

Licence Professionnelle en Informatique

Génie Logiciel

Orienté Objet

UML



E. Grislin-Le Strugeon

E. Adam

UVHC

ISTV

Plan

- Concepts orientés objet
- Principes des méthodes OO
- Qu'est-ce que UML ?
- Caractéristiques de UML
- Méthode préconisée
- Les 9 diagrammes
- Références
- Outils supportant UML

Historique des objets

1967 : langage de programmation Simula

années 70 : langage de programmation SMALLTALK

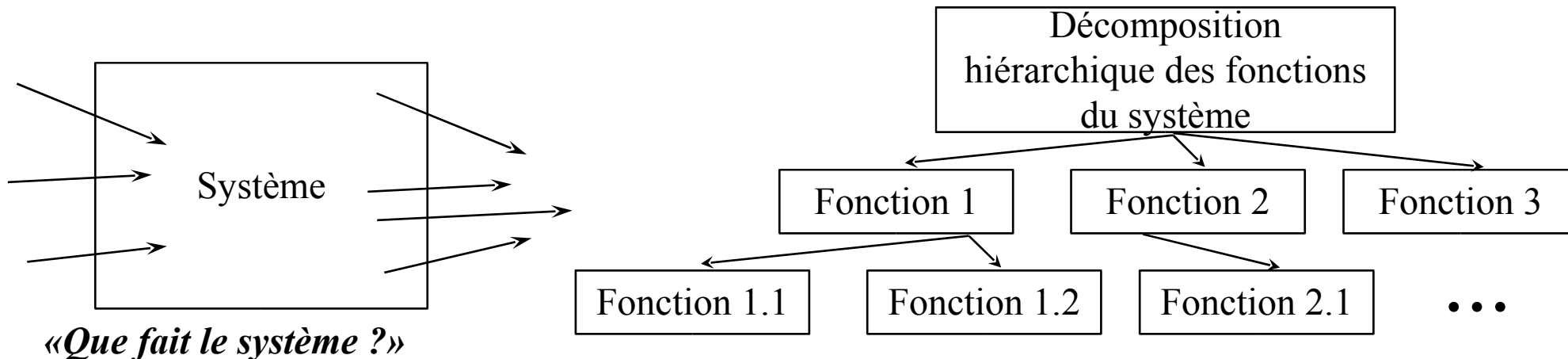
années 80 : fondements théoriques C++, Objective C, ...

années 90 : méthodes d'analyse et de conception OO (Booch, OMT, ROOM, Fusion, HOOD, Catalysis, ...), Java

1997 : UML standardisé par l'OMG
(Object Management Group)

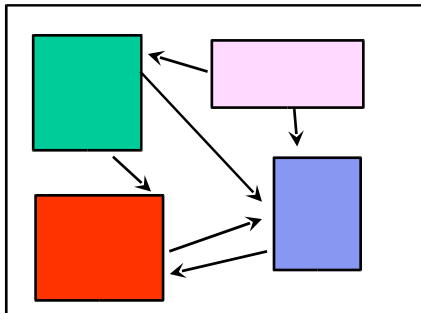
Principe de l'Analyse Fonctionnelle

- Analyse fonctionnelle :
 - accent mis sur ce que **fait** le système (ses fonctions)
 - identification des fonctions du système puis **décomposition en sous-fonctions**, récursivement, jusqu'à obtention de fonctions élémentaires, implémentables directement
 - **la fonction détermine la structure**

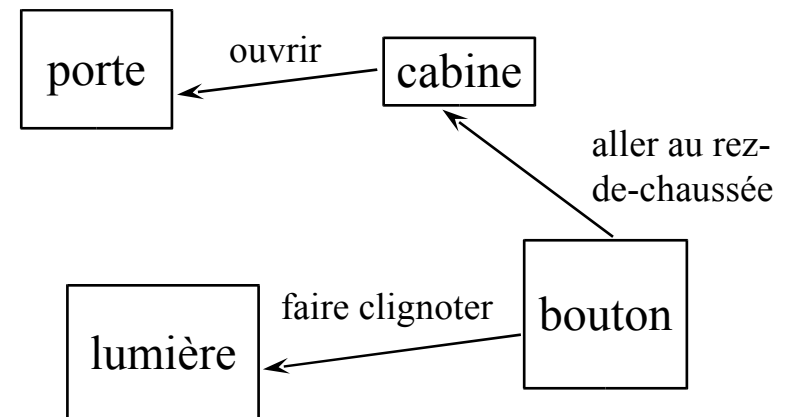


Principe de l'Analyse Orientée Objet

- Analyse OO :
 - accent mis sur ce qu'**est** le système (ses composants)
 - identification des composants du système : les objets
 - fonction = collaboration entre objets
 - **les fonctions sont indépendantes de la structure**
 - un objet intègre à la fois des données et des opérations




«De quoi se compose le système ?»



Orienté Objet

- Concepts Orientés-Objet :
 - abstraction
 - encapsulation
 - héritage
 - polymorphisme
 - objets, classes, événements, états
- Un système informatique est vu comme un ensemble structuré d'éléments qui collaborent

Objet

- L'analyse doit permettre de **fixer (stabiliser) les opérations** (fonctions, services) offertes par un objet.
- opérations fixées  **quelque soit l'architecture interne de l'objet** (la façon dont il est implémenté), son utilisation reste la même
- donc :
modifications possibles de la structure de l'objet sans obligation de révision de son utilisation

Exemple de modif. de structure

- Représentation d'une personne

- 1ère idée : nom et prénom chaînes de [20] car.
date de naissance : int, char[4], int

avec une fonction qui renvoie la différence d'âge entre 2 personnes

- en analyse fonctionnelle classique : une personne est un record (ou struct)
 - en OO : une personne est un objet capable de donner son âge.
- 2ème idée : on veut modifier la date en int, int, int

Conséquences :

- en analyse fonctionnelle classique : nécessité de revoir la fonction de calcul de la différence d'âge
- en OO : simplement modifier «en interne» à la personne, la représentation de la date et l'opération qui renvoie son âge. La fonction de calcul de la différence n'est pas modifiée.

Objet

- Un **objet** est formé d'un **état** et d'un **comportement**
 - un **état** : l'ensemble des valeurs de ses caractéristiques à un instant donné
 - représenté par des **attributs**
 - un **comportement** : manière dont l'objet agit et réagit.
 - représenté par des **opérations**
- objet informatique : **représentation** des entités d'un monde réel ou virtuel, dans le but de les piloter ou de les simuler

Ex. d'objets informatiques

- objets matériels :
porte, ascenseur, bouton, clavier, souris, avion, ...
- différents objets, dans une entreprise :
compte en banque, équation mathématique,
accueil, commercial, contrôleur qualité,
groupe d'abonnés à un forum Internet,
facture, commande, marché, ...
- pour un logiciel de gestion de la clientèle :
client, requête SQL,
bouton, fenêtre, ...

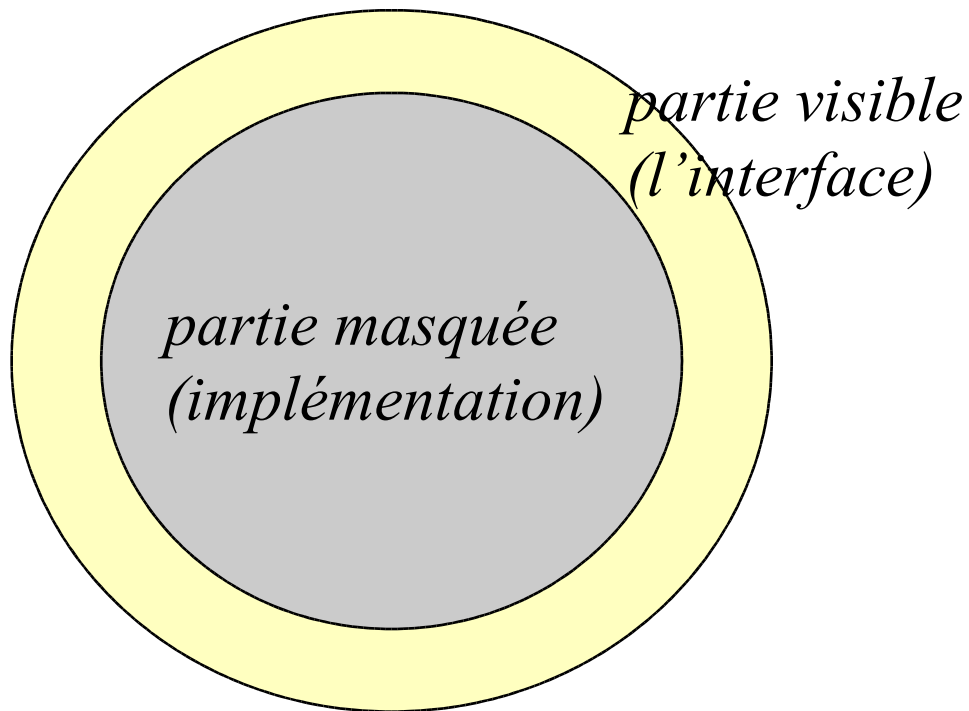
Abstraction

L'abstraction est à la fois :

- un processus : identifier une entité en mettant en évidence ses caractéristiques pertinentes
- un résultat : l'ensemble des caractéristiques d'une entité qui la distingue de tous les autres types d'entités

Encapsulation

L'encapsulation permet de présenter uniquement l'abstraction à son observateur en lui permettant d'ignorer les détails de sa réalisation.



- garantit l'intégrité et la sécurité des données
- facile à maintenir (changements dans l'implémentation sans modifier l'interface)
- concept de «boîte noire»

Ex. *Utiliser une voiture*



Utiliser une voiture

- Bougies,
- Fusibles,
- Cardans,
- Radiateur, ...



Démarrateur,
Direction,
Freins,
Accélérateur,
Changement de vitesse,
Rétroviseur, ...



Ex. 2

réservoir
niveau
remplir vider

réacteur
niveau température
remplir vider chauffer

vanne
position
ouvrir fermer

← attributs

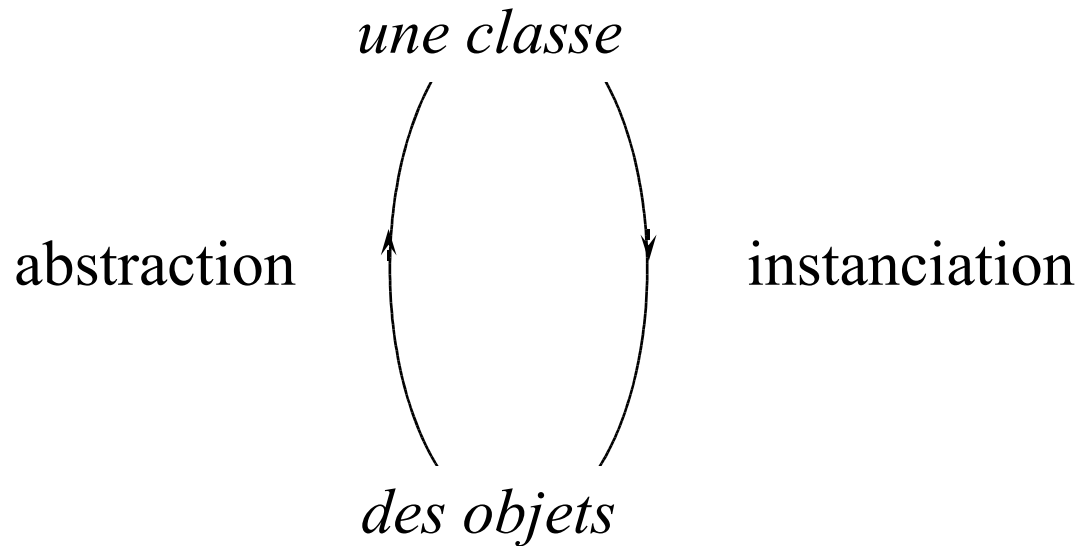
← opérations

modularité et encapsulation

Classe

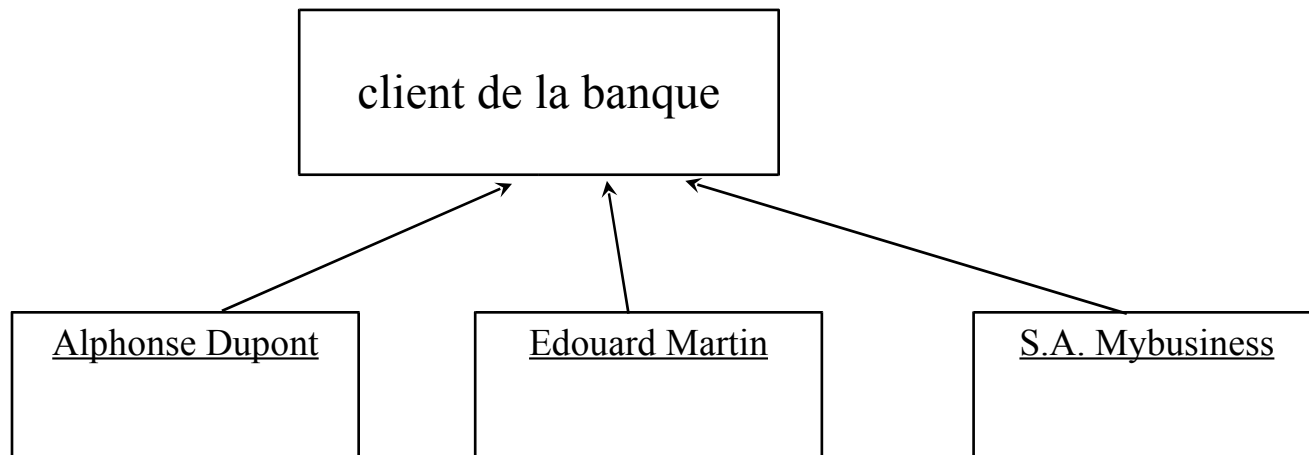
Une classe regroupe des objets qui se ressemblent
(structure commune et comportement commun)

≈ «type»



Ex.

