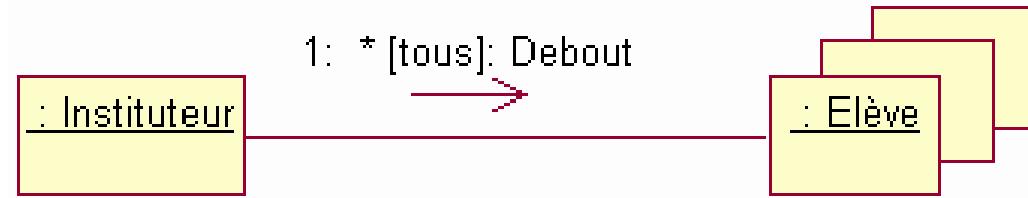
On représente principalement dans les diagrammes de collaboration :

- les structures statiques : les objets;
- les liens entre objets (comme dans les diagrammes de classes);
- les interactions qui sont une suite de messages (structure dynamique) échangés (petites flèches) que l'on va numérotter permettant ainsi de les lire d'une manière chronologique ;

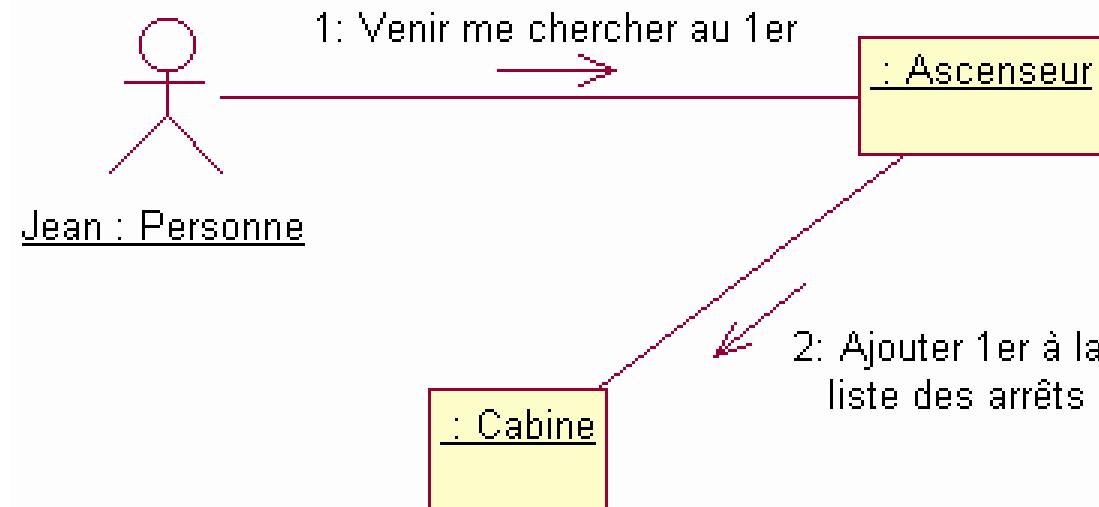
## 4.2 - La Phase d'élaboration

### F – Le Diagramme de Collaboration ( Suite 1 ) : Règles de construction

Itération sur  
un message



Stéréotype  
Acteur



## 4.2 - La Phase d'élaboration

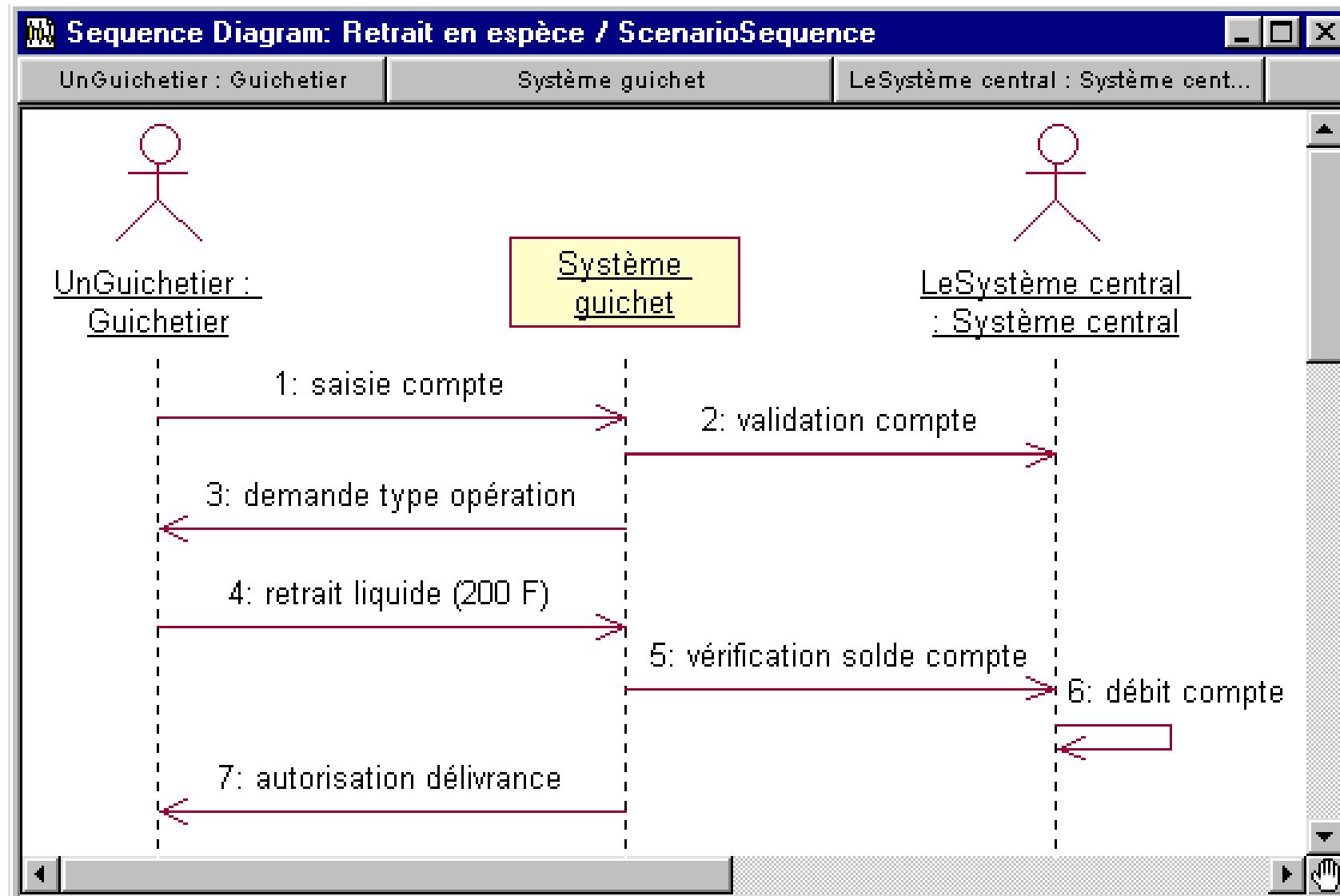
### F – Le Diagramme de Collaboration ( Suite 3 ) : Scénario de Use Case

#### USE CASE : “Retrait en espèces”

- Le guichetier saisit le numéro de compte du client.
- L’application valide le compte auprès du système central.
- L’application demande le type d’opération au guichetier.
- Le guichetier sélectionne un retrait de 200 DH.
- Le système « guichet » interroge le système central pour s’assurer que le compte est suffisamment approvisionné.
- Le système central effectue le débit du compte.
- Le système notifie au guichetier qu’il peut délivrer le montant demandé.

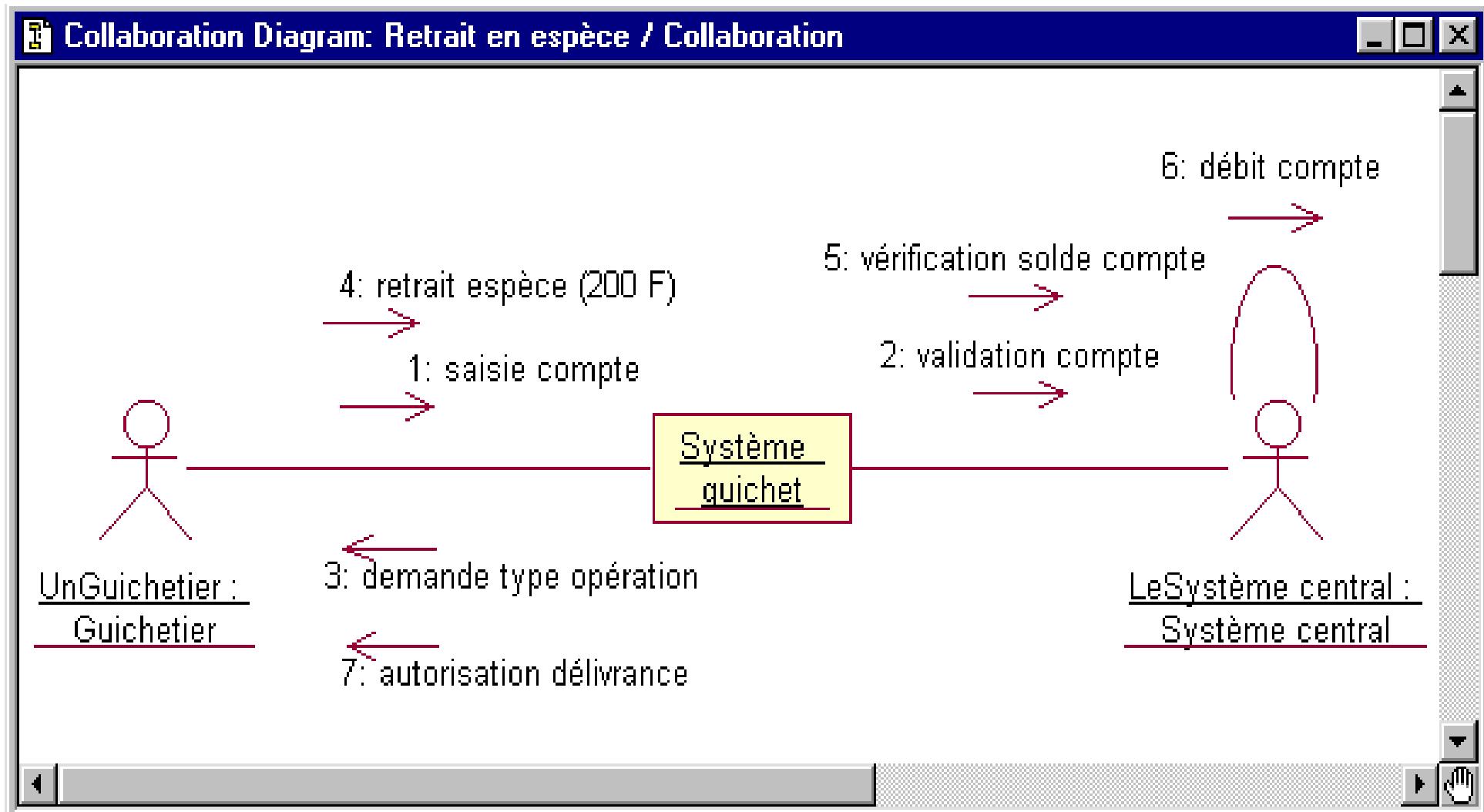
## 4.2 - La Phase d'élaboration

### F – Le Diagramme de Collaboration ( Suite 4 ) : Scénario de Use Case



## 4.2 - La Phase d'élaboration

### F – Le Diagramme de Collaboration ( Suite 5 ) : Scénario de Use Case



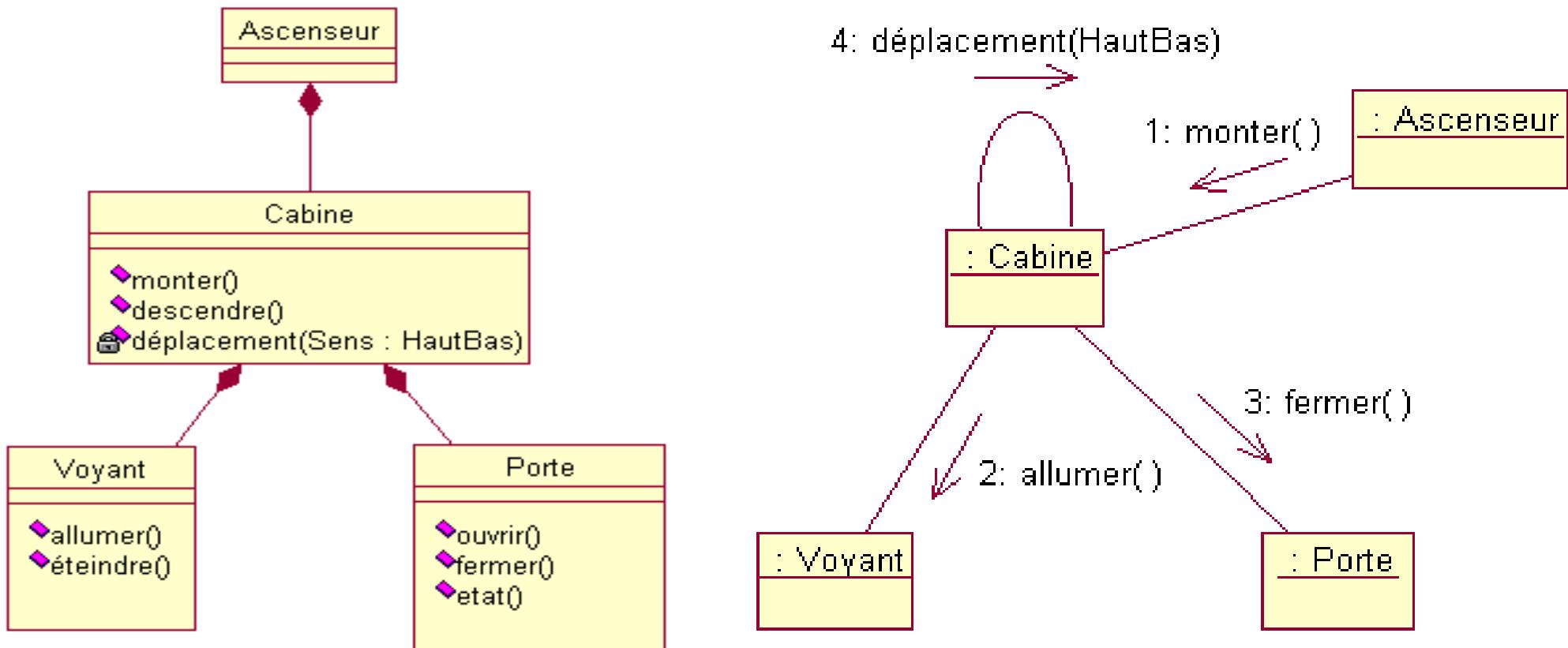
## 4.2 - La Phase d'élaboration

### F – Le Diagramme de Collaboration

**Définition :** Les diagrammes de collaboration montrent des interactions entre objets, en insistant plus particulièrement sur la structure spatiale statique qui permet la mise en collaboration d'un groupe d'objets.