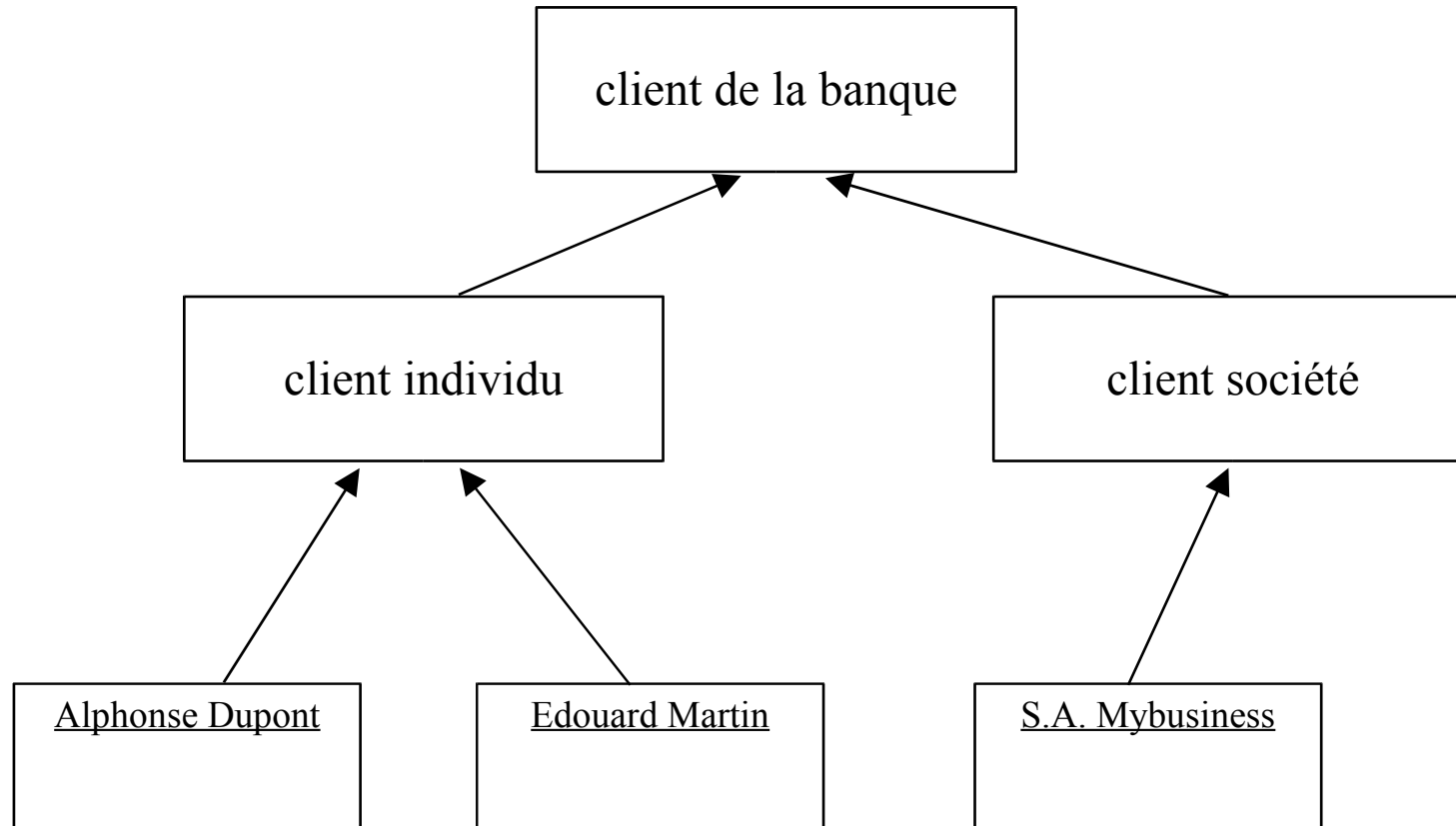
- classes : rôles des salariés
des instances : individus ayant ces rôles
- classes : facture, commande
instances : facture X, facture Y, commande XX, ...



Héritage

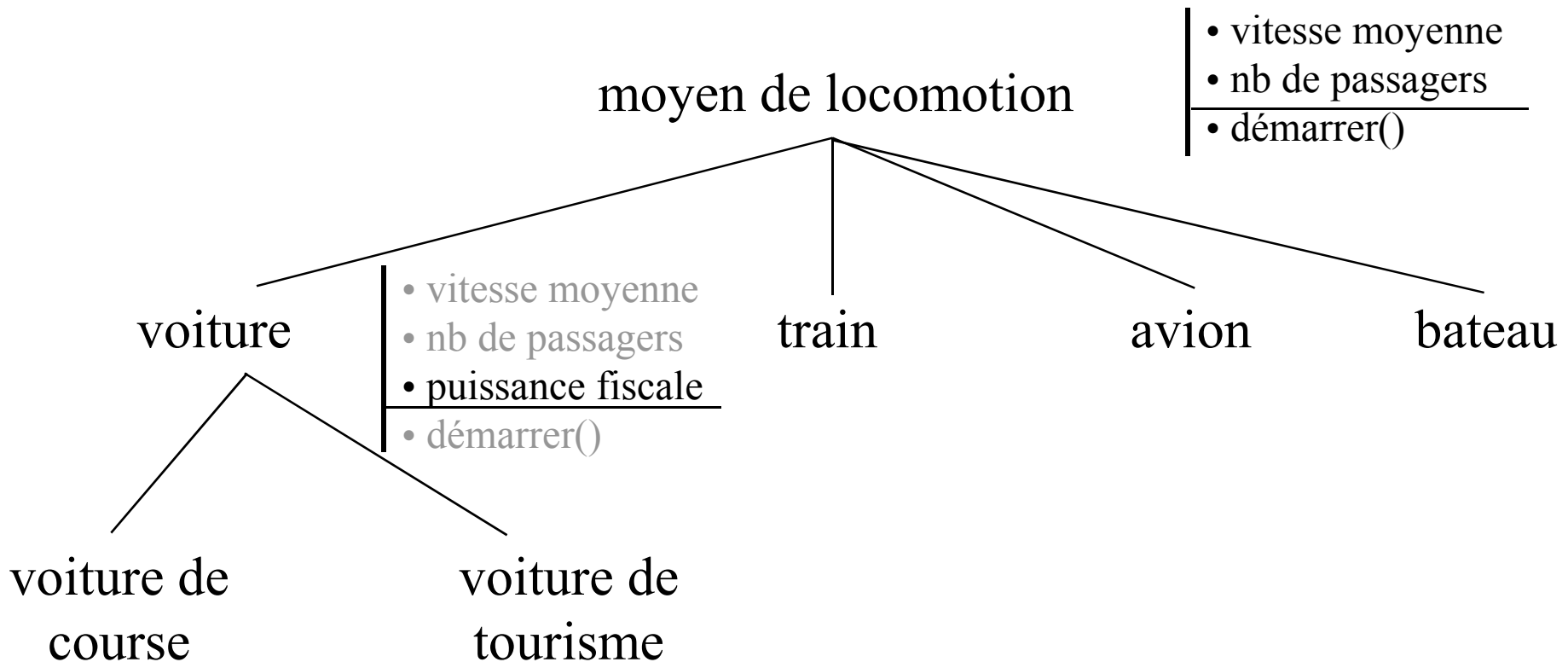
- classement des abstractions en hiérarchie
- **héritage** : transmission des caractéristiques d'une classe vers ses sous-classes
 - héritage simple : une classe hérite des propriétés d'une super-classe (généralisation / spécialisation)
 - héritage multiple : une classe hérite des propriétés de plusieurs super-classes (agrégation)

Ex. de hiérarchie de classe

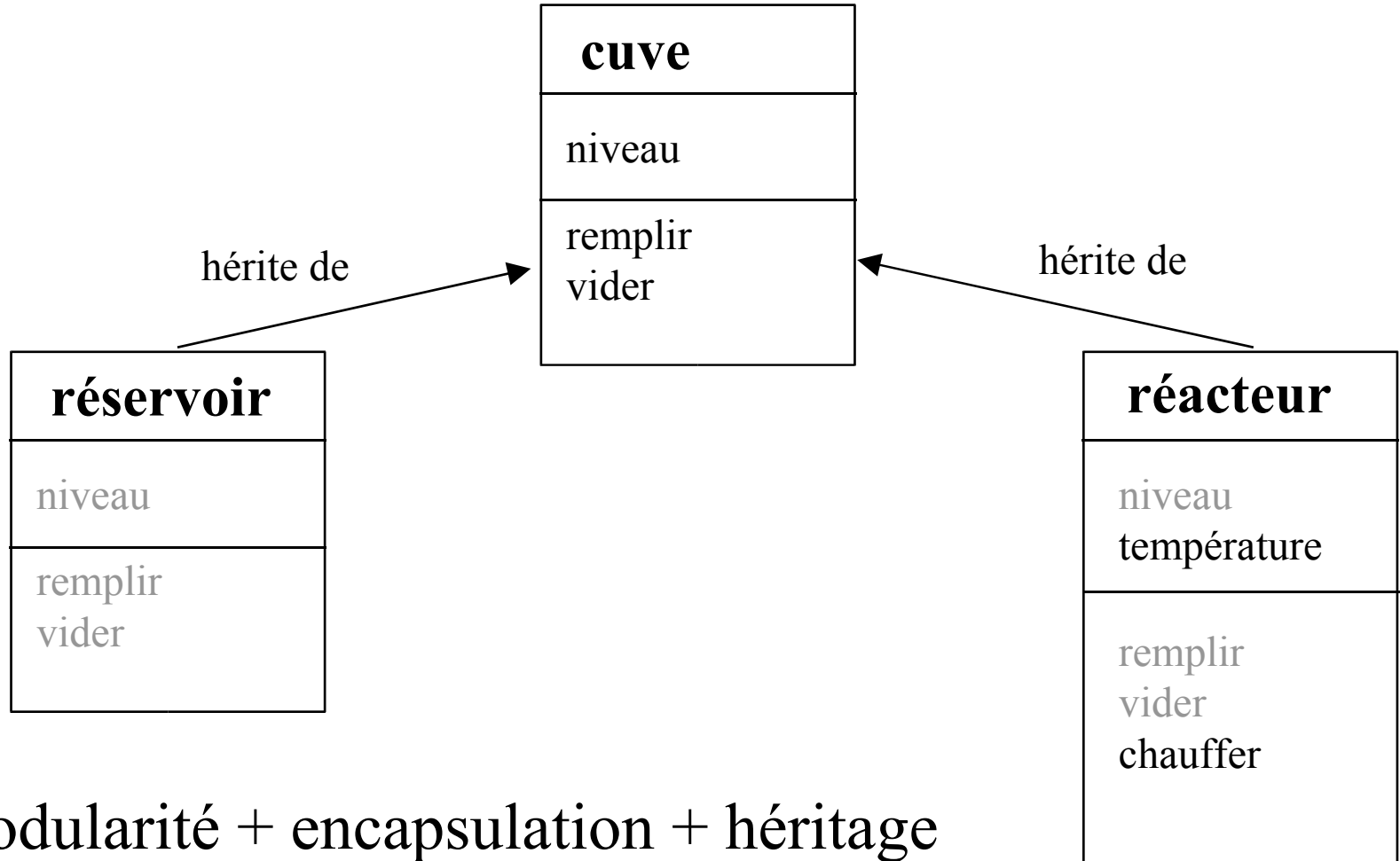


Ex.

- héritage simple : «est un»

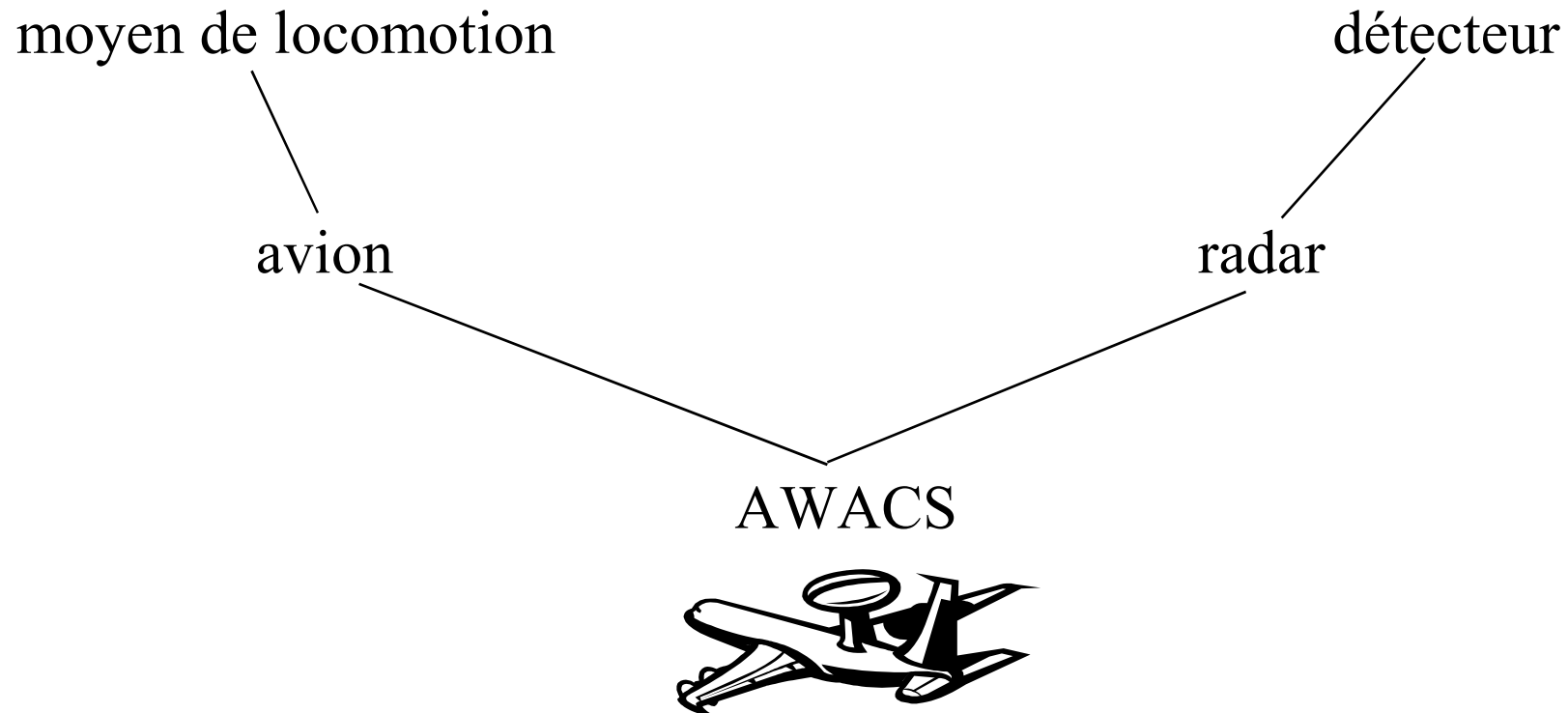


Ex. 2



Ex.

- héritage multiple : «partie de»



Polymorphisme

Polymorphisme :

capacité des objets d'une même hiérarchie à répondre différemment à la même demande

⇒ une méthode peut être interprétée de différentes façons

Ex. 1

moyen de locomotion | démarrer()

démarrer = pédaler



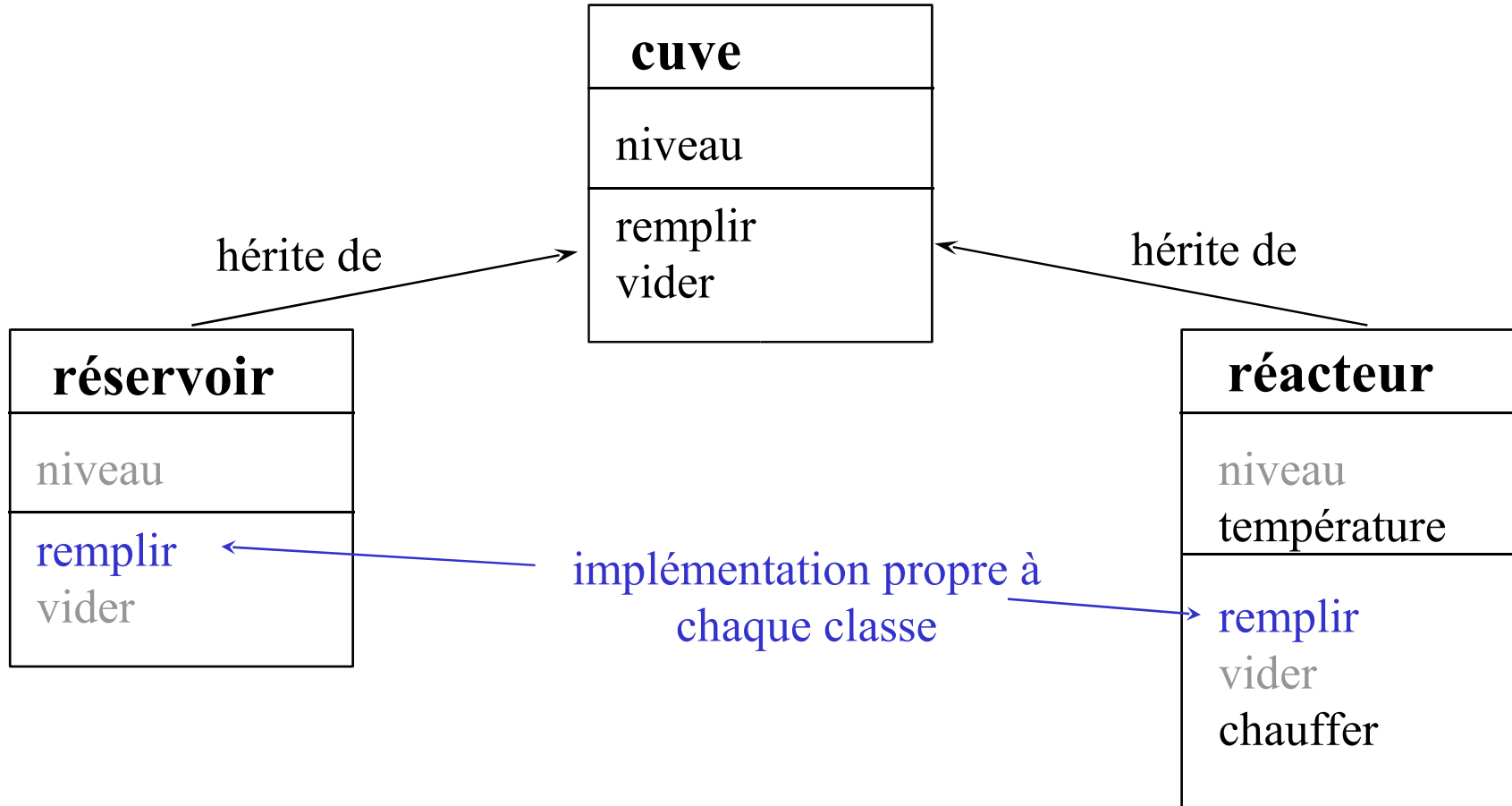
démarrer = mettre les gaz



démarrer = hisser les voiles



Ex. 2



OO = modularité + encapsulation + héritage + polymorphisme

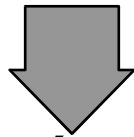
Pourquoi l'objet ?