Les diagrammes de collaboration expriment à la fois le contexte d'un groupe d'objets (au travers des objets et des liens) et l'interaction entre ces objets (par la représentation de l'envoi de messages).

Les diagrammes de collaboration sont une extension des diagrammes d'objet.



## 4.2 - La Phase d'élaboration

### F – Le Diagramme de Collaboration

#### **intérêt de diagramme de collaboration**

Les intérêts des diagrammes de collaborations sont :

- de faire coexister les descriptions dynamiques et statiques. Ce qui donne une vision globale du système ;
- de pouvoir décrire des méthodes ou la réalisation de cas d'utilisation (par exemple une suite de messages va permettre de visualiser la réalisation d'une opération) et d'observer leurs effets externes ;
- de représenter un moyen indispensable de vérifier la cohérence globale d'une analyse objet ;
- de visualiser le comportement particulier d'un système à travers un acteur.
- Permettent de comprendre et de décrire le comportement des objets et leurs interactions.

## 4.2 - La Phase d'élaboration

### E– Le Diagramme d’État Transition

#### **Origine**

Deux types de représentations :

Dynamique entre objets avec les deux diagrammes d’interaction :

**diagramme de séquence**

**diagramme de collaboration**

Dynamique interne à un objet avec le **diagramme d’états-transitions**.

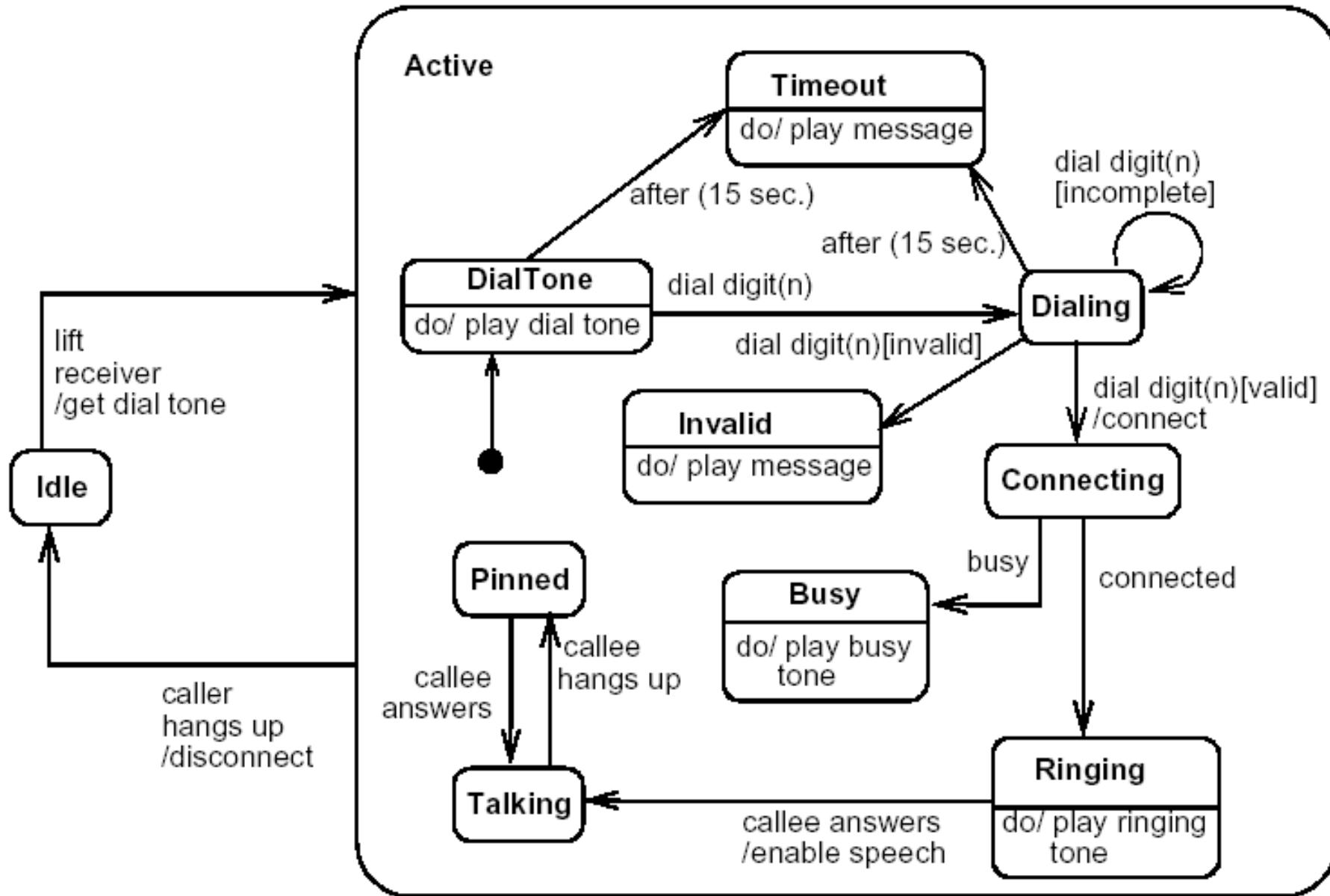
#### **Définition**

un diagramme d'état décrit l'évolution au cours du temps d'un objet (instance d'une classe) en réponse aux interactions avec d'autres objets ou acteurs.

Il décrit le cycle de vie des objets d'une seule classe. On établit un diagramme d'état pour chaque classe qui a un comportement dynamique significatif, ce qui signifie que toutes les classes n'en ont pas besoin

Un diagramme d'état est un graphe orienté comportant des états (nœuds) connectés par des transitions (arc orientés).

## Exemple



## E– Le Diagramme d’État Transition

### État :

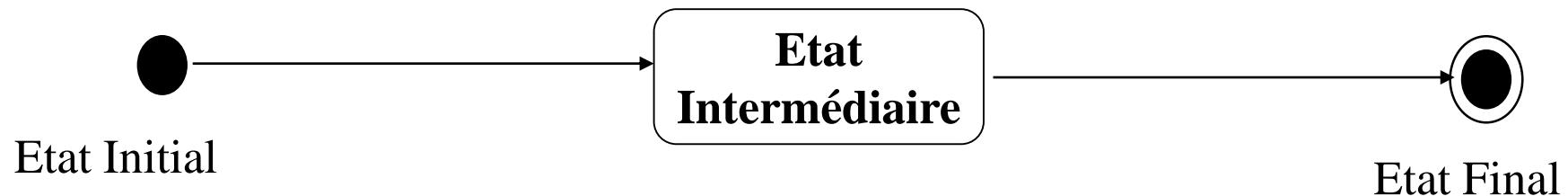
un état spécifie la réponse de l'objet aux événements d'entrée. Il est stable. Il peut être caractérisé par un ensemble de valeurs d'attributs et de liens d'un objet. Chaque objet est à un moment donné dans un état particulier :

-**Etat Initial** : état d'une instance juste après sa création (un seul état initial)

- **Etat Intermédiaire** : un objet est toujours dans un état donné pour un certain temps

-**Etat Final** : état d'une instance juste avant sa destruction (un automate infini peut ne pas avoir d'état final).

### Formalisme :



## E– Le Diagramme d’État Transition

### **Transition - Condition**

#### **Transition :**

Relation entre deux états indiquant qu'un objet dans le premier état va passer dans le deuxième quand un événement particulier apparaîtra, et que certaines conditions seront satisfaites.

#### **Événement:**

un événement est un stimulus pouvant transporter des informations (sous forme de paramètres). Un événement peut être émis par un objet du système ou par un objet externe au système. Il peut également provenir de l'interface du système, par exemple un clic de souris.

#### **Condition :**

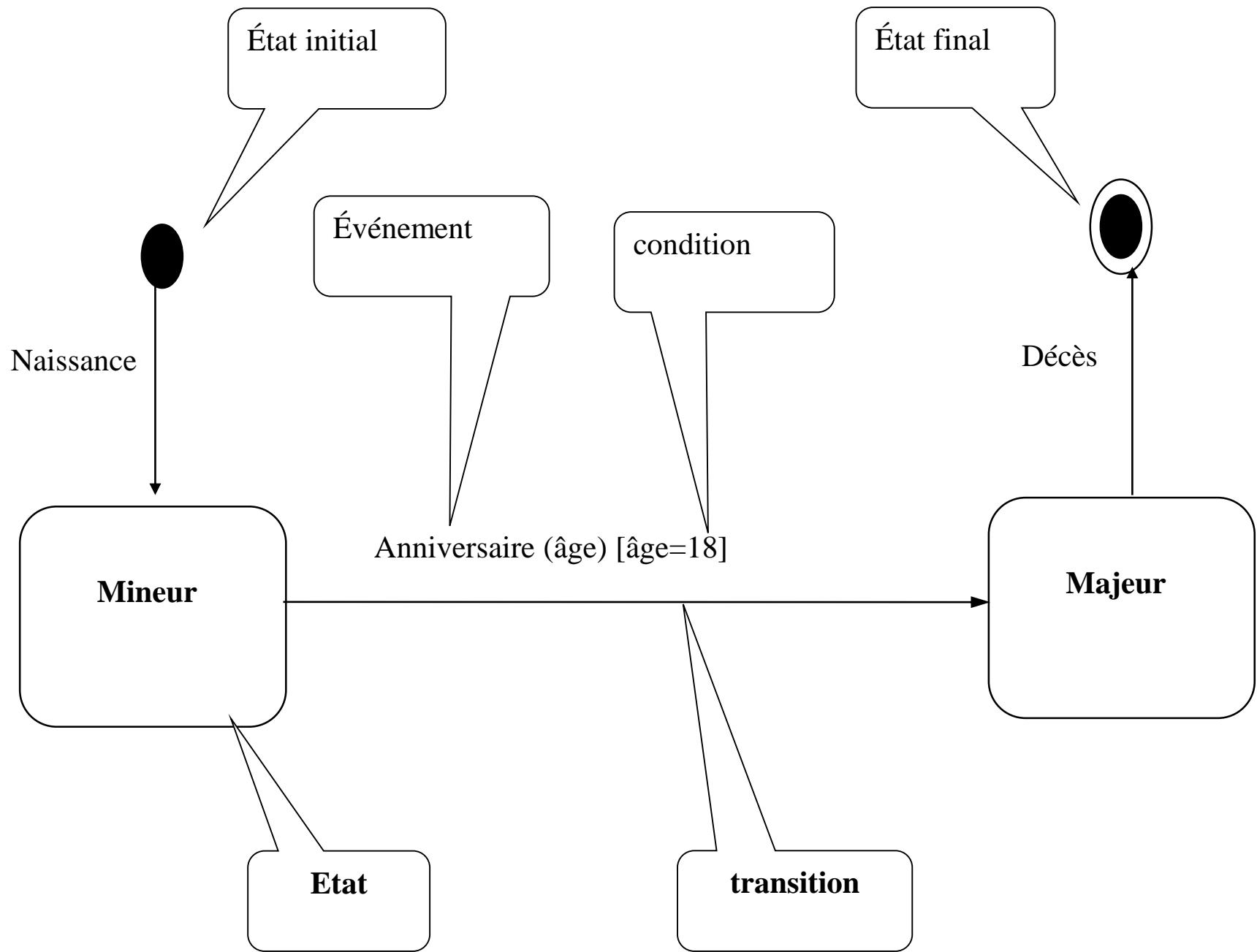
une condition est une expression booléenne, attachée à un événement, qui doit être vraie pour le déclenchement de la transition. Elle est indiquée entre crochets à la suite du nom de l'événement

# Événement

Événement : Fait instantané venant de l'extérieur du système et survenant à un instant donné

Types d'événements :

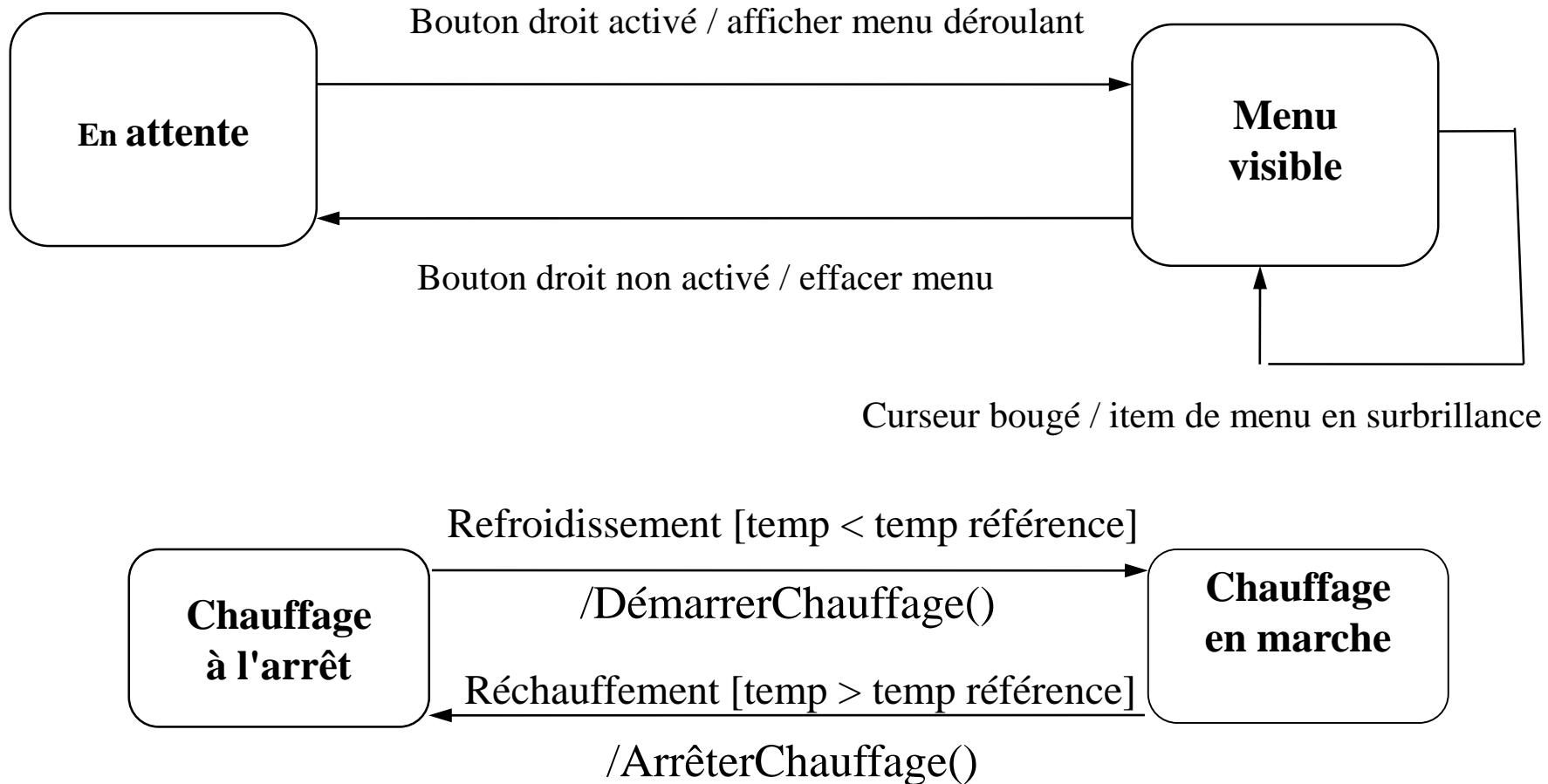
- Signal : réception d'un message asynchrone
- Appel d'une opération (synchrone) : liée aux cas d'utilisation, opération du diagramme de classes...
- Satisfaction d'une condition booléenne : **when(cond)**, évaluée continuellement jusqu'à ce qu'elle soit vraie
- Temps
  - Date relative : **when(date = date)**
  - Date absolue : **after(durée)**



## E– Le Diagramme d’État Transition

### Action :

une action est une opération instantanée (atomique) déclenchée par une transition Elle est associée à un événement. La notation d'une action sur une transition est précédée d'une barre oblique ("/").



# Action

Action : Réaction du système à un événement

Caractéristiques : atomique, instantanée, non interruptible

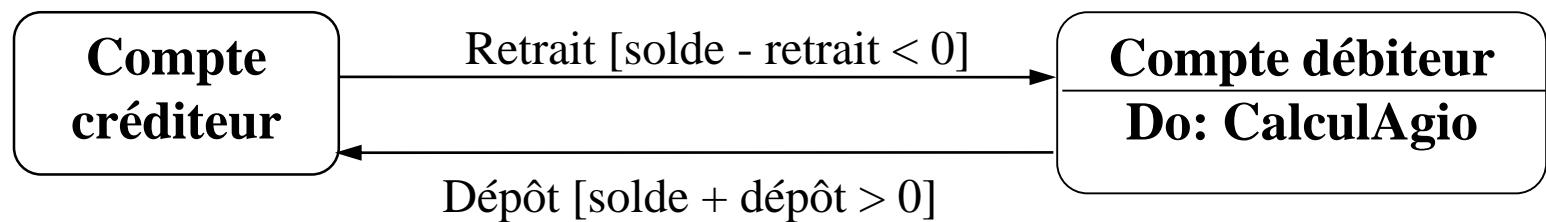
Exemples d'actions (syntaxe laissée libre) :

- affectation
- envoi d'un signal
- appel d'une opération
- création ou destruction d'un objet

## E– Le Diagramme d’État Transition

### Activité :

une activité est une opération qui dure un certain temps. Elle est associée à un état. Elle peut être continue ou finie (se termine d’elle même). Notation : "Do : activité".



### En entrée :

L'action est exécutée chaque fois que l'on entre dans l'état.

Notation : **entry** : action

### En sortie :

L'action est exécutée chaque fois que l'on quitte l'état.

Notation : **exit** : action

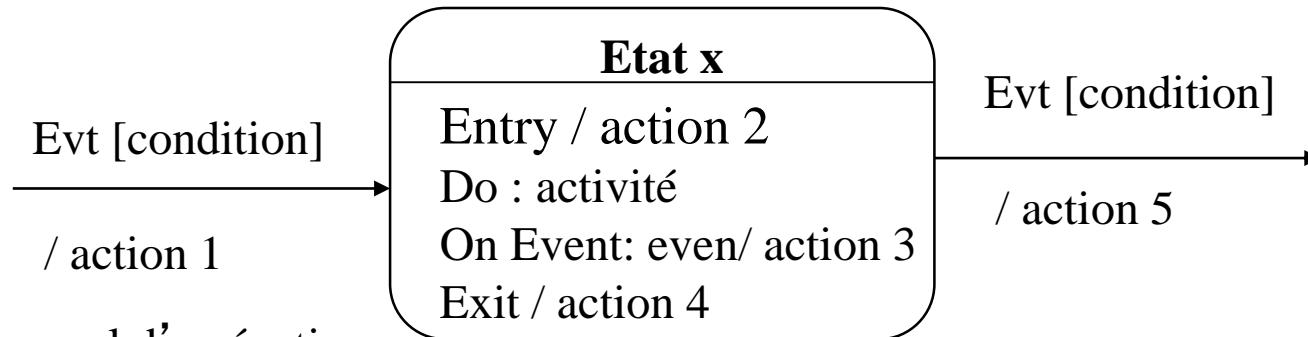
### Action interne :

L'action ou opération exécutée dans un état seulement si un événement survient.

Notation : **On Event** : action

## E– Le Diagramme d’État Transition

### Notation complète et ordonnancement :



### Ordre temporel d’ exécution :

#### En entrée :

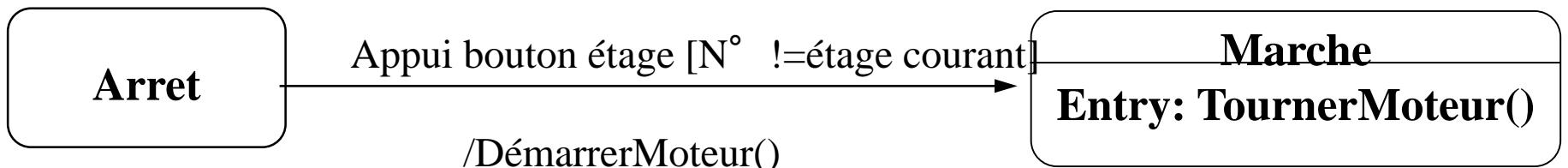
- Action sur la transition d’ entrée (action 1)
- Action d’ entrée dans l’ état (action 2)
- Démarrage de l’ activité

#### Dans l’ état :

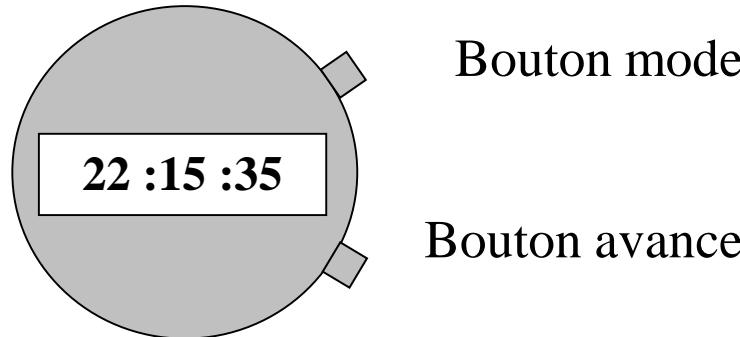
- Interruption de l’ activité
- Exécution de l’ action interne (action 3)
- Reprise de l’ activité

#### En sortie :

- Arrêt de l’ activité
- Action de sortie de l’ état (action 4)
- Action sur la transition de sortie (action 5)



**Exemple :** Fonctionnement d'un réveil à affichage digital



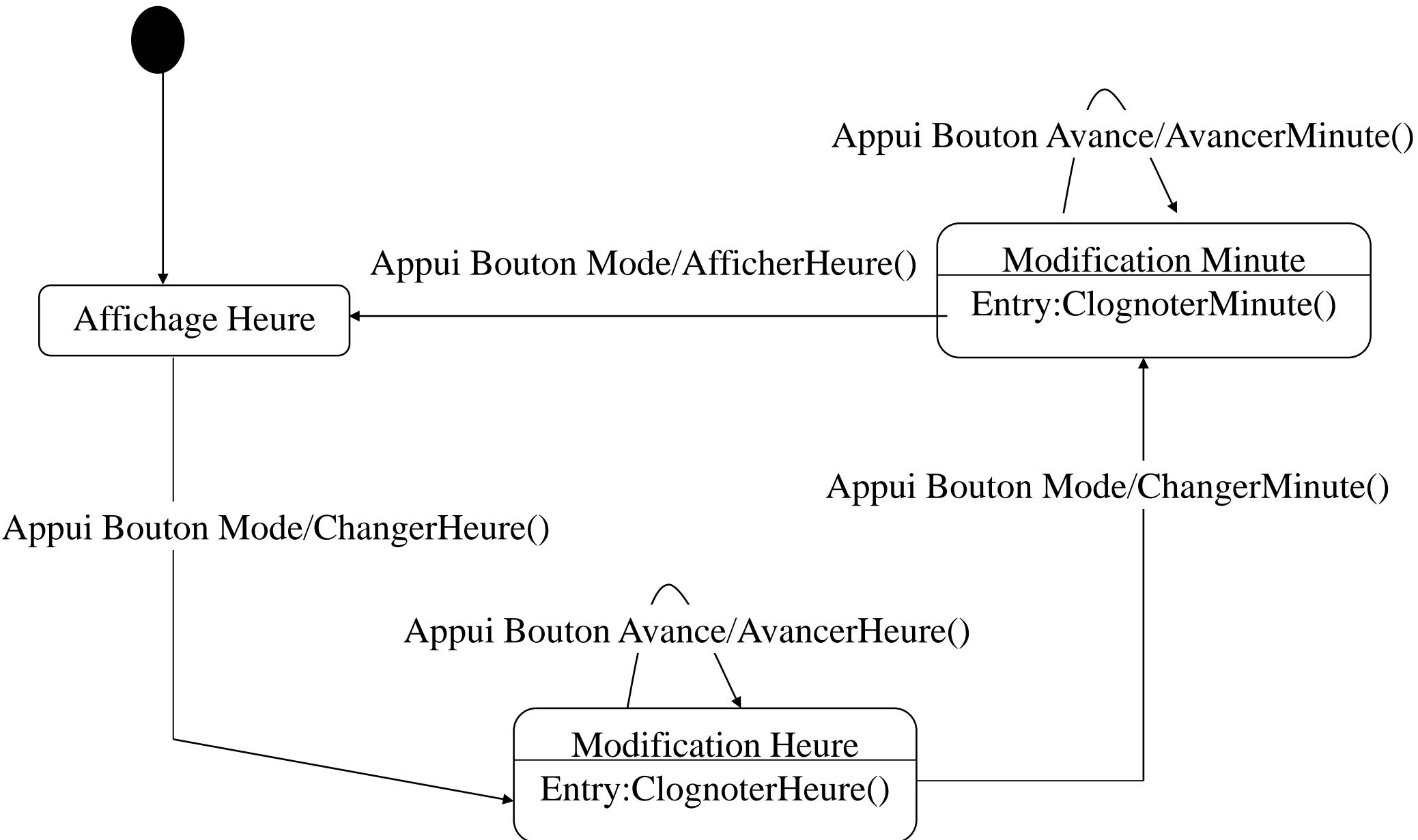
En fonctionnement normal, la montre affiche les heures, les minutes et les secondes qui défilent.