- pour gérer la complexité de représentation des problèmes
car décomposition en sous-ensembles
- pour gérer la complexité de résolution des problèmes
car gestion de sous-problèmes
- pour gérer le processus de développement
car développement modulaire («par morceaux»)
- pour maintenir le logiciel
car extensibilité et réutilisabilité
- pour sécuriser le logiciel
car possibilité de masquage des données

Principes des méthodes OO

Constat:

les fonctions et les données sont étroitement liées



ne pas les séparer lors de la conception

\neq conception classique (découpage fonctionnel)

Difficulté :

déterminer les objets fondamentaux du système

Principes des méthodes OO

- Analyse OO :
 - trouver les objets
 - les organiser (les classer et les regrouper)
 - décrire leurs interactions (scénarios, interfaces)
 - définir leurs opérations (d'après les interfaces nécessaires)
 - définir l'intérieur des objets (informations stockées)
- Programmation OO :
 - on peut avoir analyse OO et programmation non OO
 - le style de programmation peut être OO même avec un langage classique
- Test OO : le test peut être considéré lui-même comme un objet

OOD

- **Grady BOOCH** : pionnier de l'Orienté-Objet (1981)
- Distingue 2 niveaux : logique et physique
 - logique :
 - diagrammes de classes et d'instances
 - diagrammes états/transitions
 - diagrammes de temps
 - physique :
 - diagrammes de modules et de processus
- Version la plus aboutie : Booch'93

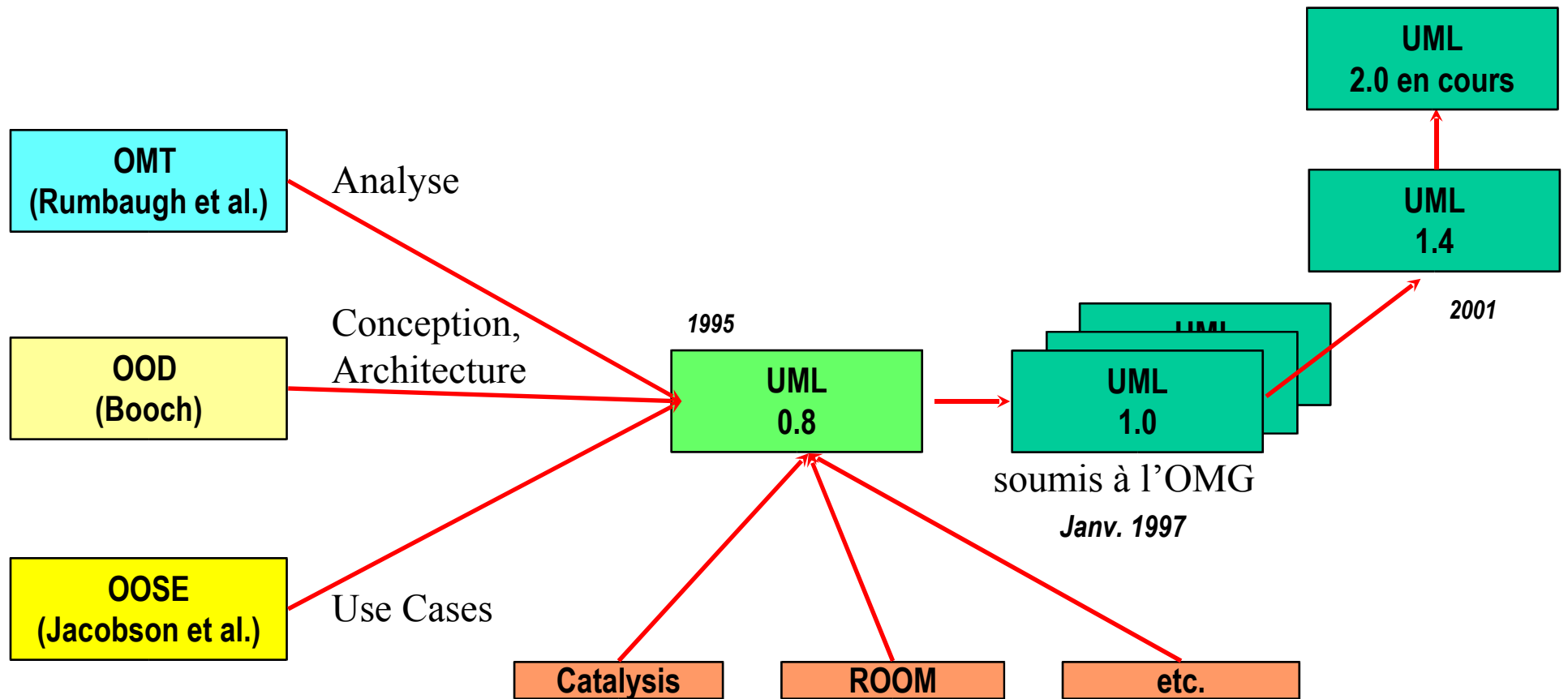
OMT

- **James RUMBAUGH et al. (1991)**
- **OMT = Object Modeling Technique**
- 3 modèles : statique, dynamique et fonctionnel
- 4 phases : analyse, conception système, conception objet, implantation
- Evolution constante depuis 1991

OOSE

- **Ivar JACOBSON**
- **OOSE** = Object-Oriented Software Engineering
- = **Objectory**
- 5 modèles : besoins, analyse, conception, implantation, test
- 3 types d'objet : entités, contrôles, interfaces
- notion de cas d'utilisation (Use Cases)

Convergence



Emprunts

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

Qu'est-ce que UML ?

UML = **Unified Modeling Language**
for Object-Oriented Development

C'est le 1er standard international en conception de système d'information.

Il provient de l'unification de différents modèles
Orientés-Objet.

Caractéristiques de UML

UML fournit :

- une notation
- un méta-modèle sémantique
- un langage de définition de contraintes sur les objets (OCL)
- une définition d'une interface avec les services de CORBA

UML / méthodes des années 80

UML réutilise les 2 points forts des méthodes des années 80 :

- un modèle des classes fondé sur le modèle des données (sorte de MCD, modèle Entité-Association étendu)
- un modèle des scénarios fondé sur les modèles de processus (MCT, MOT) à base de réseau de Petri simplifié

Caractéristiques de UML

- UML **standardise** les modèles et notations utilisés lors du développement :
 - modèles sémantiques
 - notations syntaxiques
 - diagrammes
- UML **ne standardise pas** le processus de développement

Structure

Le langage de description est basé sur une structure qui assure la cohérence de la notation par un ensemble de mécanismes :

- les stéréotypes,
- les étiquettes,
- les notes,
- les contraintes,
- la relation de dépendance,
- les dichotomies type-instance, type-classe

Mécanismes

- stéréotype

tout élément de modélisation d'UML
est basé sur un stéréotype (méta-
classification)

«interface»

- étiquette

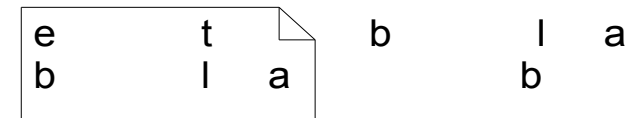
couple (nom,valeur)

décrit une propriété d'un élément

{pédale_accélération = off}

- note

commentaire attaché à un élément



- contrainte

relation sémantique entre éléments

{contrainte}

Types primitifs

- booléens
- expressions
- listes
- multiplicité
- nom
- point
- chaîne
- temps
- ...

UML : méthode ou modèle ?

Une méthode comporte :

- un **langage de modélisation** : une notation utilisée pour décrire les éléments de modélisation
- un **processus** décrivant les étapes et les tâches à effectuer pour mener à bien la conception

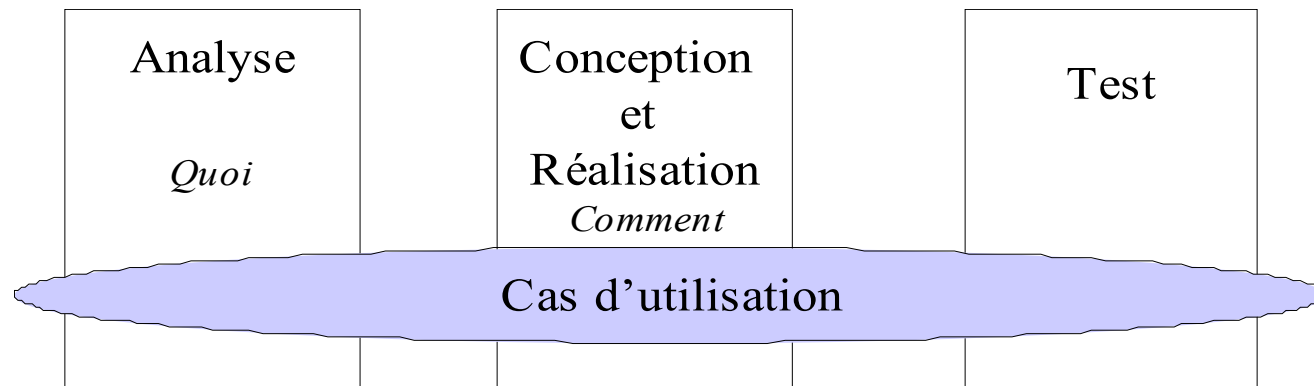
Ce dernier point est absent de UML, donc :

UML n'est pas une méthode mais un langage d'expression des éléments de la modélisation.

UML : des modèles => une méthode

UML ne définit pas de processus de développement standard
mais un processus est préconisé :

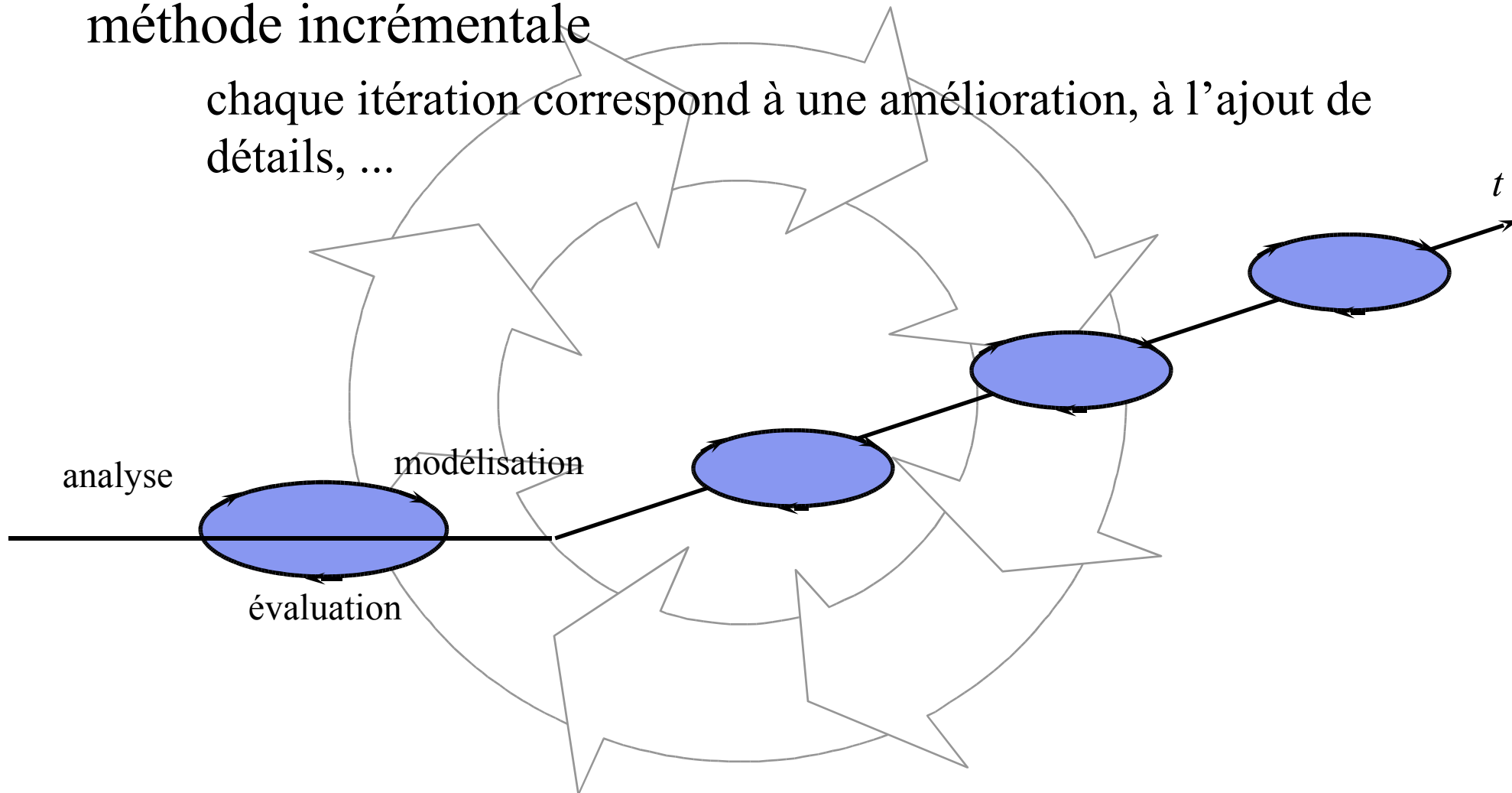
- guidé par les Use Cases
- centré sur l'architecture (la structure) du système
- itératif et incrémental (construction progressive)