Une pression sur le bouton mode sélectionne le chiffre des heures qui se met à clignoter. A ce moment une pression sur le bouton avance incrémente le compteur d'une heure.

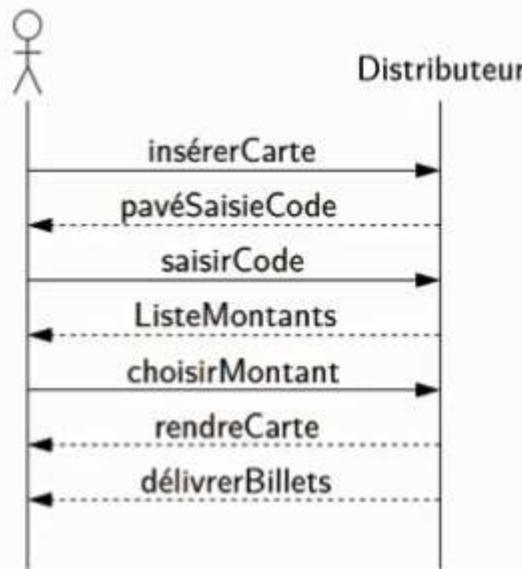
Une nouvelle pression sur le bouton mode sélectionne le chiffre des minutes qui se met à clignoter. A ce moment une pression sur le bouton avance incrémente le compteur d'une minute.

Une nouvelle pression sur le bouton mode revient à l'affichage normal.

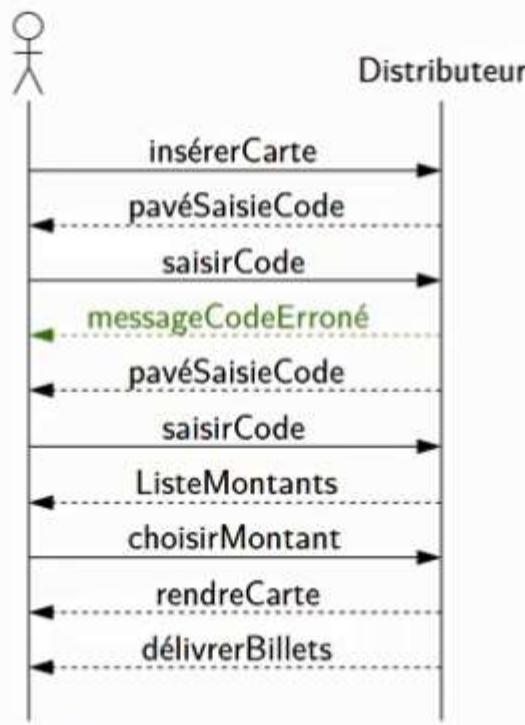
**Dessiner le diagramme état-transition de cette montre.**



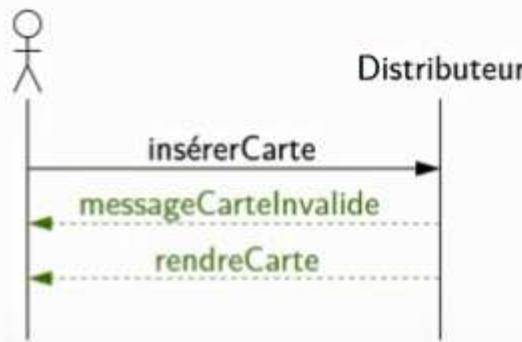
# Exemple : distributeur automatique



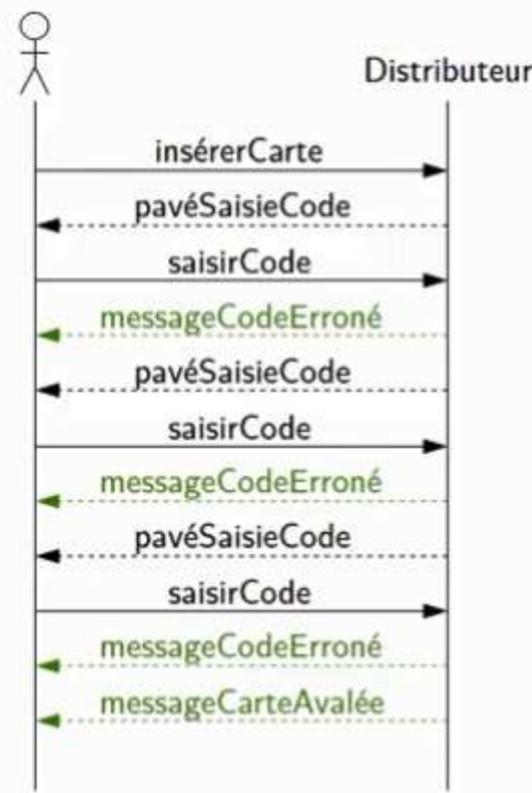
Scénario principal



Une erreur de code



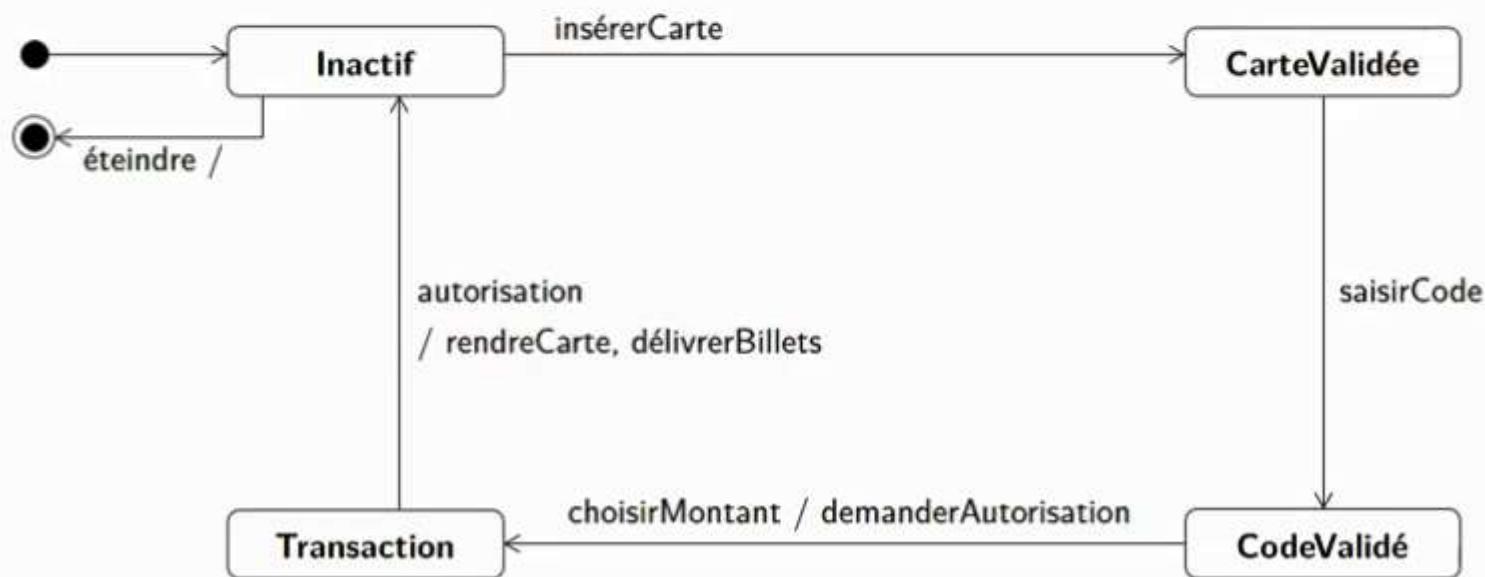
Carte invalide

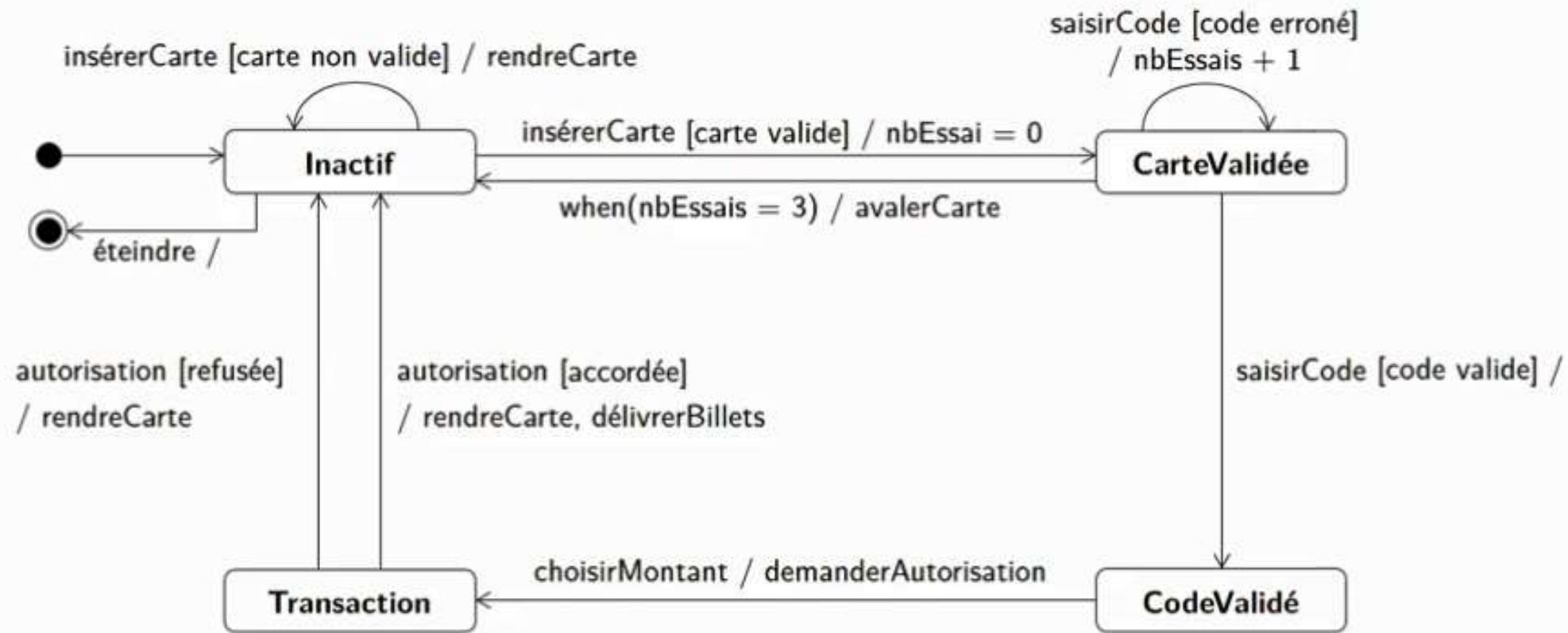


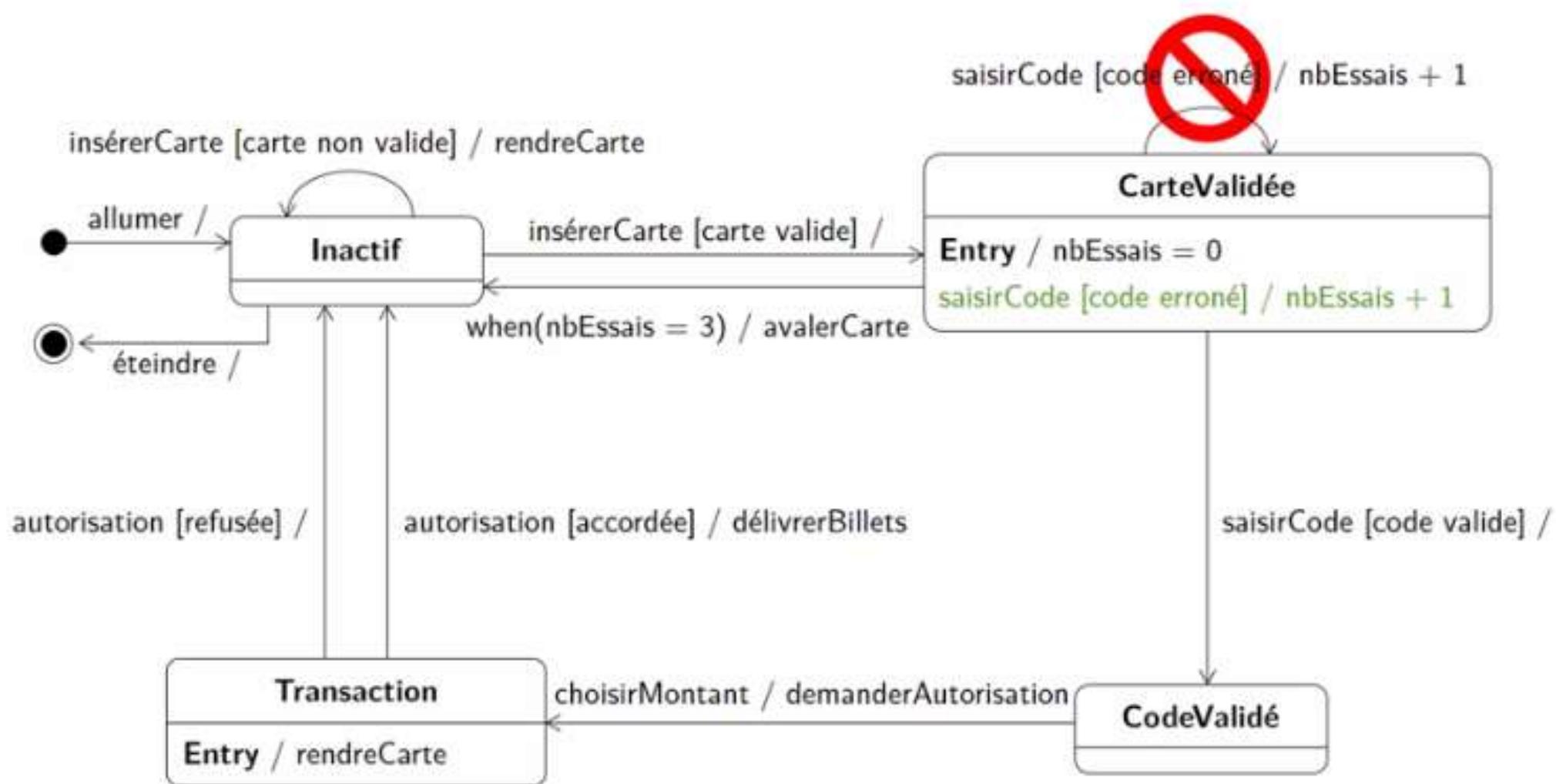
Trois erreurs de code

# Diagramme états-transitions correspondant

Scénario nominal







# Utilisation des diagrammes états-transitions

En phase d'analyse :