Méthode préconisée

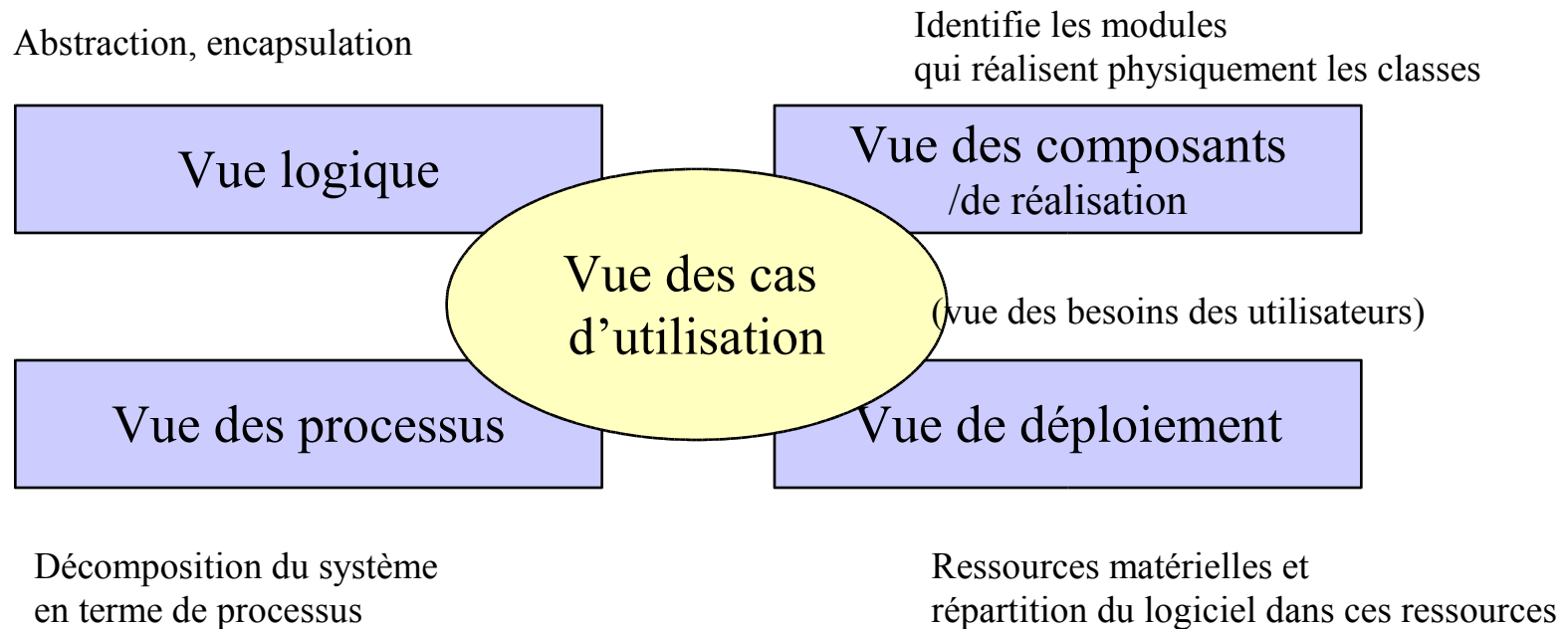
méthode incrémentale

chaque itération correspond à une amélioration, à l'ajout de détails, ...

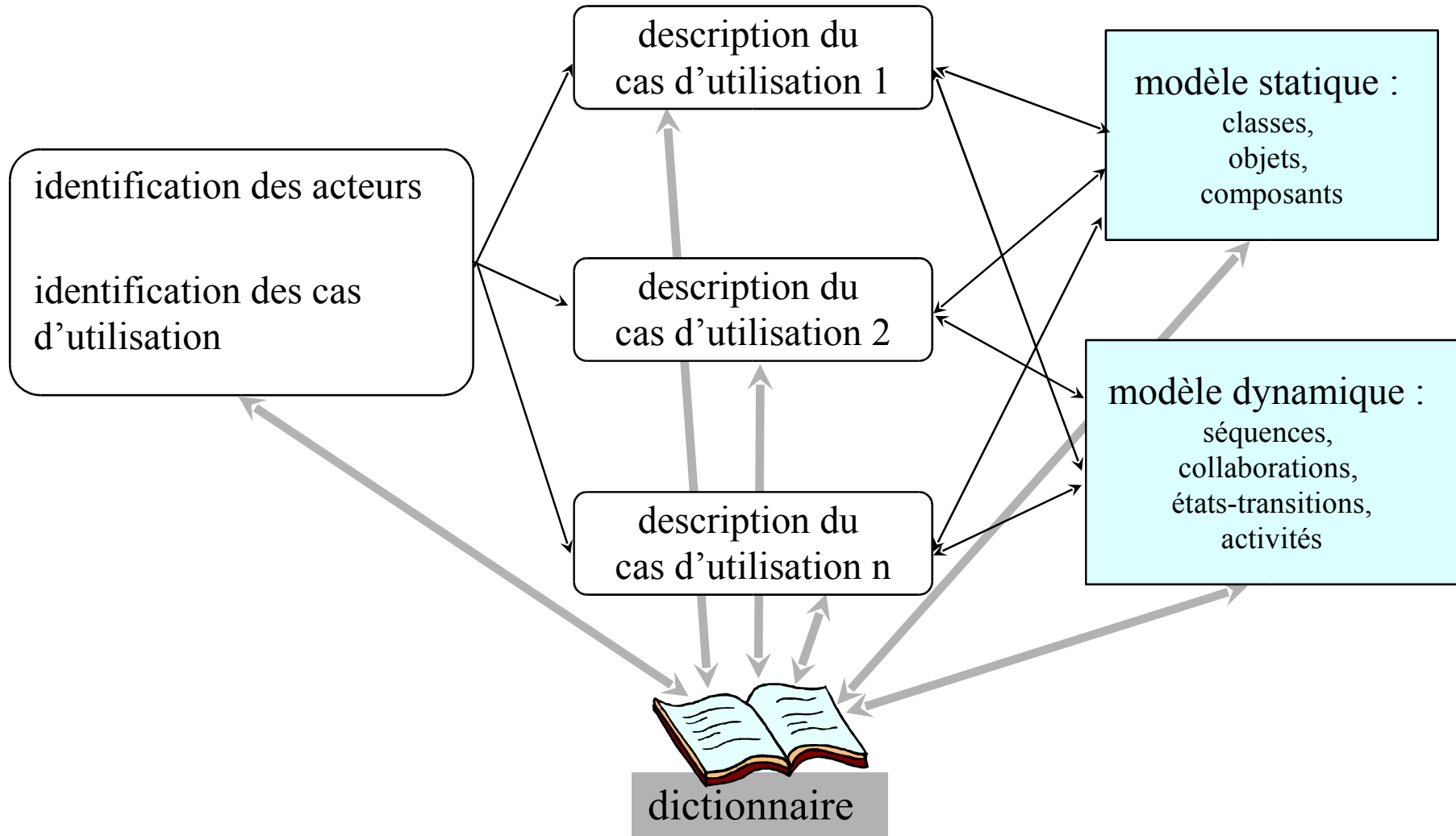


Modèle des «4+1 vues»

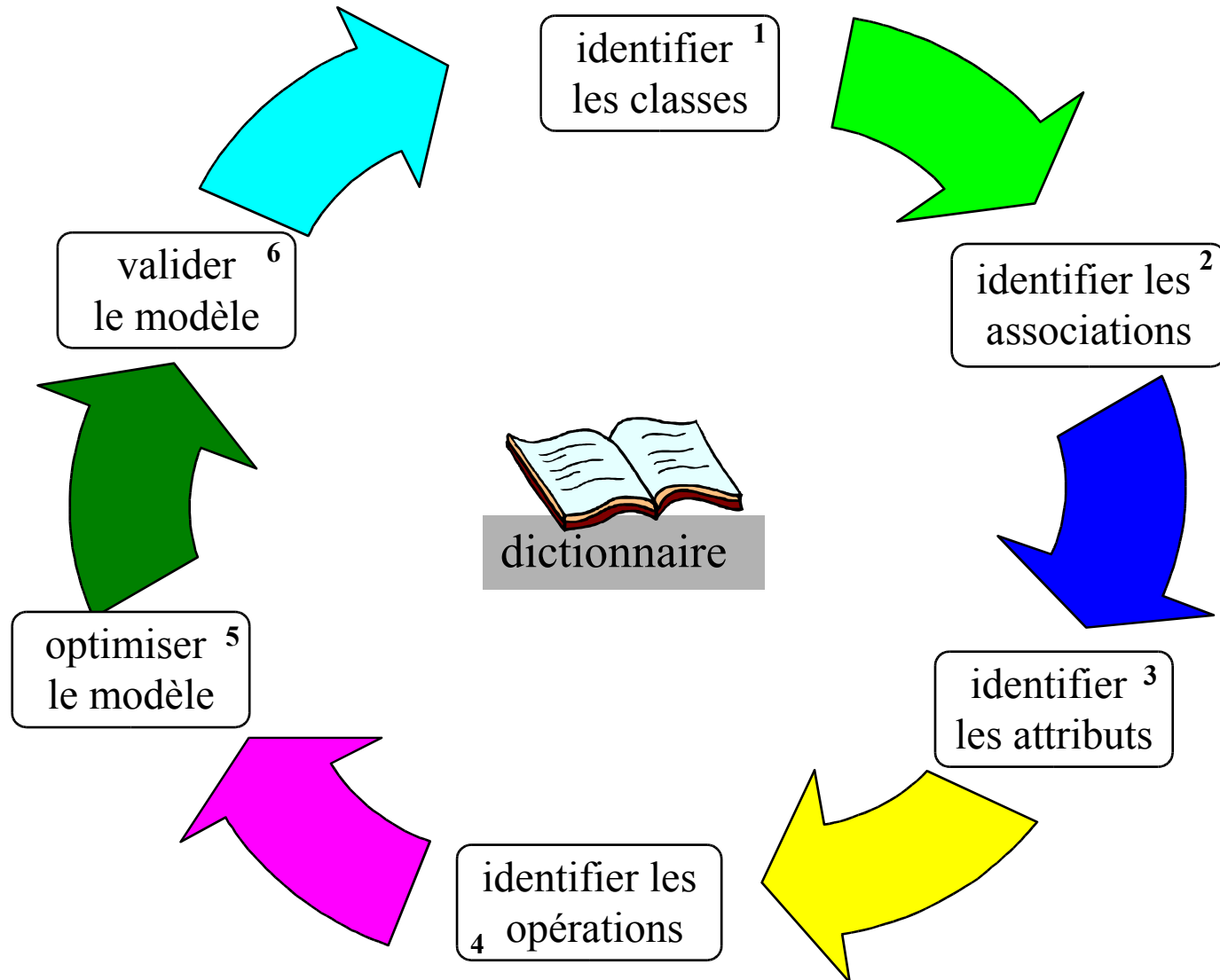
Vision de l'architecte du logiciel et les modèles associés :



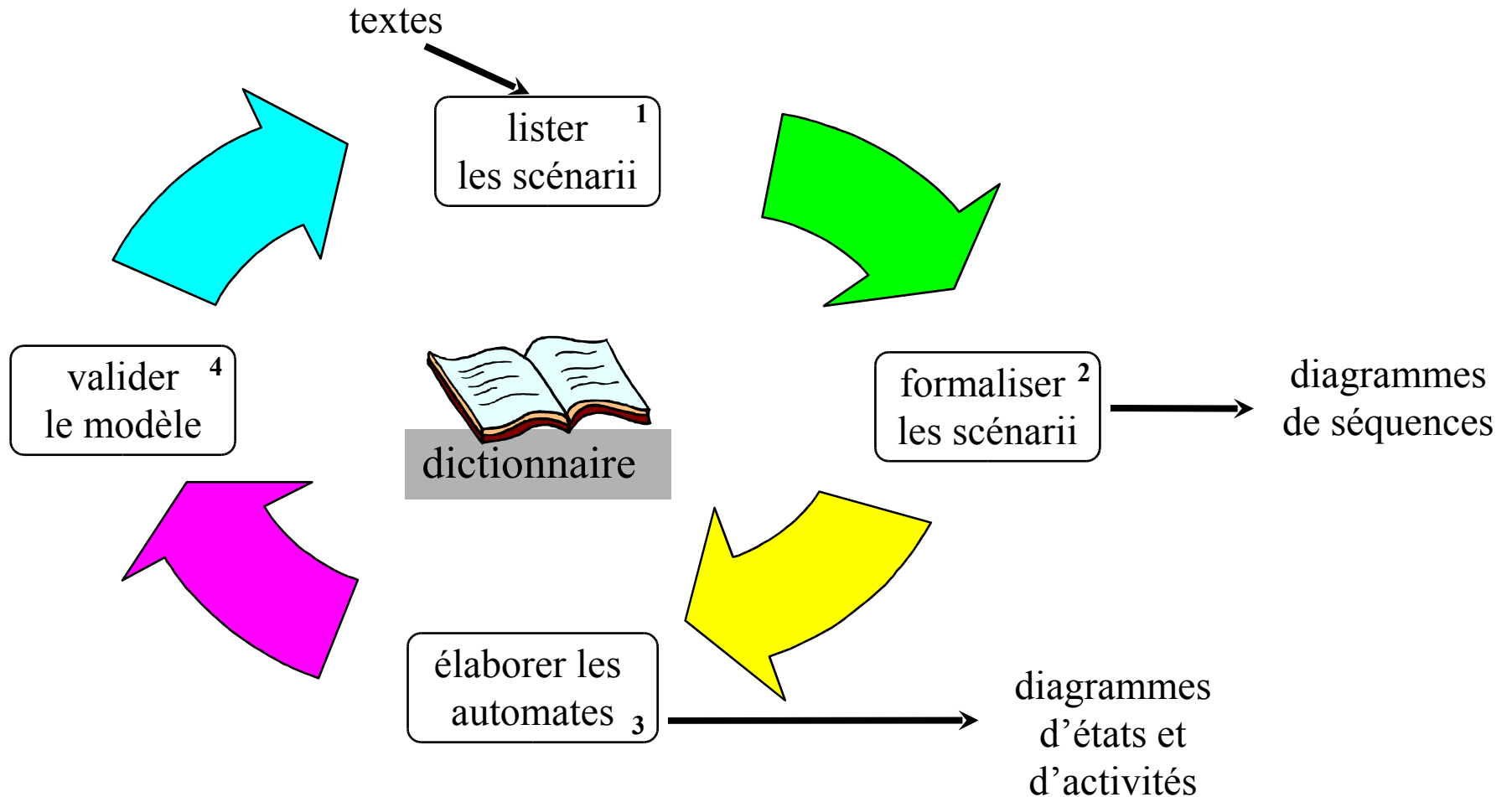
Synthèse de la méthode de modélisation



Construction du modèle statique

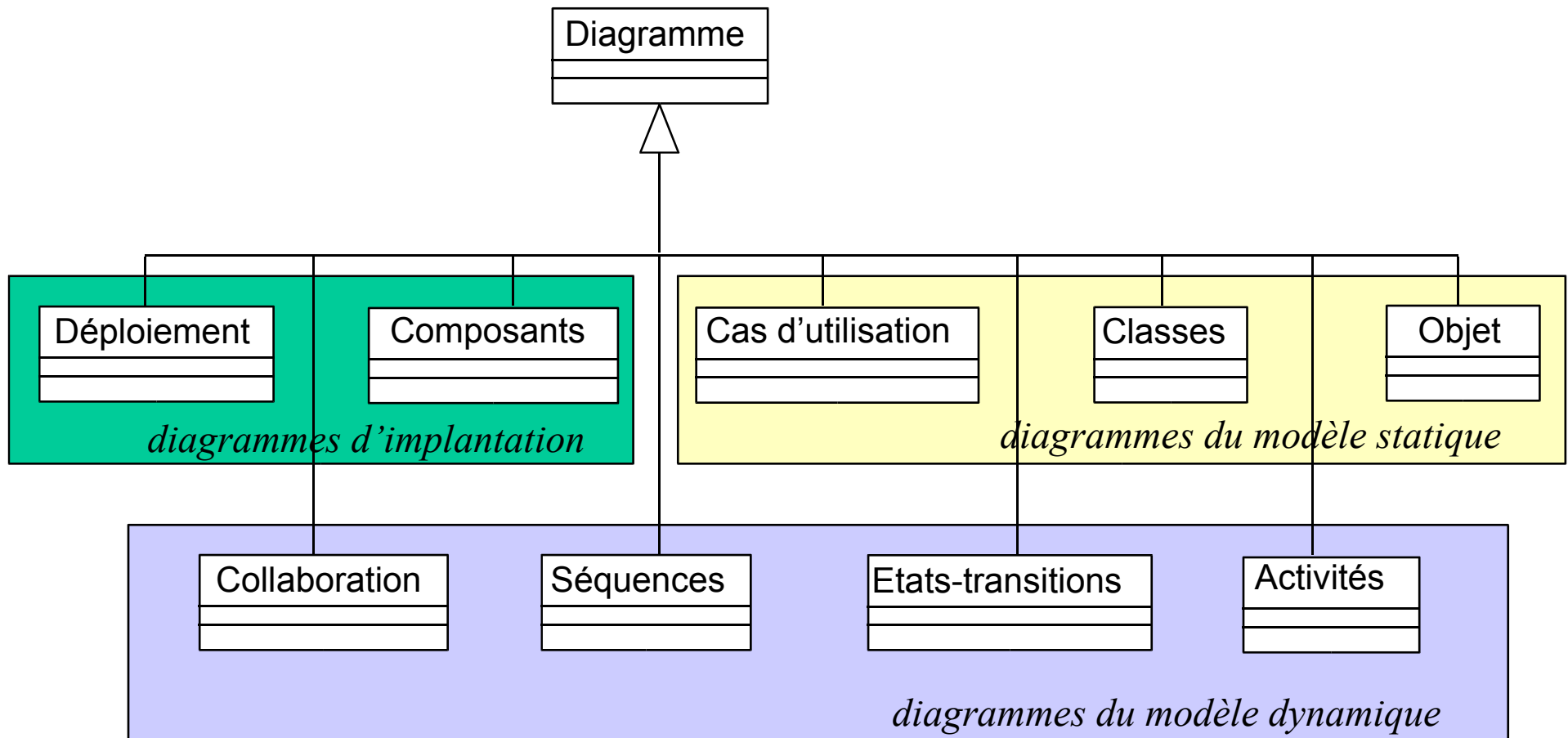


Construction du modèle dynamique



Les 9 diagrammes

les 9 diagrammes UML :



Vue logique / cas d'utilisation

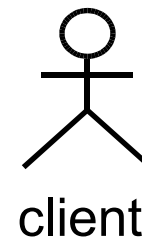
- Identification des acteurs et des cas d'utilisation (première analyse des besoins)
 - détermination des acteurs (types d'utilisateurs)
 - les acteurs sont externes au système. Déterminer les limites du système
 - représentation des **acteurs**
 - services attendus exprimés par les interactions acteur - système (pour chaque acteur)
 - représentation des **cas d'utilisation** :
 - description textuelle
 - une vue générale, peu détaillée
 - une vue par acteur, plus détaillée

Acteur

Acteur = entité externe agissant sur le système :

être humain, machine, autre système ou sous-système

- L'acteur peut consulter et/ou modifier l'état du système
(les entités externes passives ne sont pas des acteurs)
- un acteur est défini en fonction du **rôle** qu'il joue
- il représente un nombre quelconque d'entités externes agissant sur le système
- c'est un stéréotype de classe (classe «actor» prédéfinie)



Acteurs / Personnes ?

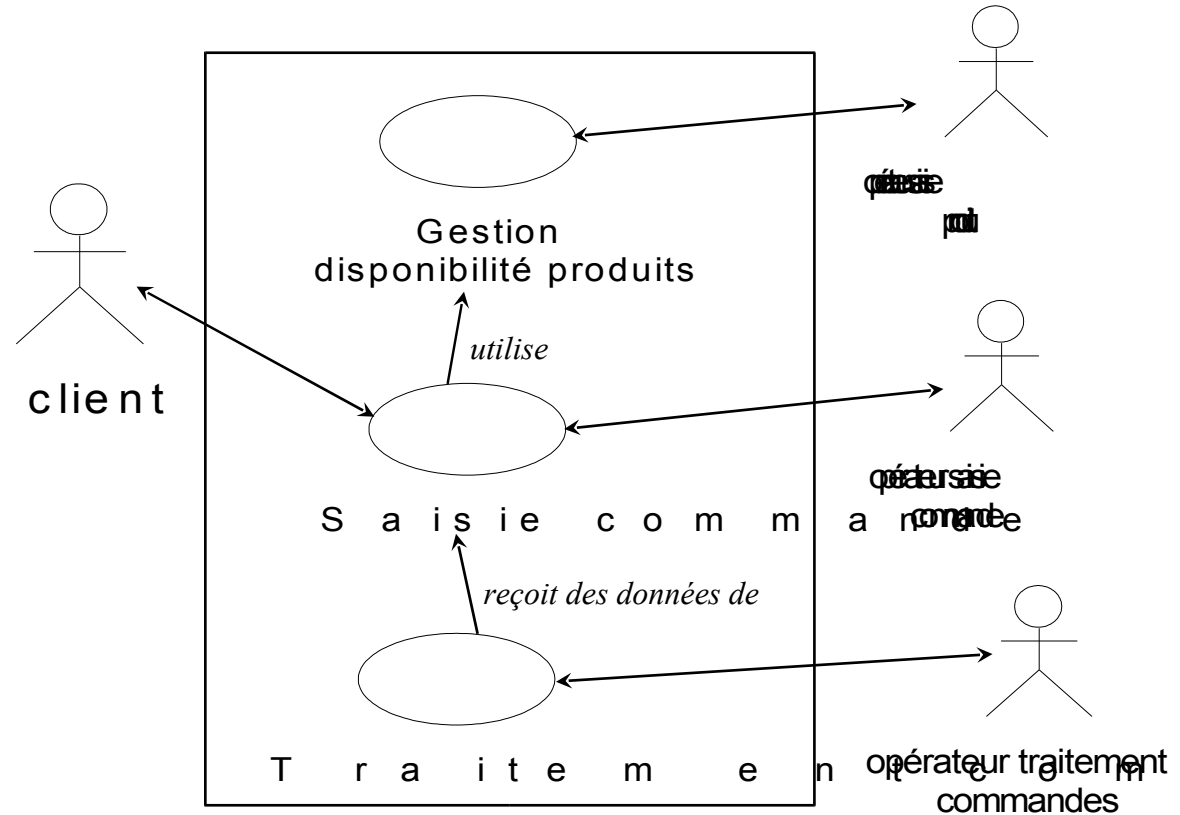
les personnes sont souvent représentées par 3 éléments du modèle :

- des acteurs : utilisateurs du système
- des objets (objets métier) : informations décrivant chaque utilisateur
- des objets d'interfaces : manipulation des informations contenues par les objets métier

Diagramme des Cas d'utilisation (Use Cases)

Modèle du système du point de vue de son utilisation (fonctions) :

- interactions entre le système et les acteurs
- réactions du système aux événements extérieurs

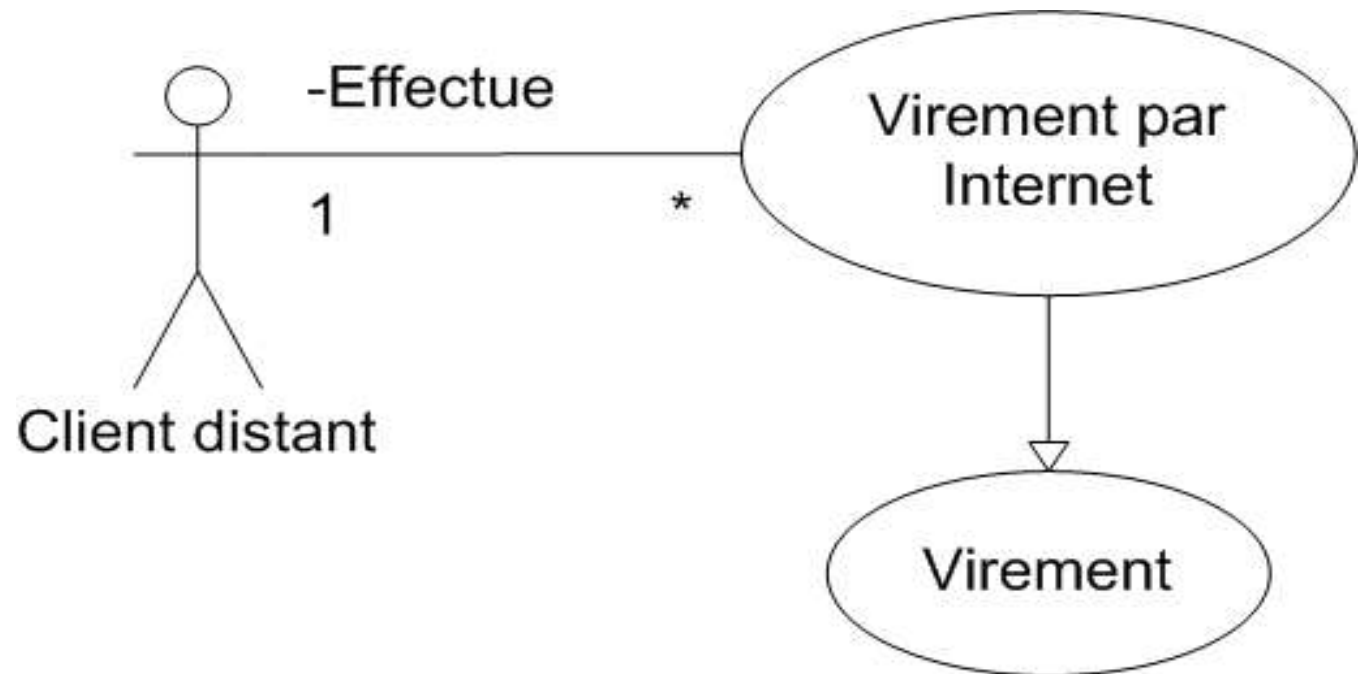


L'ensemble des cas d'utilisation décrit les exigences fonctionnelles du système : c'est un **moyen de communication** entre les utilisateurs et l'équipe de développement

Organiser les cas d'utilisation (1)

Généralisation :

les cas d'utilisation descendants héritent de la sémantique de leur parent

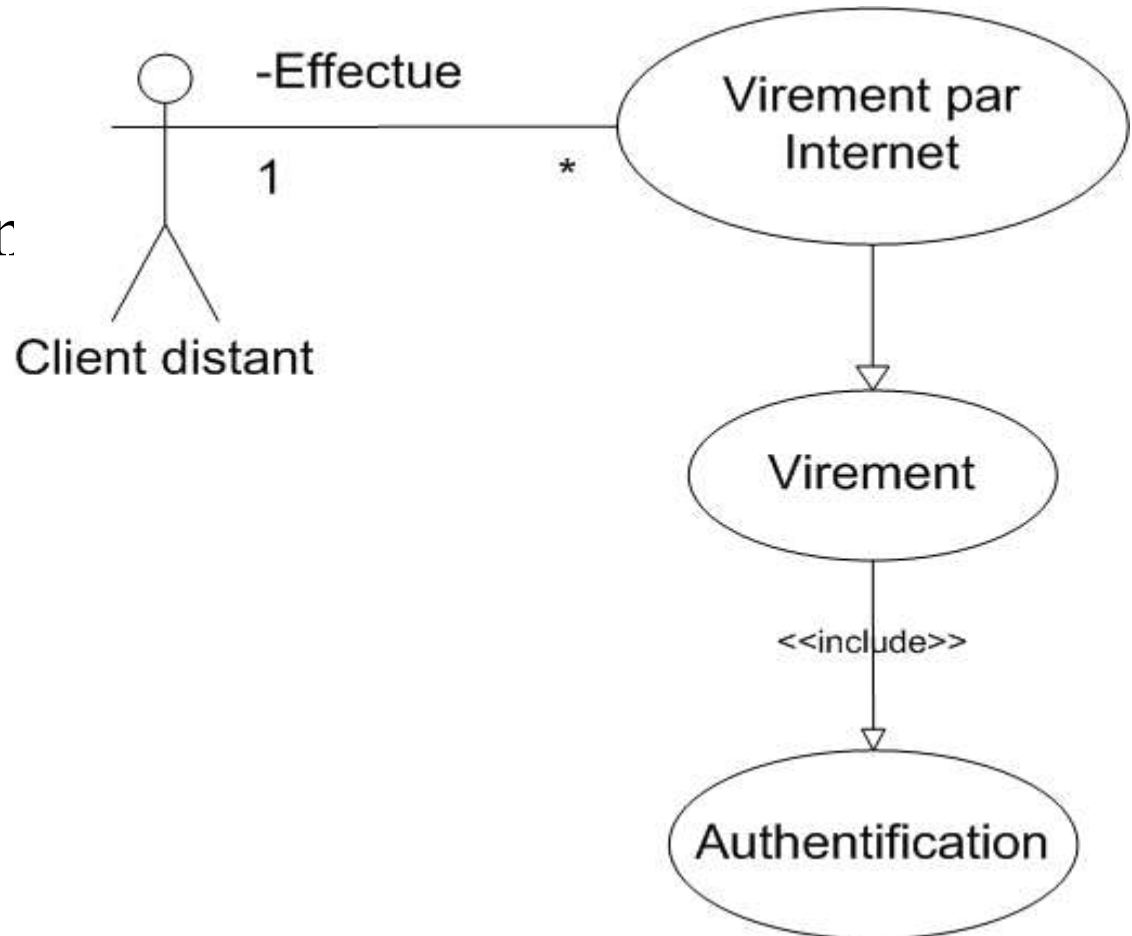


Organiser les cas d'utilisation (2) : <<include>>

Inclusion :

le cas d'utilisation inclus est exécuté en tant que partie d'un cas d'utilisation plus vaste