- Description de la dynamique du système vu de l'extérieur
- Synthèse des scénarios liés aux cas d'utilisation
- Événements = action des acteurs

En phase de conception :

- Description de la dynamique d'un objet particulier
- Événements = appels d'opérations

# Diagramme états-transitions d'un objet

Spécification du conteneur :

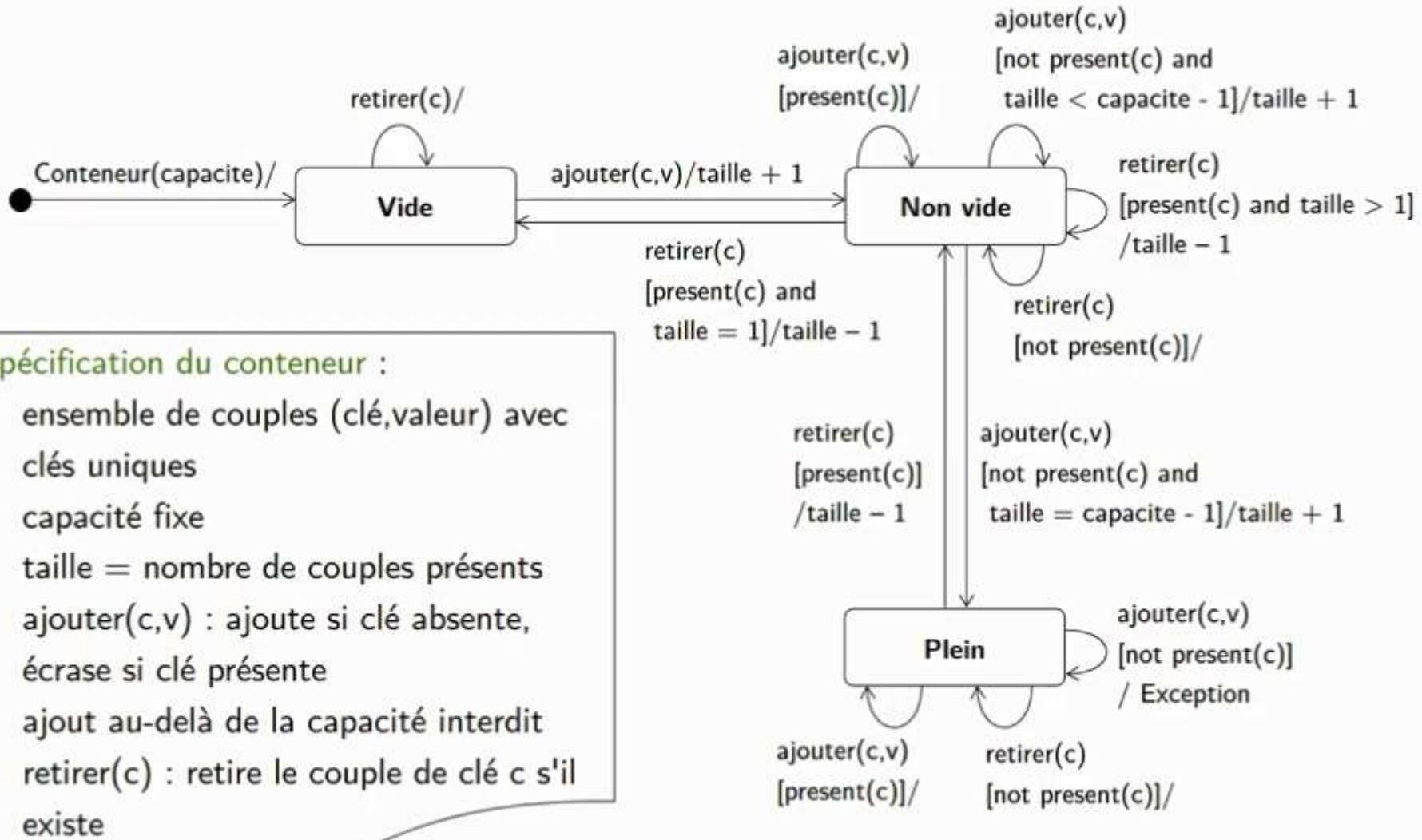
- ensemble de couples (clé,valeur) avec clés uniques
- capacité fixe
- taille = nombre de couples présents
- ajouter(c,v) : ajoute si clé absente, écrase si clé présente
- ajout au-delà de la capacité interdit
- retirer(c) : retire le couple de clé c s'il existe

**Conteneur**

capacite : int  
taille : int

Conteneur(cap : int)  
ajouter(c:Object,v:Object)  
retirer(c:Object)  
present(c:Object) : boolean

# Diagramme états-transitions d'un objet



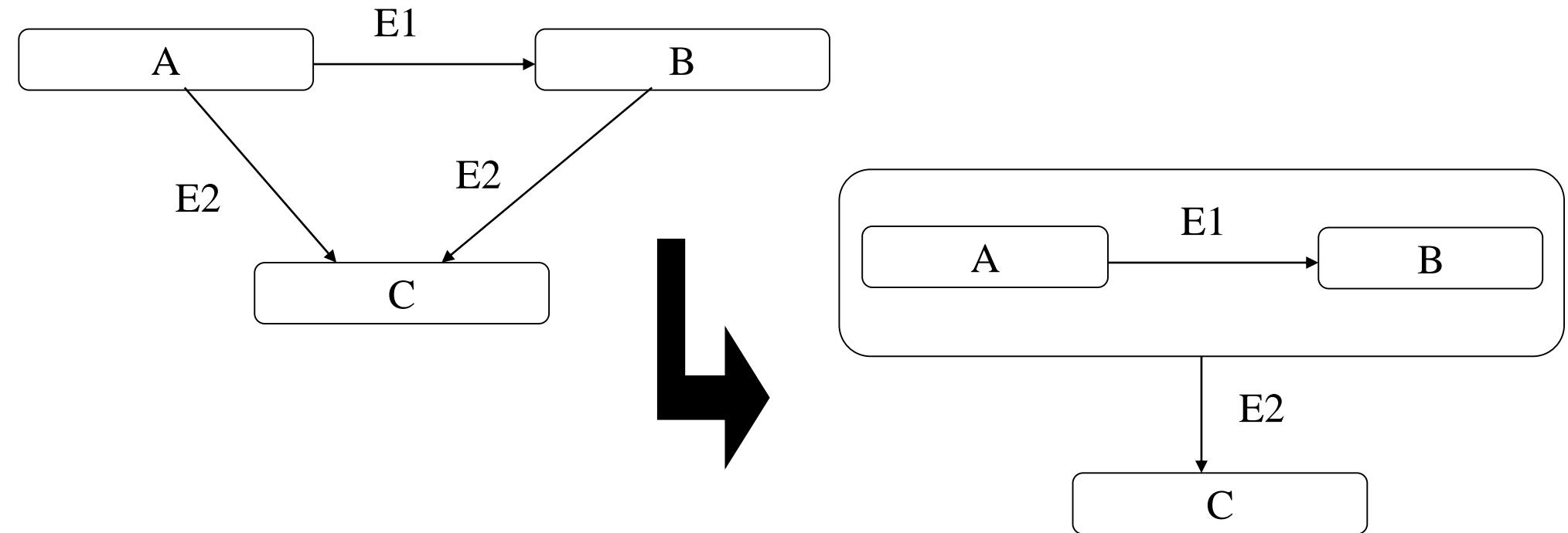
## E– Le Diagramme d'État Transition

Généralisation d'états:

Dans le cas d'un comportement dynamique **complexe**, les diagrammes d'états sur un niveau deviennent rapidement **illisibles**

Pour éviter ce problème, il est nécessaire de structurer les diagrammes d'états en :

- **Super-états** : états généraux
- **Sous-états** : héritent des caractéristiques des états généraux

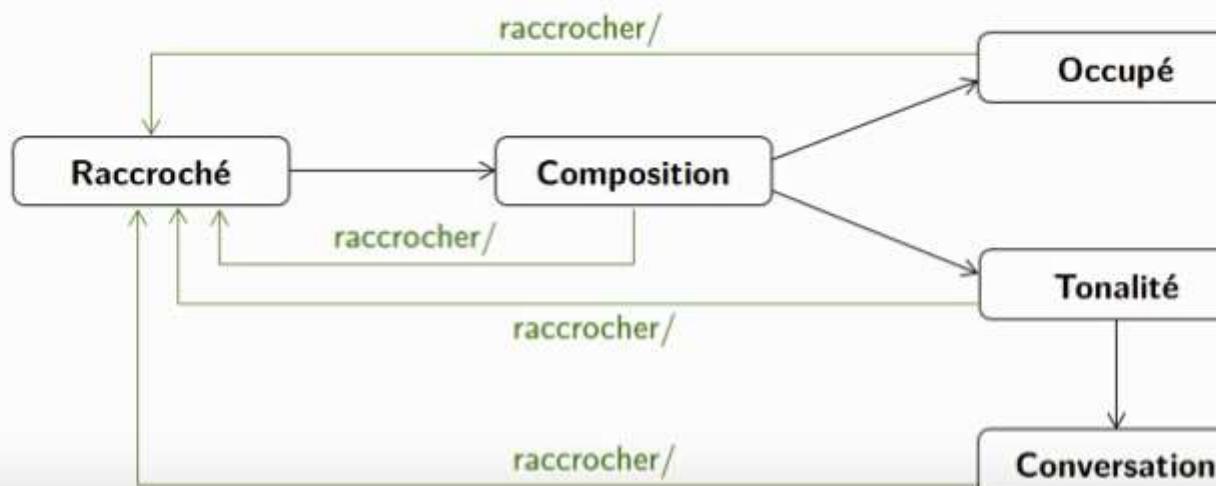


# États composites

État composite : État regroupant un ensemble d'états

Objectifs :

- Hiérarchiser les états
- Structurer les comportements complexes
- Factoriser les actions

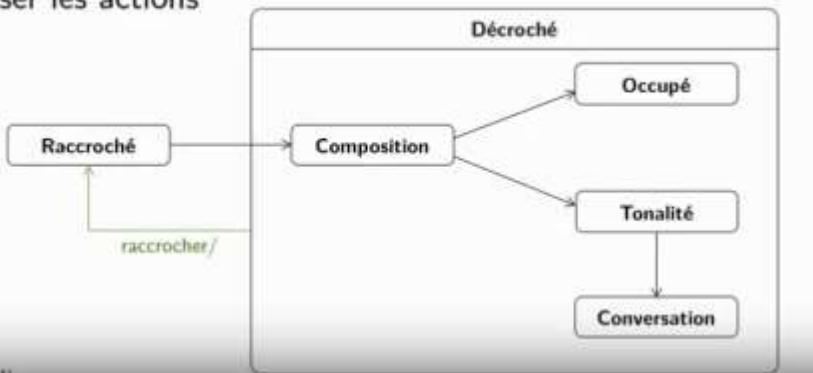


# États composites

État composite : État regroupant un ensemble d'états

Objectifs :

- Hiérarchiser les états
- Structurer les comportements complexes
- Factoriser les actions