*Le mot clé **uses** ou **utilise** est également accepté*



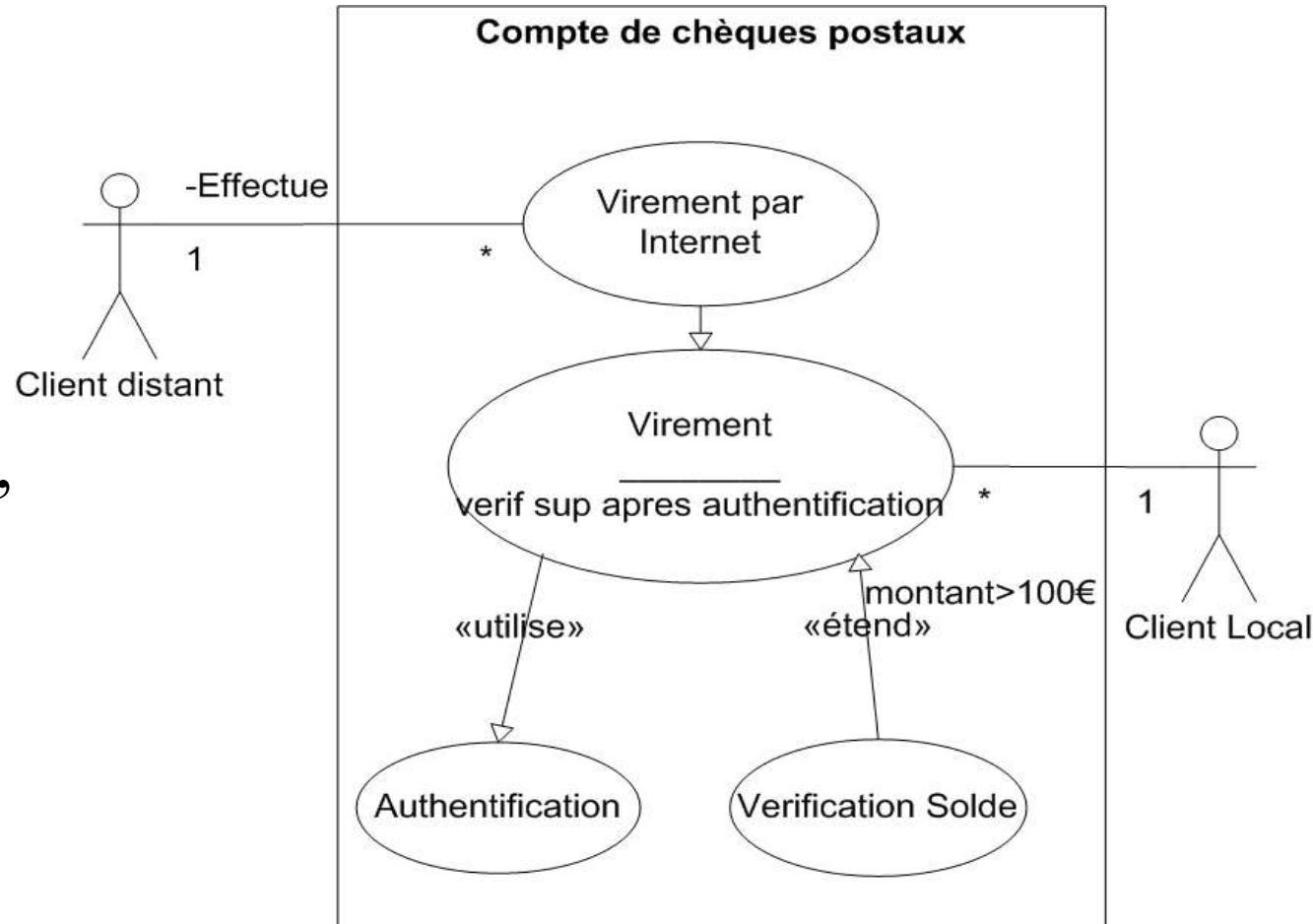
Organiser les cas d'utilisation (3):

<<extend>>

Extension :

le cas d'utilisation de base peut fonctionner seul,
mais il peut être complété par un autre,

à certains endroits
(*points d'extension*)



Vue logique : Détail des cas d'utilisation

Pour chaque cas d'utilisation :

- modèle statique :
 - diagramme d'objets participants (simple)
 - identification des objets/classes
 - analyse textuelle du cahier des charges :
 - les classes sont très souvent décrites par des noms communs.
 - les opérations sont souvent représentées par des verbes
 - généralisation : ébauche du diagramme de classes
 - rechercher les classes possédant des caractéristiques communes : identification de catégories
 - regroupement des classes par catégorie (1 classe appartient à 1 seule catégorie)
 - stéréotypes prédéfinis de classes :
 - contrôleur : pilotage et l'ordonnancement
 - dispositif : manipulation du matériel
 - interface : traduction d'information à la frontière entre 2 systèmes
 - substitut : manipulation d'objets externes au système
 - vue : représentation des objets du domaine

Détail des cas d'utilisation

Pour chaque cas d'utilisation :

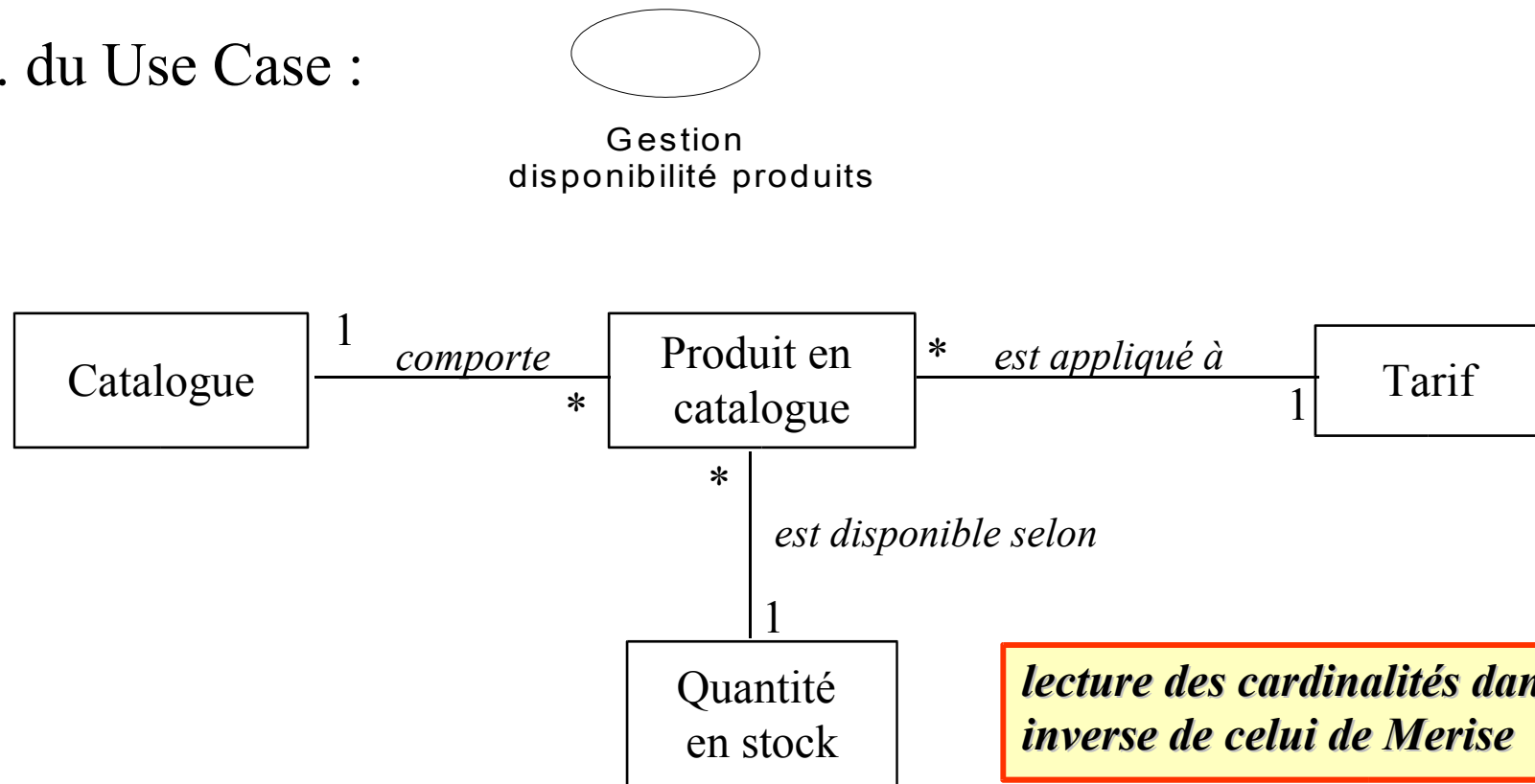
– modèle dynamique :

- représentation des **scénarii** d'interaction acteur-système par des **diagrammes de séquence**
- diagramme de collaboration
 - Les fonctionnalités décrites par les cas d'utilisation sont réalisées par des collaborations d'objets
 - les relations entre classes correspondent à des collaborations potentielles
- diagrammes d'états (automates)
- diagrammes d'activités

Diagramme d'objets participants

On construit un diagramme d'objet simplifié pour chaque Use Case décrit textuellement.

ex. du Use Case :



lecture des cardinalités dans le sens inverse de celui de Merise

Diagramme de classes

description des entités du système et de leurs relations
(structure statique)

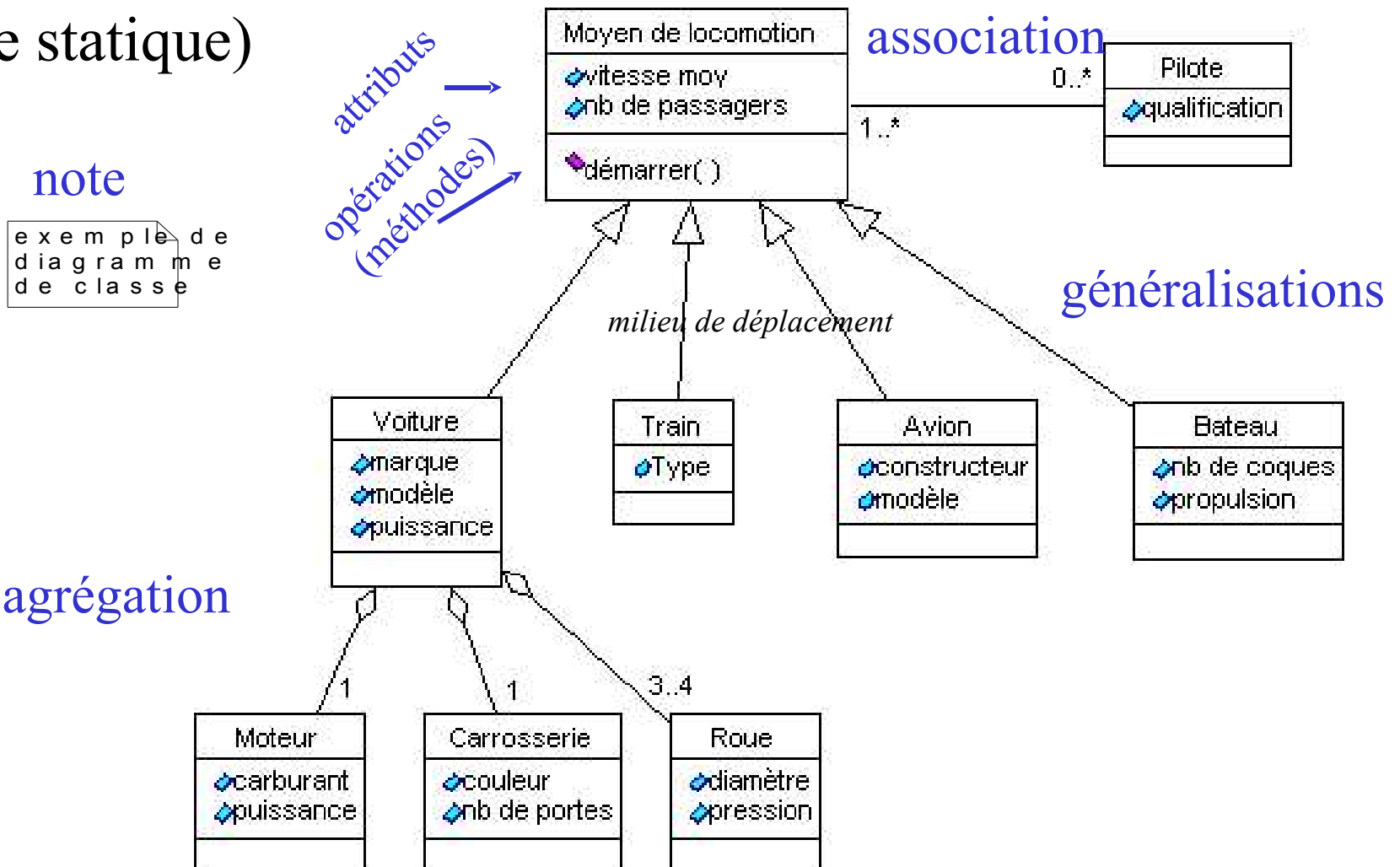


Diagramme de classes: Paquetages

Un paquetage (package) définit un sous-système ou une catégorie de classes

Notation :

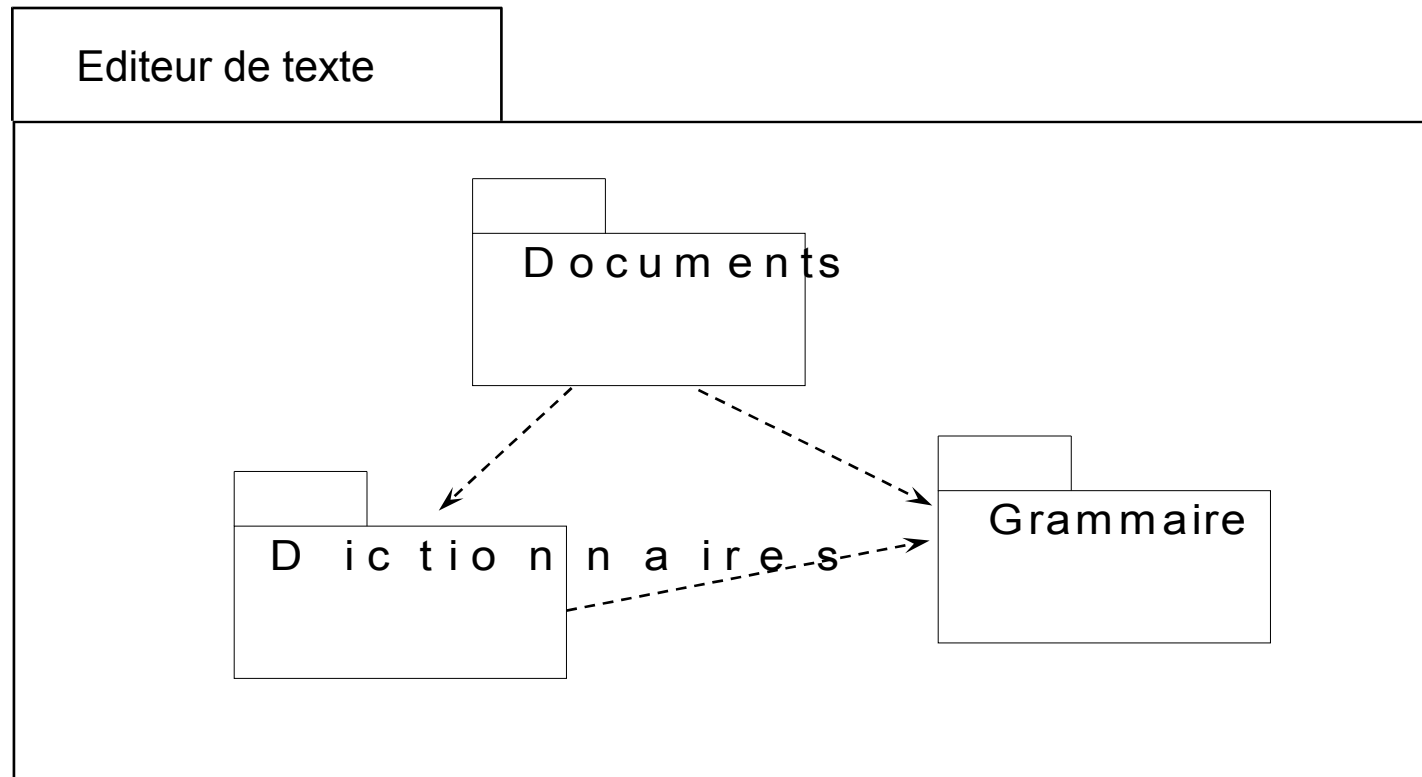


Diagramme de classes: Interfaces

Une interface est l'élément le plus abstrait d'un diagramme de classes

Elle contient

des attributs de type constante uniquement

des déclarations de fonctions (pas d'implémentation)

Notation :

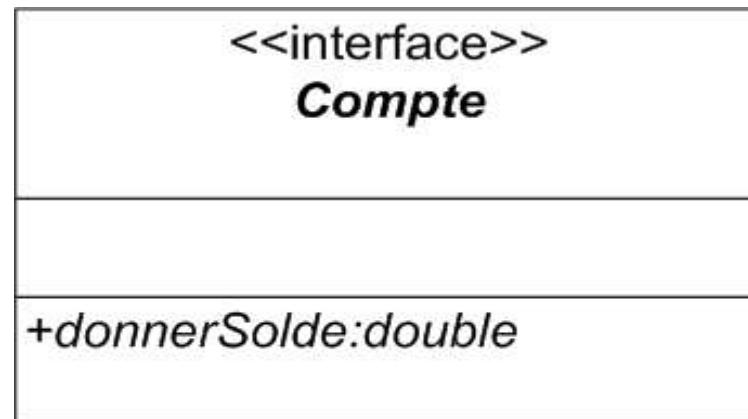


Diagramme de classes: Classes Abstraites

Une classe abstraite contient des fonctions non implémentée
Pas de possibilité de créer des objets directement
Il faut étendre ces classes (par héritage)

<i>CompteBanquaire</i>
+compte:double
+ <i>authentifier:void</i> +donnerSolde:double

Diagramme de classes: Classes

Une classe permet de définir un type d'objet

Un membre privé est représenté par –

Un membre protégé est représenté par #

Un membre public est représenté par +

CompteBanquaireAuthentifié
+nom:String #solde -noCarte:long -typeCarte:String
#authentifier:void -changerCode:void +donnerType:String

Diagramme de classes: Relations (1)

Une classe **implémente** une ou des interfaces et ne peut **hériter** que d'une seule classe

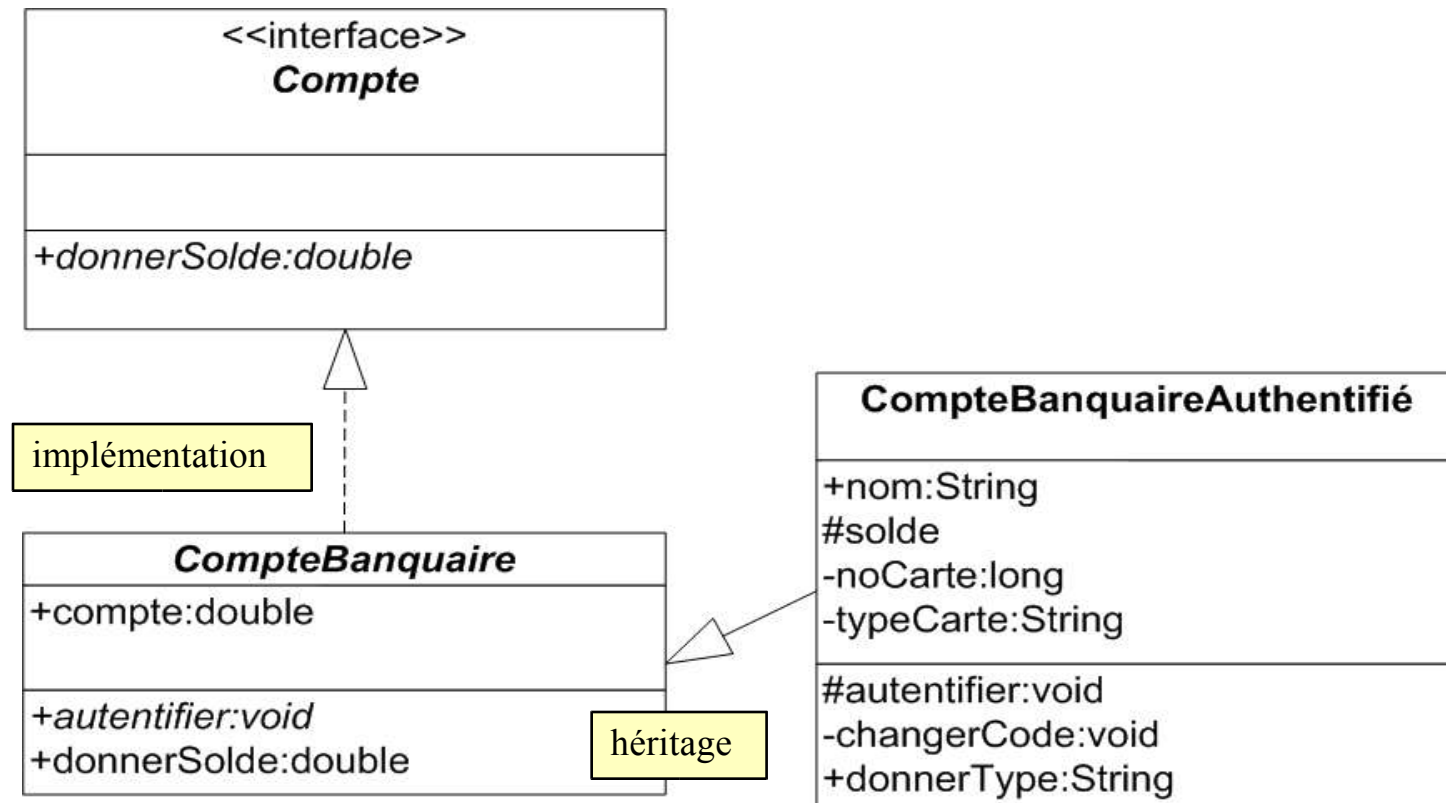
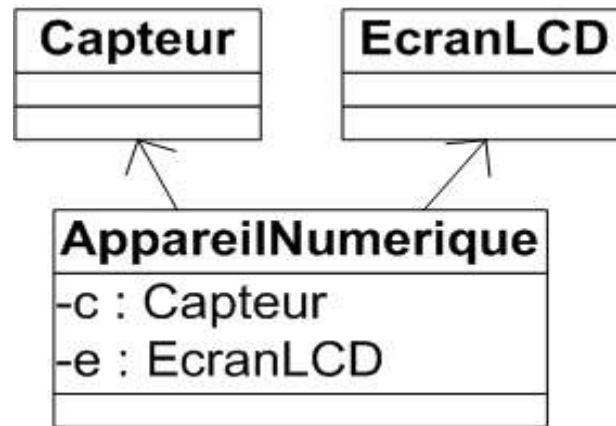


Diagramme de classes: Relations (2)

L'agrégation signifie qu'un objet en contient d'autres



L'agrégation est souvent représentée par une association

Diagramme de classes: Relations (3)

L'association représente une liaison entre classe



Diagramme de classes: Relations (4)

La dépendance représente une relation sémantique entre deux classes

Plus floue et difficile à mettre en œuvre, souvent ignorée

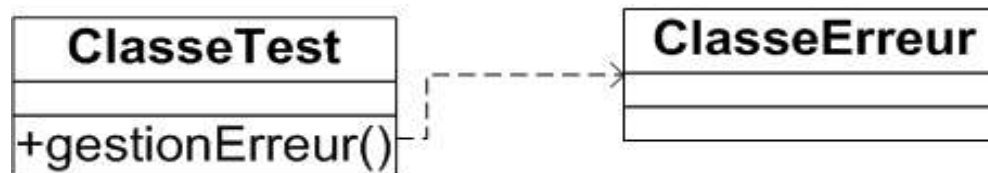


Diagramme de classes: Stéréotype

Le stéréotype représente une appartenance à une catégorie de classe (processus, énumération, ...)