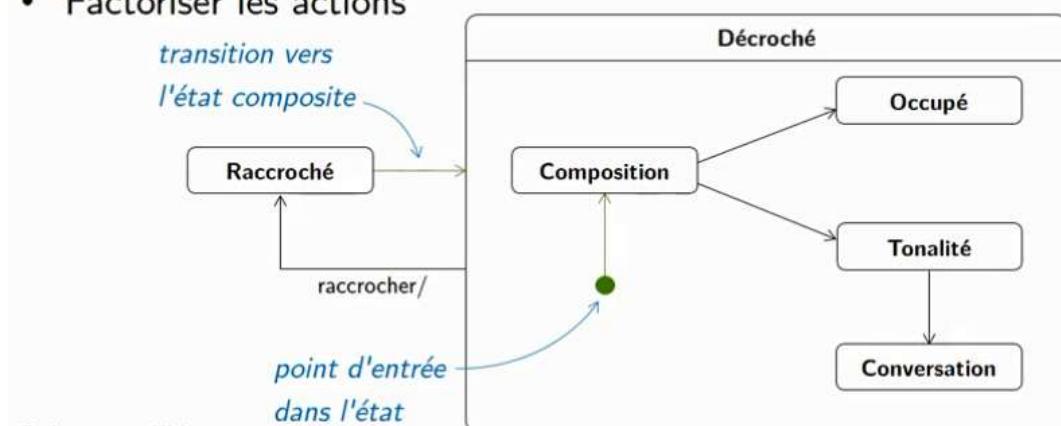


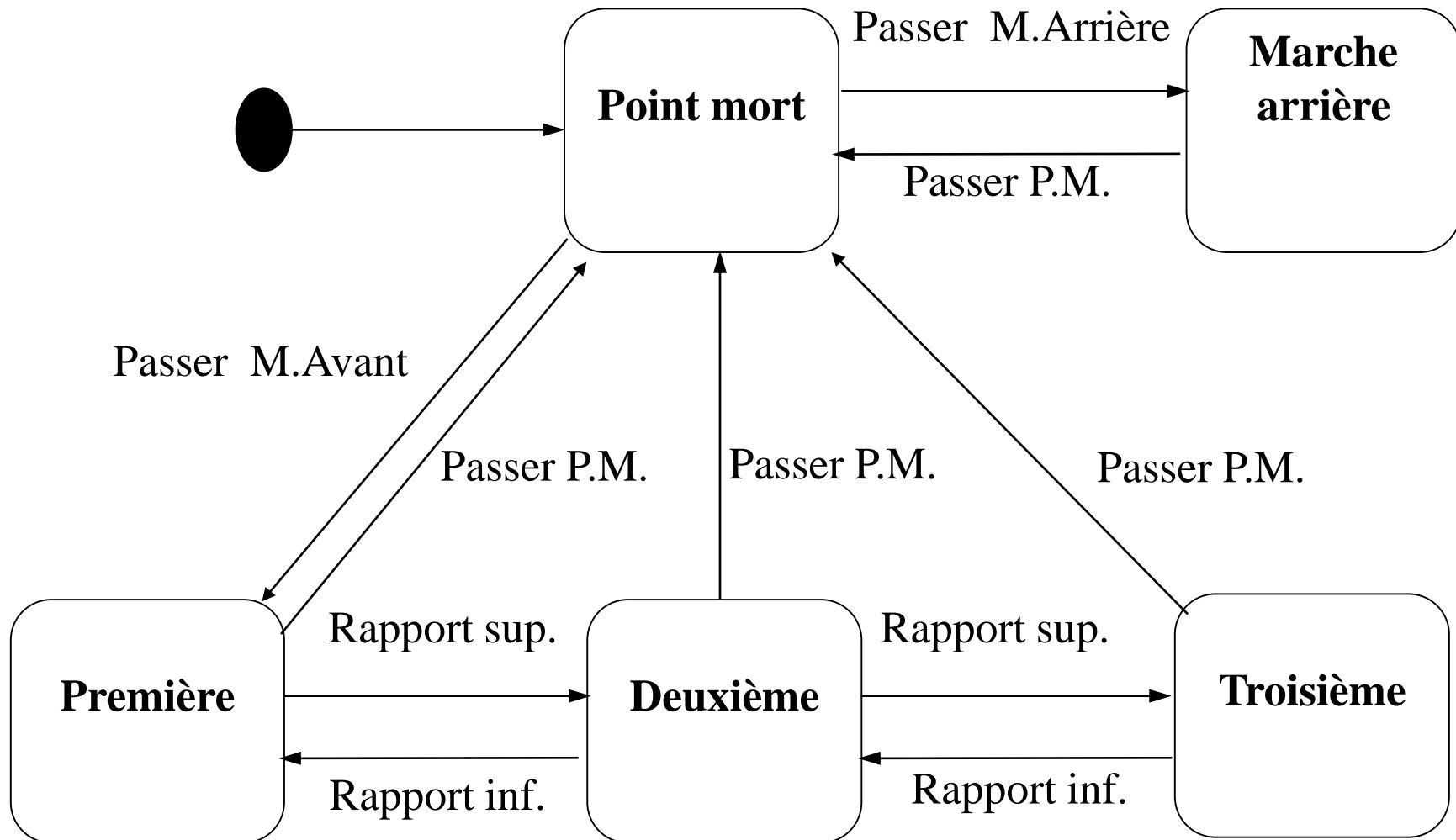
# États composites

État composite : État regroupant un ensemble d'états

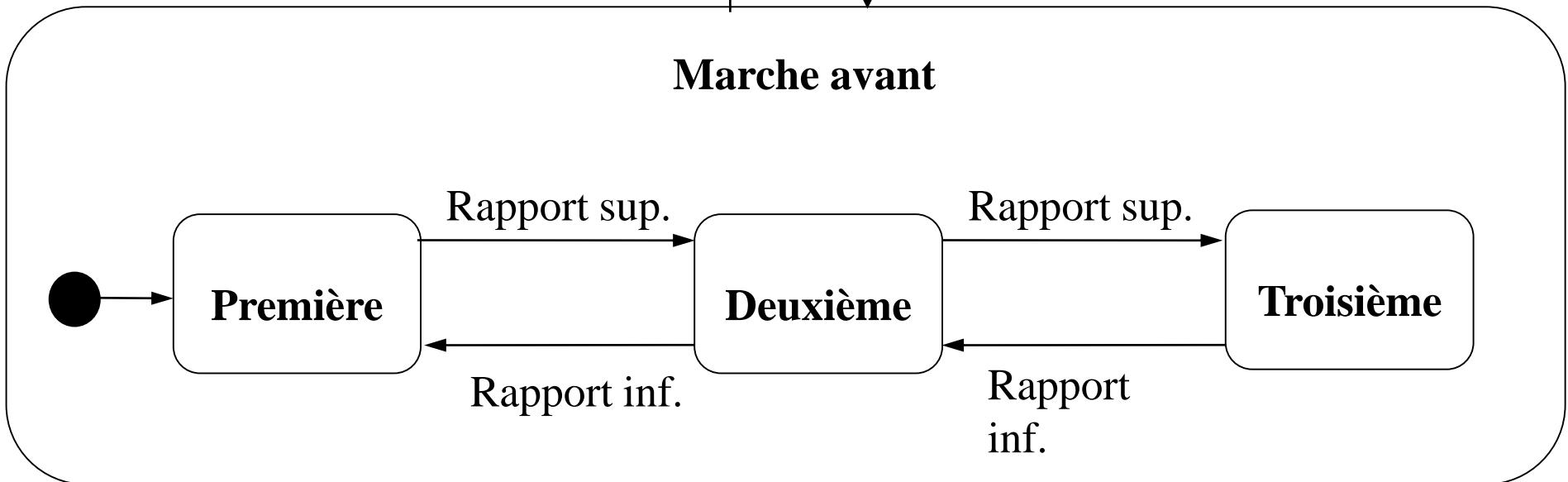
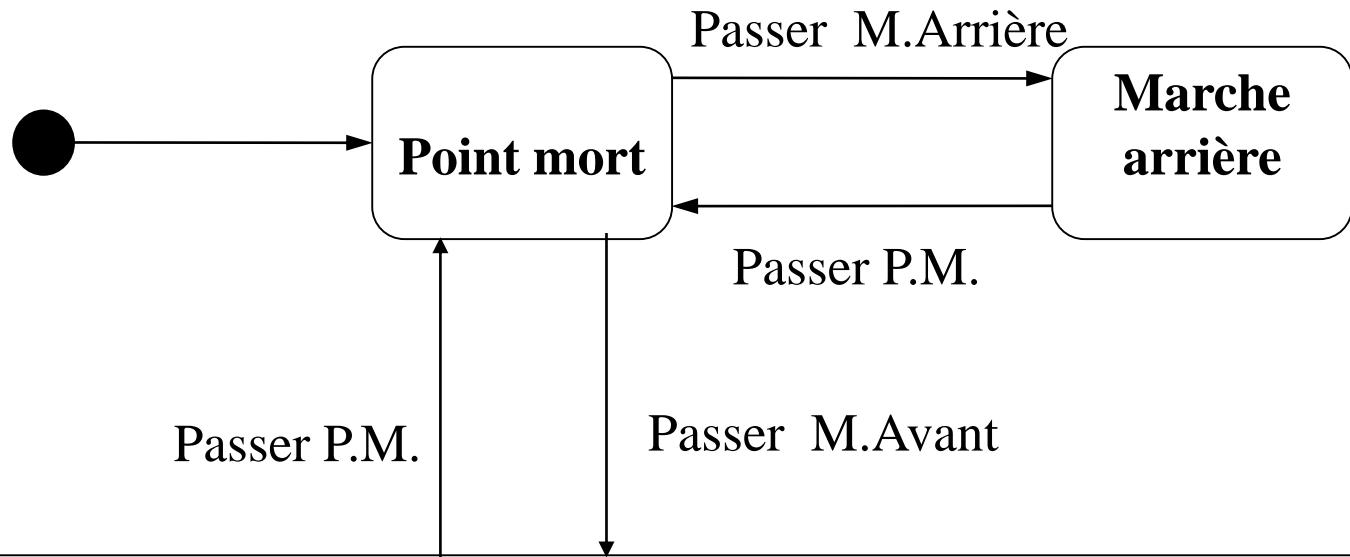
Objectifs :

- Hiérarchiser les états
- Structurer les comportements complexes
- Factoriser les actions





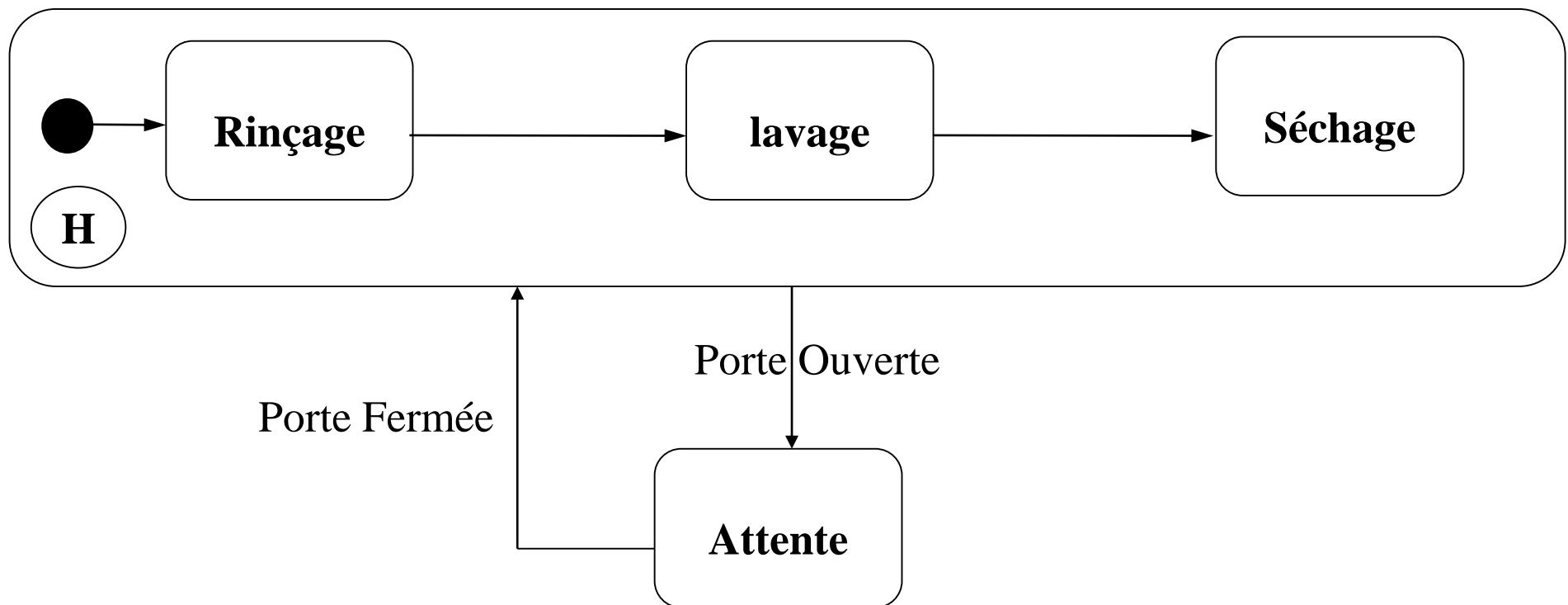
**Objectif : réduire la complexité**



## E– Le Diagramme d'État Transition

### Notion d'historique :

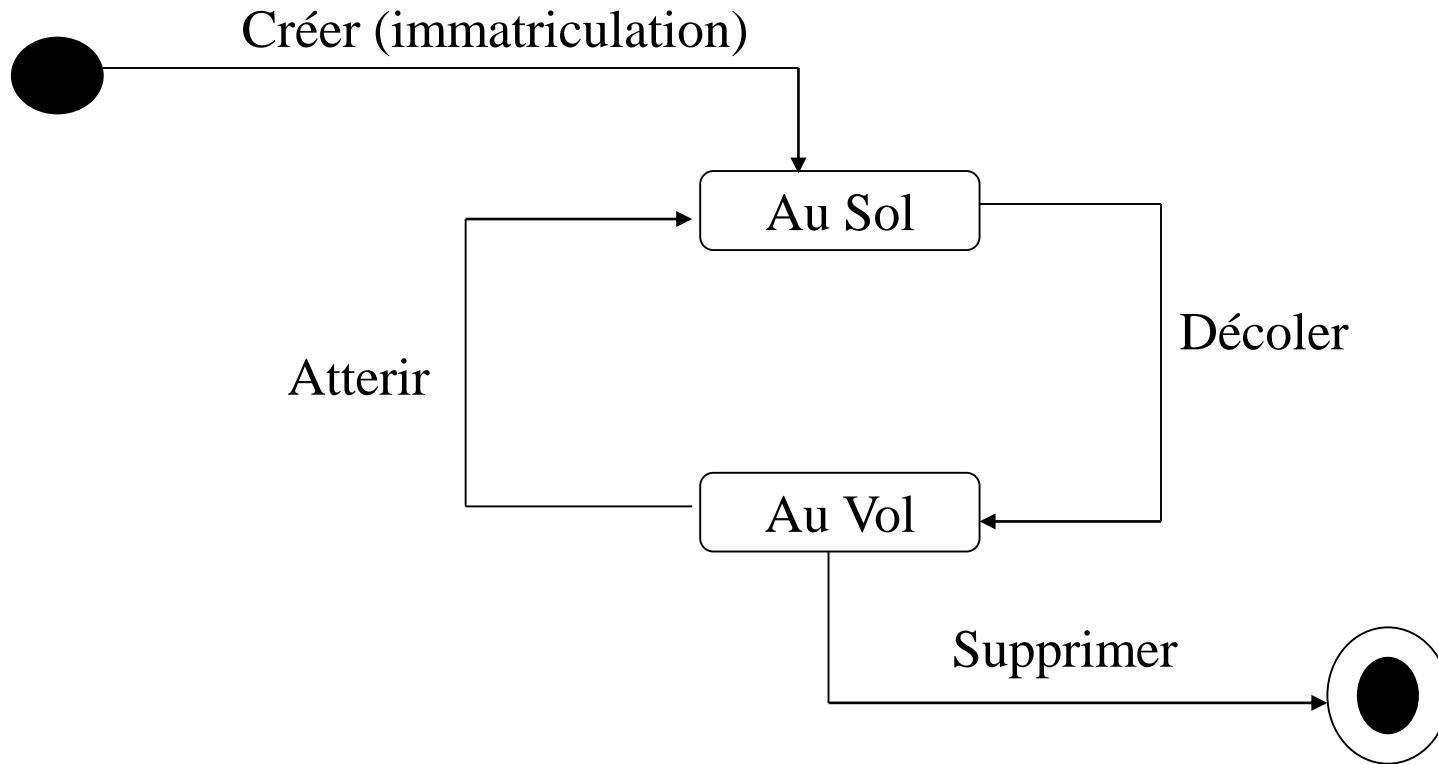
- Par défaut, un automate n'a **pas de mémoire**
- la notation **H** offre un **mécanisme pour mémoriser** le dernier sous état dans lequel l'automate se trouve avant une transition afin de pouvoir retrouver cet état si nécessaire.
- **Exemple** : Cycle de lavage d'un lave-vaisselle



## E– Le Diagramme d’État Transition

### Création et Destruction

La création d'un objet se représente par l'envoi d'un événement de création à la classe de l'objet



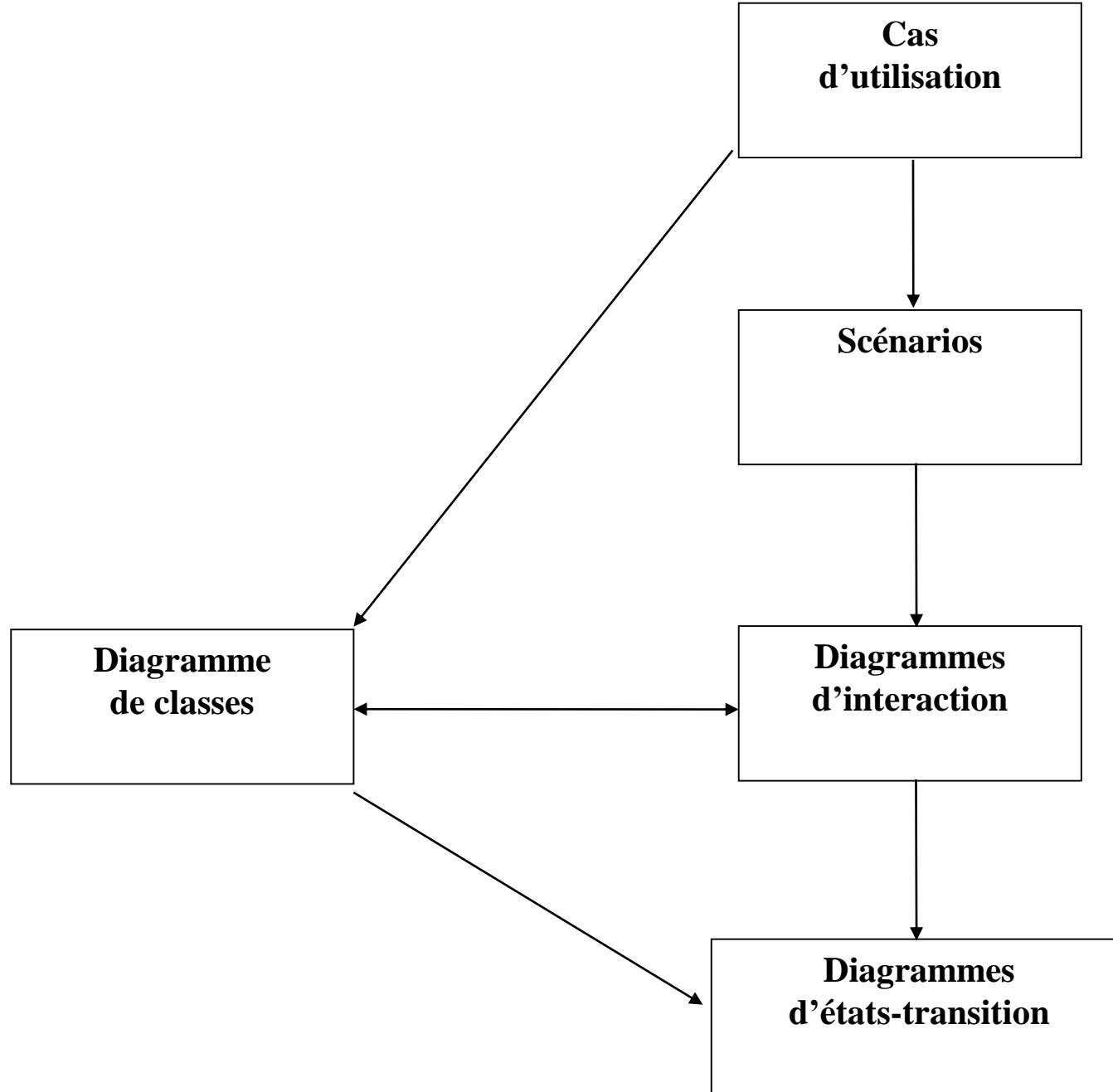
La destruction est effective lorsque le flot de contrôle de l'automate atteint un état final non emboîté

## E– Le Diagramme d’État Transition

### **Intérêts des diagrammes d’états-transitions**

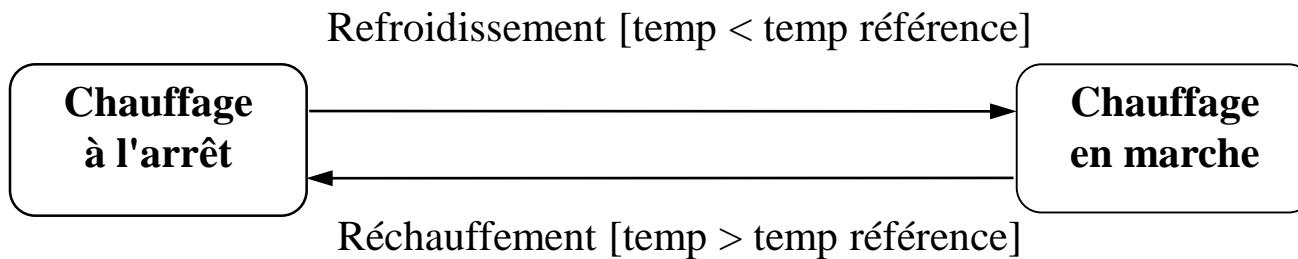
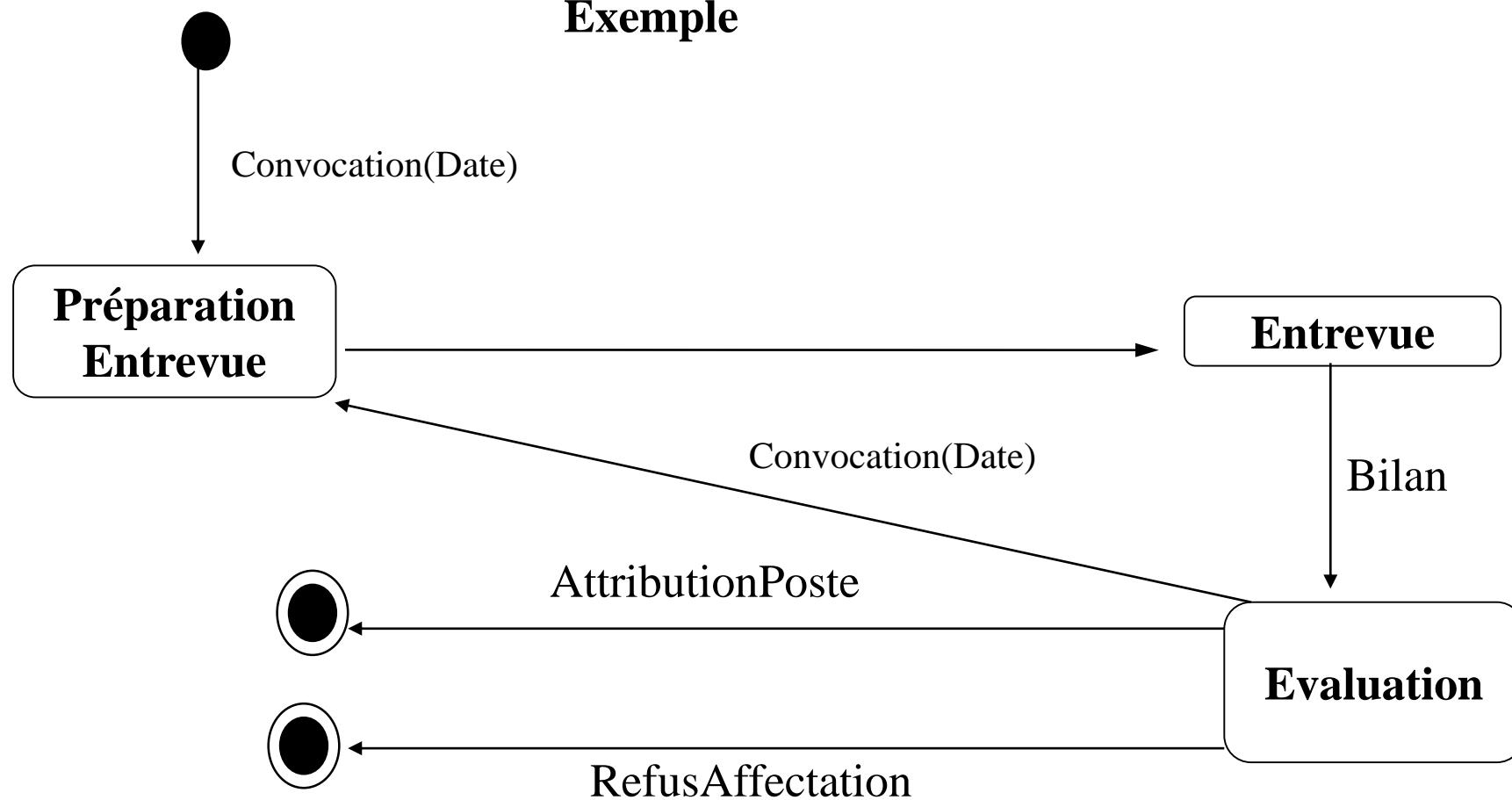
Les intérêts de la modélisation par les diagrammes d’états-transitions sont de :

- Donner vie aux objets (s’ils s’y prêtent) représentés jusqu’à présent de manière statique comme des occurrences de classes qu’on peut généraliser par les classes ;
- Mieux visualiser le système en diminuant sa complexité (parce que l’on va détailler);
- Tenir compte des états lors de l’implémentation (en effet la traduction des états peut être faite simplement, la plupart des langages le permettent);
- Représenter un aspect du modèle dynamique, l’autre étant illustré les diagrammes de collaborations et les diagrammes de séquences ;
- Pouvoir décrire les changements d’états des automates



Etude de cas : une agence de recrutement effectue des compagnes de recrutement pour le compte de nombreuses entreprise (clients de l'agence) désirant embaucher du personnel qualifié sur postes avec des profils bien précis. Les candidats sont d'abord sélectionnés par l'agence puis convoqués pour déposer leurs pièces de dossier. Ils sont ensuite invités à se présenter à un entretien personnel où leurs compétences et aptitudes sont examinées par une commission constituée d'un ensemble d'examineurs. Lorsque tous les candidats sont examinés dans une compagnie, la même commission d'examineurs se réunit à une date bien précise pour évaluer l'ensemble des dossiers des candidats. A l'issue de cette évaluation, les candidats se voient attribuer ou refuser le poste auquel ils ont aspiré au niveau de leur candidature.

# Exemple

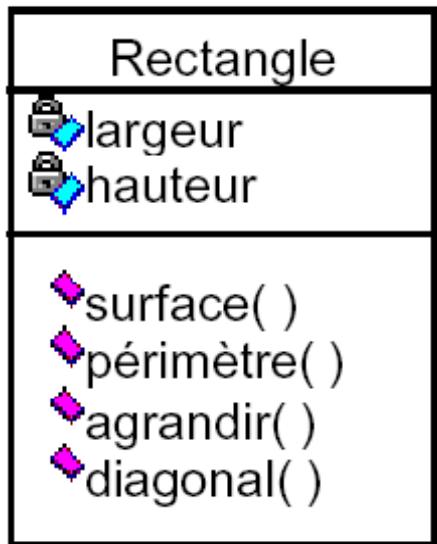


## 4.3 - La Phase d'implémentation ( codage )

### C – Transposition d'un modèle UML dans un contexte relationnel

#### • Classe avec attributs et méthodes

La génération de code SQL tient compte du comportement statique ( attributs ) mais pas du comportement dynamique ( méthodes )



Comment identifier les instances ?

⇒ Ajouter une clé primaire artificielle si une clé naturelle n'existe pas

```
Create Table RECTANGLE (
    Id_Rectangle Integer Primary Key ,
    Largeur Number ,
    Hauteur Number )
```

Que faire avec les méthodes ?

⇒ Plusieurs solutions suivants les besoins ...

Solution 1: Créer de nouveaux attributs calculés et les mémoriser ( surface , périmètre,...)

Solution 2 : Utiliser des Vues ( SELECT ) sans mémorisation pour déterminer la surface , le périmètre , etc...

Solution 3 : Utiliser des mises à jour ( UPDATE ) pour les autres méthodes ( exemple : modifier la valeur de la largeur et de la longueur avec la méthode Agrandir )

## C – Transposition d'un modèle UML dans un contexte relationnel ( Suite )

### • Classe avec attributs et méthodes

Solution 1: Créer de nouveaux attributs calculés et les mémoriser ( surface , périmètre,...)

**Create Table RECTANGLE (**

**Id\_Rectangle Integer Primary Key , Largeur Number , Hauteur Number ,  
Surface Number , Perimetre Number )**

Solution 2: Utiliser des Vues sans mémorisation pour déterminer la surface , le périmètre , etc...

**Create vue V\_Rectangle As**

**Select Id\_Rectangle , Largeur , Hauteur ,  
Largeur \* Hauteur Surface ,  
2 \* ( Largeur + Hauteur ) Perimetre )**

**From Rectangle**

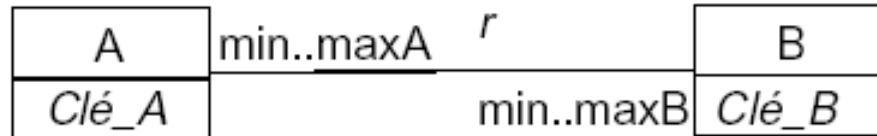
Solution 3: Utiliser des mises à jour pour les autres méthodes

**Update Rectangle**

**Set Largeur = 2 \* Largeur , Hauteur = 2 \* Hauteur  
Where Id\_Rectangle = ...**

## C – Transposition d'un modèle UML dans un contexte relationnel ( Suite )

### • Association binaire générale



Objectif: Mémoriser l'association dans une base de données

La solution dépend de l'association “ r ” ( des multiplicités maxA et maxB )

MaxA MaxB	1	>1
1	- si la clé de A = la clé de B, ne rien faire - sinon choisir une des autres solutions	Ajouter la clé de A dans la relation de B comme attribut
>1	Ajouter la clé de A dans la relation de B comme attribut	Créer une relation r ayant comme attribut la clé de A et la clé de B

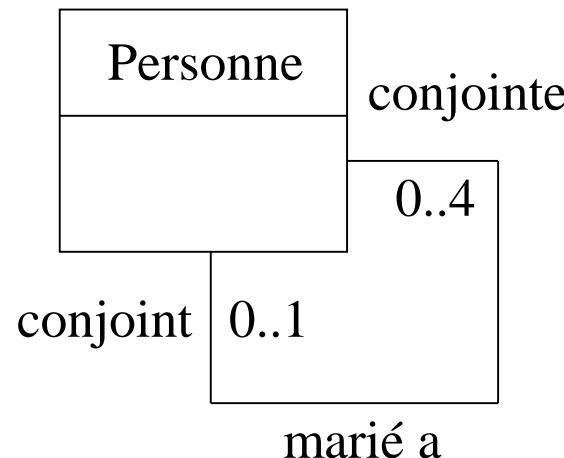
## C – Transposition d'un modèle UML dans un contexte relationnel ( Suite )

### Associations binaires hiérarchiques



**Create Table Region (CodeRegion varchar(4) not null primary key,  
Region varchar(30),....)**

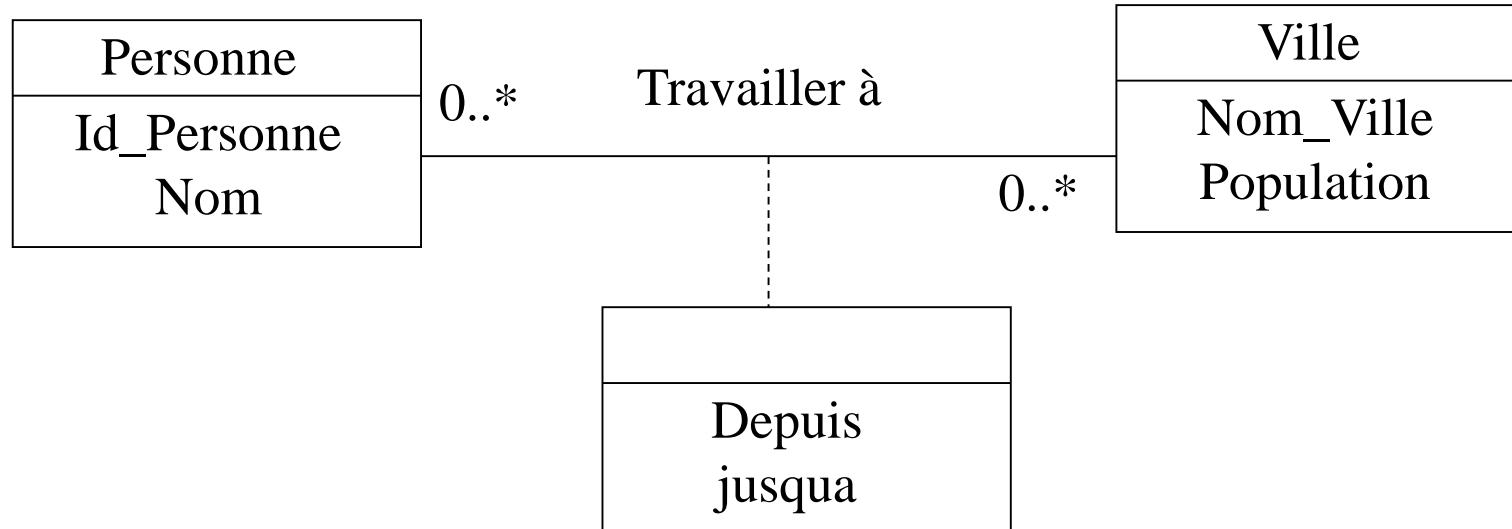
**Create Table Ville (NomVille varchar(20) not null primary key,  
CodeRegion varchar(4) References Region (CodeRegion),  
Population int,...)**



**Create table personne(Id-personne int not null primary key,  
nom varchar(20),datenaissance date,  
id\_personne\_conjoint references personne (id\_personne,...))**

## C – Transposition d'un modèle UML dans un contexte relationnel ( Suite )

### Associations binaires multivaluées



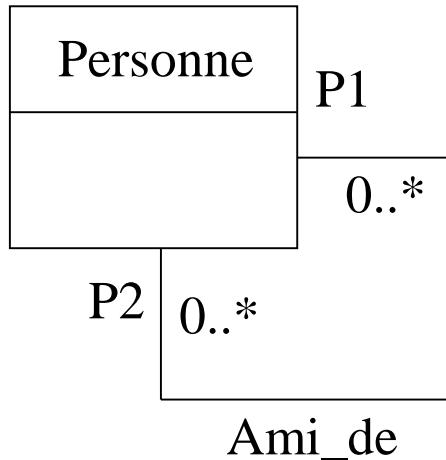
Create Table Ville (NomVille varchar(20) not null primary key, ,Population int,...)

Create table personne(Id-personne int not null primary key, nom varchar(20), Prenom varchar(20), datenaissance date, ...)

Create table travailler (id\_personne int not null reffences personne (id\_personne), nomville varchar(20) refferences ville(nomville), depuis date, jesqua date, primary key (id\_personne,nomville))

## C – Transposition d'un modèle UML dans un contexte relationnel ( Suite )

### Associations multivaluées

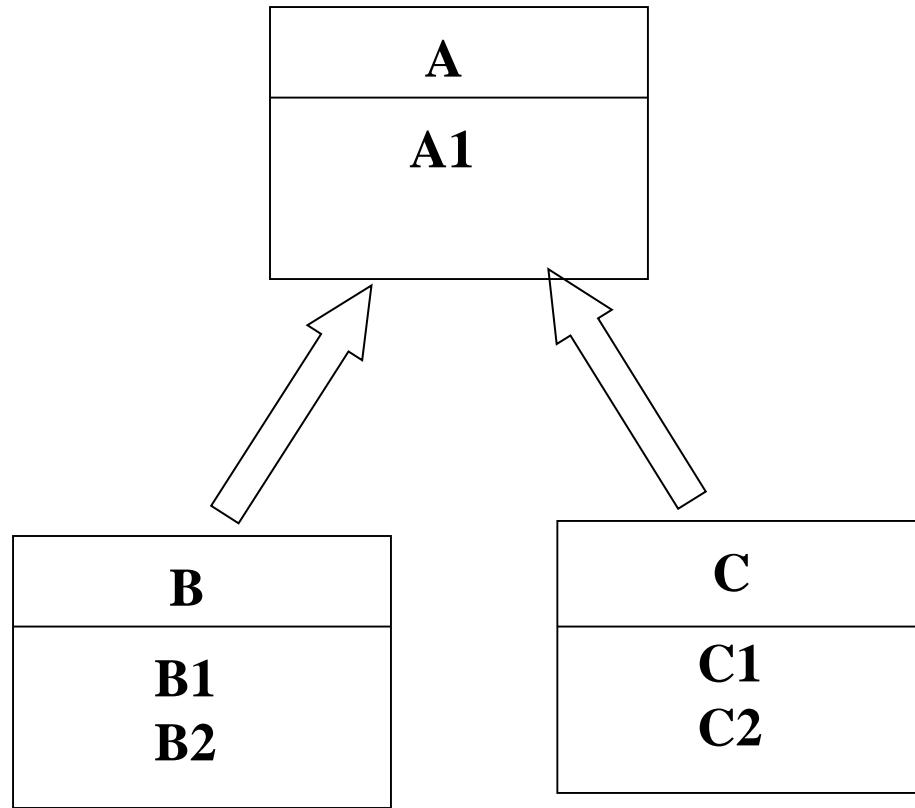


**Create table personne(Id-personne int not null primary key,  
nom varchar(20), Prenom varchar(20),  
datenaissance date, ...)**

**Create table ami\_de ( id\_personne\_P1 int not null refferences personne (id\_personne) ,  
id\_personne\_P2 int not null refferences personne (id\_personne),  
primary key (id\_personne\_P1, id\_personne\_P2))**

## C – Transposition d'un modèle UML dans un contexte relationnel ( Suite )

### Associations d'héritage



**Cas1 : les attributs de spécialisation sont en nombre limité dans B et C**  
Create tableA ( id\_A int not null primary key,

**A1 varchar(30),**

**B1 varchar(30), B2 varchar(30),**

**C1 varchar(30), C2 varchar(30))**

## **Cas2 : l'héritage est associé à une contrainte de couverture**

```
Create tableB ( id_A int not null primary key,
          A1 varchar(30),
          B1 varchar(30),
          B2 varchar(30))
```

```
Create tableC ( id_A int not null primary key,
          A1 varchar(30),
          C1 varchar(30),
          C2 varchar(30))
```

**Cas3 : l'héritage n'est pas associé à une contrainte de couverture (totalité) et les attributs de spécialisation sont nombreux dans B et C**

**Create TableA (id\_A int not null primary key,  
A1 varchar(30))**

**Create tableB ( id\_A int not null primary key references tableA(id\_A),  
B1 varchar(30),  
B2 varchar(30))**

**Create tableC (id\_A int not null primary key references tableA(id\_A),  
C1 varchar(30),  
C2 varchar(30))**