N'a pas d'influence sur la génération de code

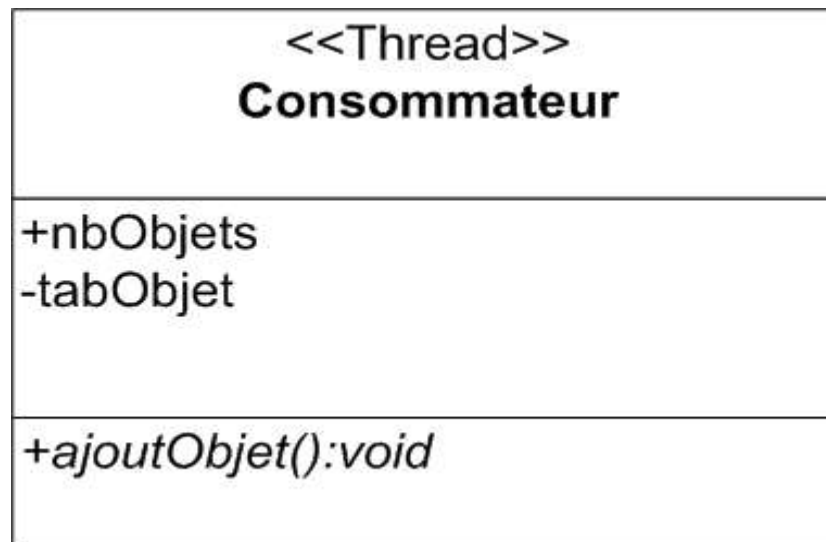


Diagramme d'objets

description d'instances objets dans un cas particulier

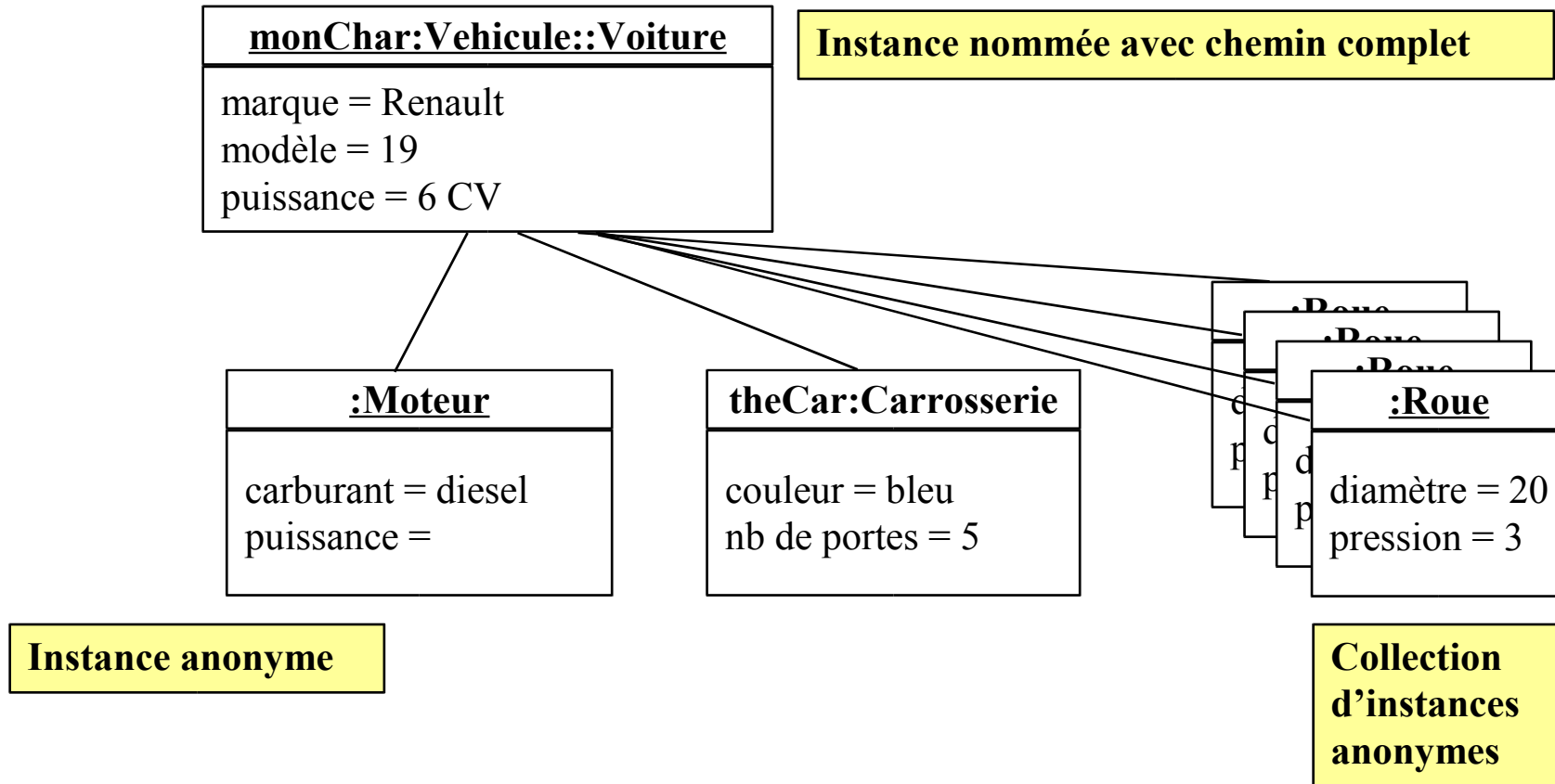


Diagramme d'objets : exemple

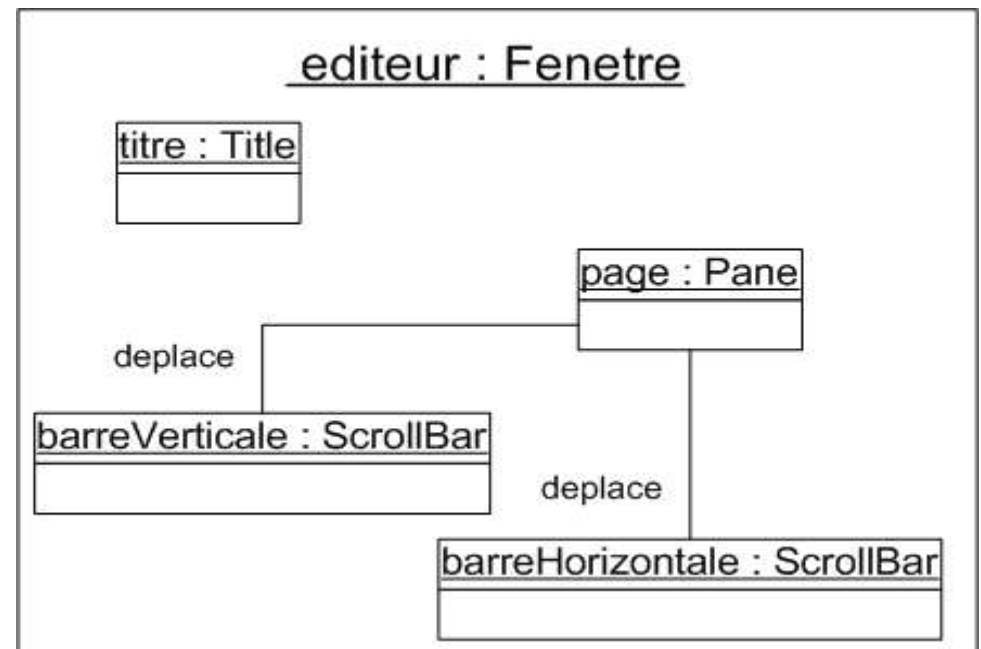
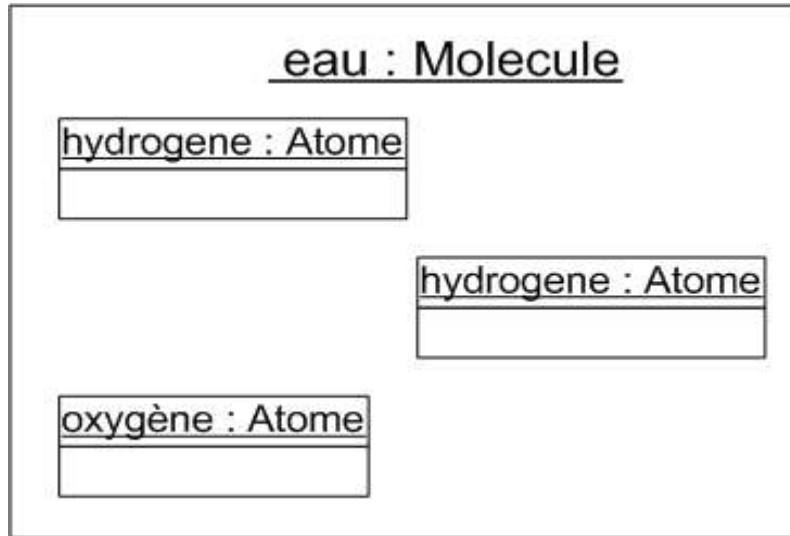
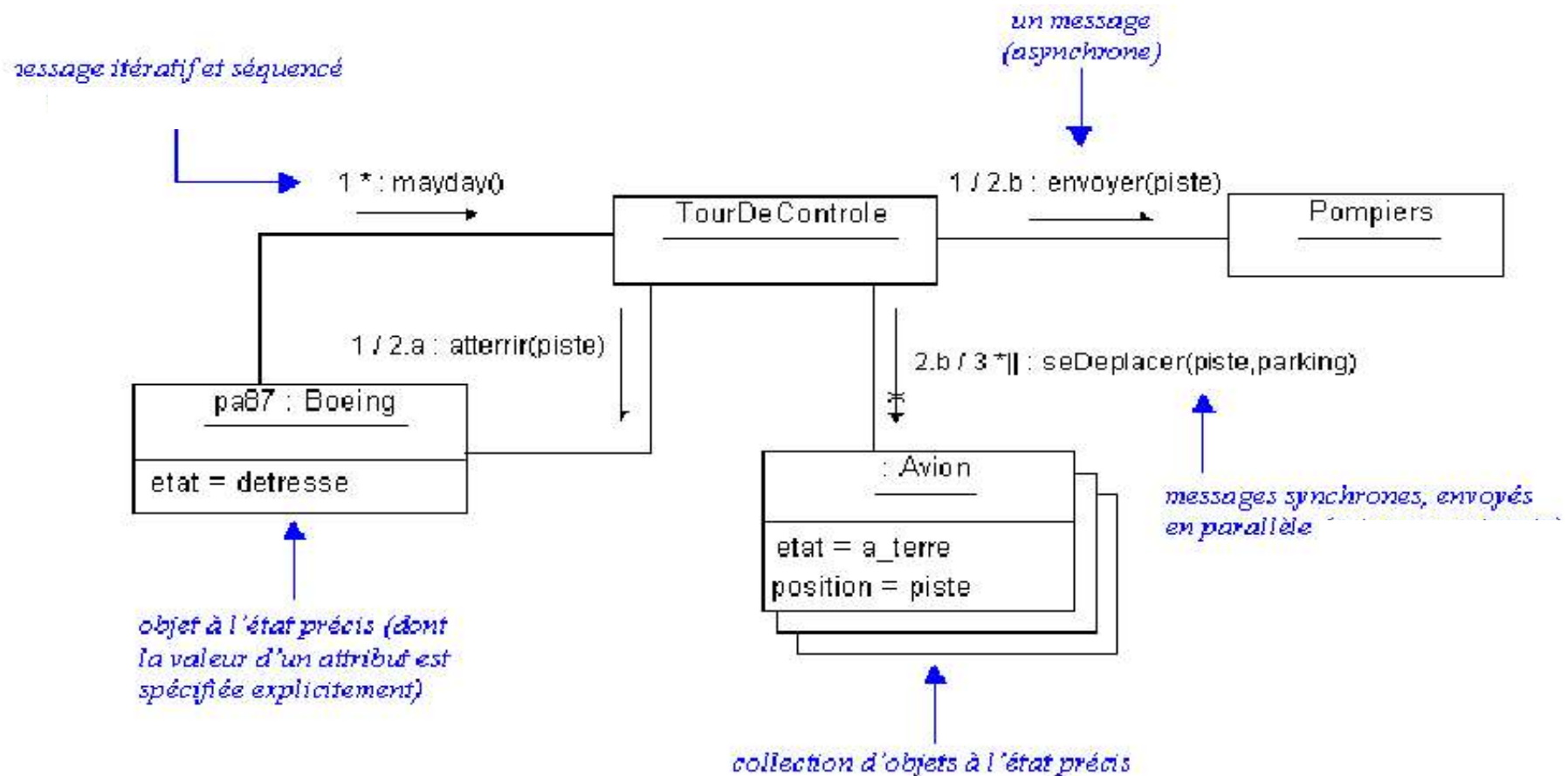


Diagramme de collaboration

- Ils indiquent les interactions entre objets et entre les objets et les acteurs



Interactions : messages (1)

- Il est possible de détailler un message selon le schéma suivant :
`[pré "/"] [["cond"]] [séq] ["*" ["||"] [["iter"]]] ":" [r ":="] msg (" [par] ")`
- **pre** est la liste des messages devant être traités avant le msg envoyé
- **cond** est une condition booléenne sur l'envoi du msg
- **seq** est le numéro d'ordre du msg
- **iter** indique si le message est séquentiel, récurent * ou émis en parallèle ||
- **r** indique de nom de la variable de retour du msg
- **msg** est le nom du message
- **par** sont les paramètres du msg

Interactions : messages (2)

Exemples :

- 3 : bonjour()
- [heure = midi] 1 : manger()
- 1.3.6 * : ouvrir()
 - msg envoyé plusieurs fois
- 3 / *||[i := 1..5] : fermer()
 - Envoi de 5 msg en parallèle après le msg n°3
- 1.3,2.1 / [t < 10s] 2.5 : age := demanderAge(nom,prenom)
 - Envoi du msg après les msg 1.3 et 2.1 si t<10
- 1.3 / [disk full] 1.7.a * : deleteTempFiles()
- 1.3 / [disk full] 1.7.b : reduceSwapFile(20%)
 - Msgs envoyés après 1.3, si disk full est vrai. Les deux msgs sont envoyés simultanément (car ils portent le même numéro). Plusieurs msg de type 1.7.a peuvent être émis

Interactions : messages (3)

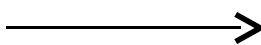
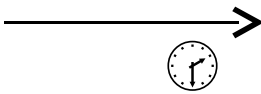
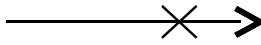
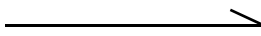

- Il existe différents types de msg, de stimuli
 - **message simple** 
 - **message minuté** : bloque l'émetteur en attendant la prise en compte du msg, le libère au bout d'un certain temps 
 - **message synchrone** : bloque l'émetteur en attendant la prise en compte du msg 
 - **message asynchrone** : n'interrompt pas l'émetteur, le msg peut être traité ou ignoré 
 - **message déroband** : n'interrompt pas l'émetteur, le msg est pris en compte que si le destinataire était en attente de ce msg 

Diagramme de séquences (1)

description des
séquences
d'événements,
états et réactions
qui doivent
survenir dans le
système (modèle
dynamique)

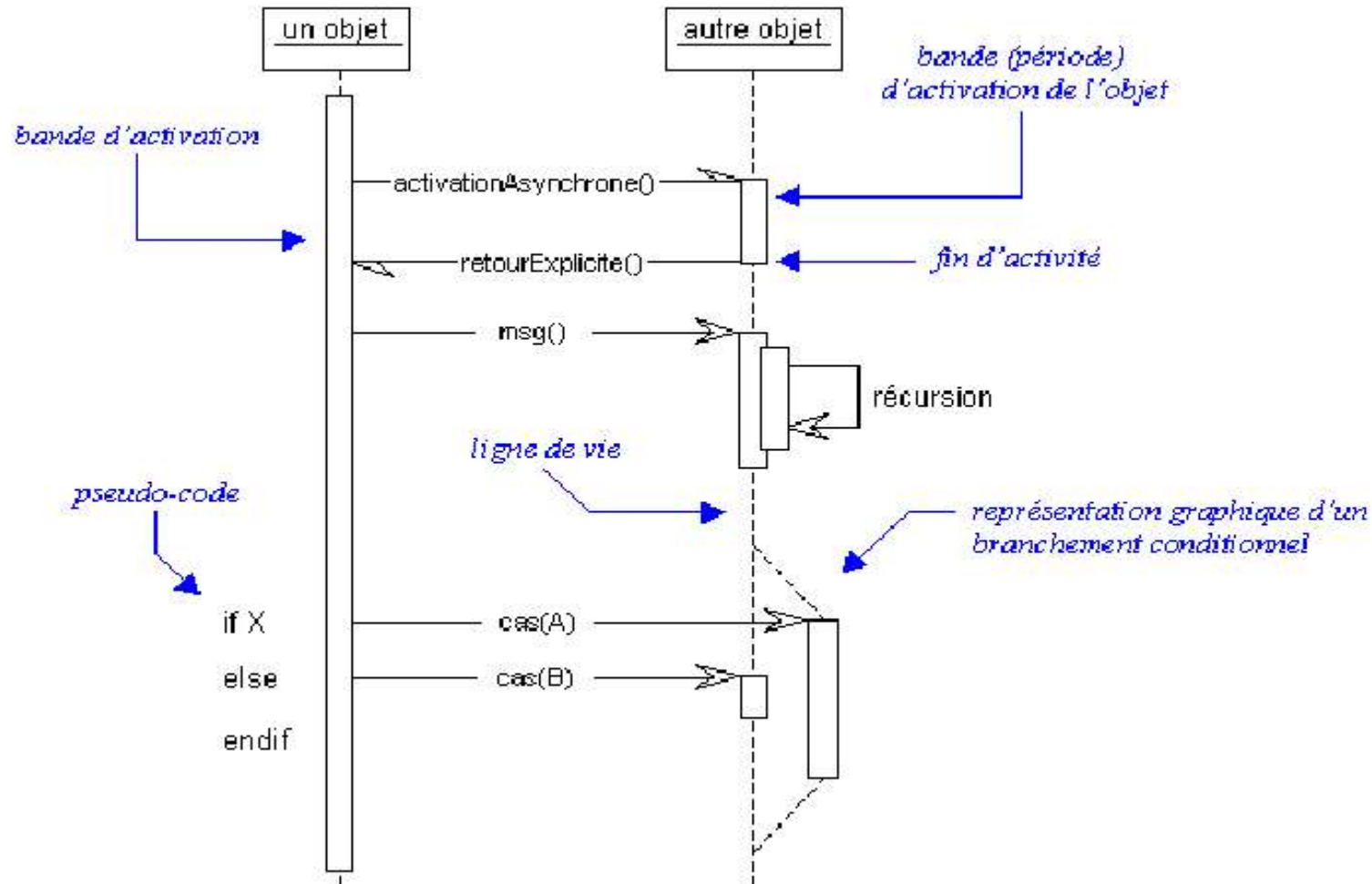
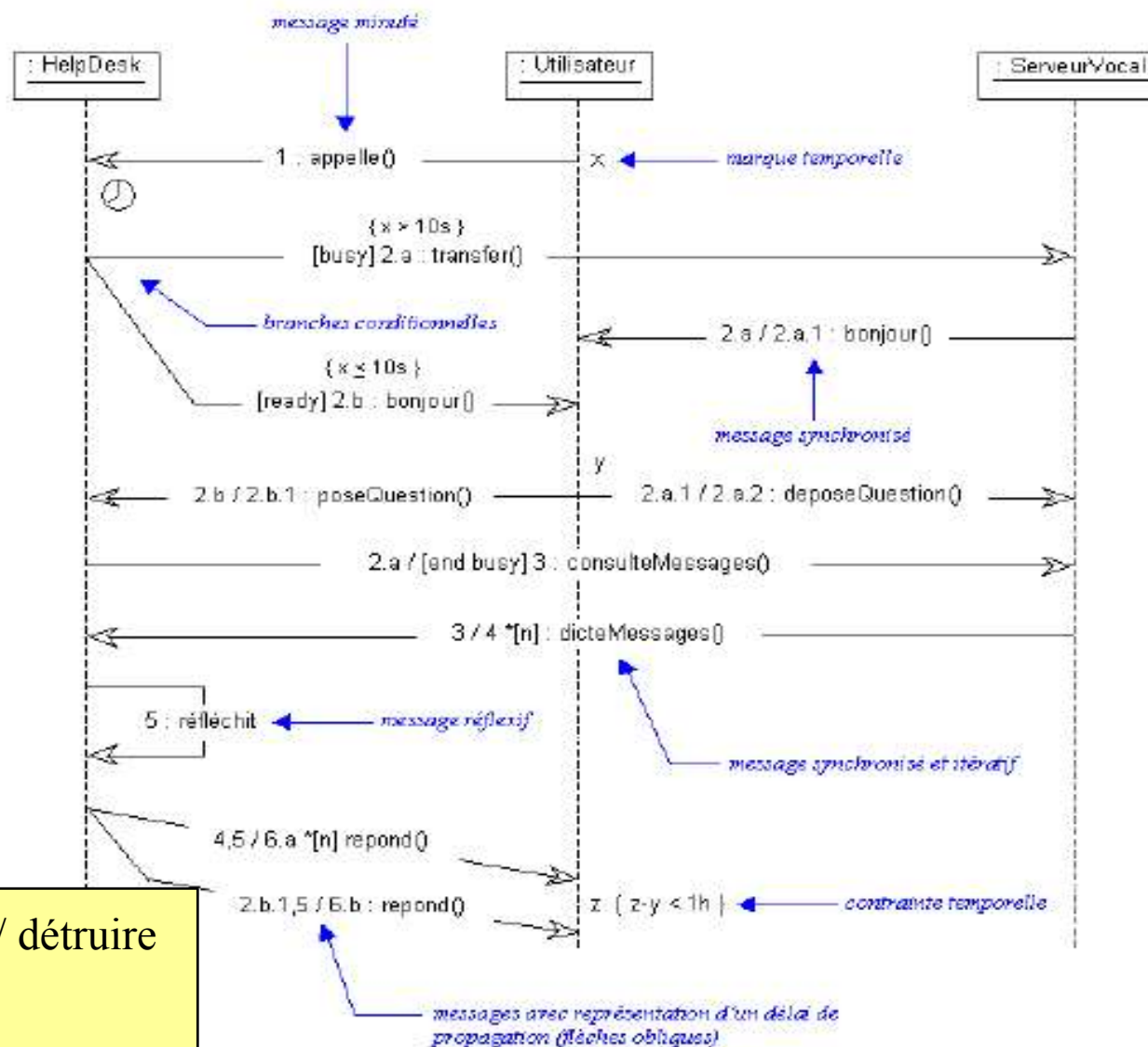


Diagramme de séquences (2)

- Pour chaque classe : choisir le diagramme de séquence le + important et en déduire un diagramme d'états
- utiliser la *hiérarchie* et le *parallélisme* pour simplifier
- attention aux boucles : reconnaître les états identiques
- vérifier les entrées-sorties des événements
- attention aux états sans précédent ou sans suivant
- tenir compte des délais, ex. : si pas d'acquiescement d'une alarme au bout de 15mn alors passage à un nouvel état
- ajuster et valider les modèles en comparant les opérations et attributs présents dans le modèle d'états et ceux présents dans la classe

Diagramme de séquences (3)



Possibilité de créer / détruire un objet

Diagramme d'états-transitions (1)

description de l'évolution au cours du temps d'une instance d'une classe (cycle de vie d'un objet)

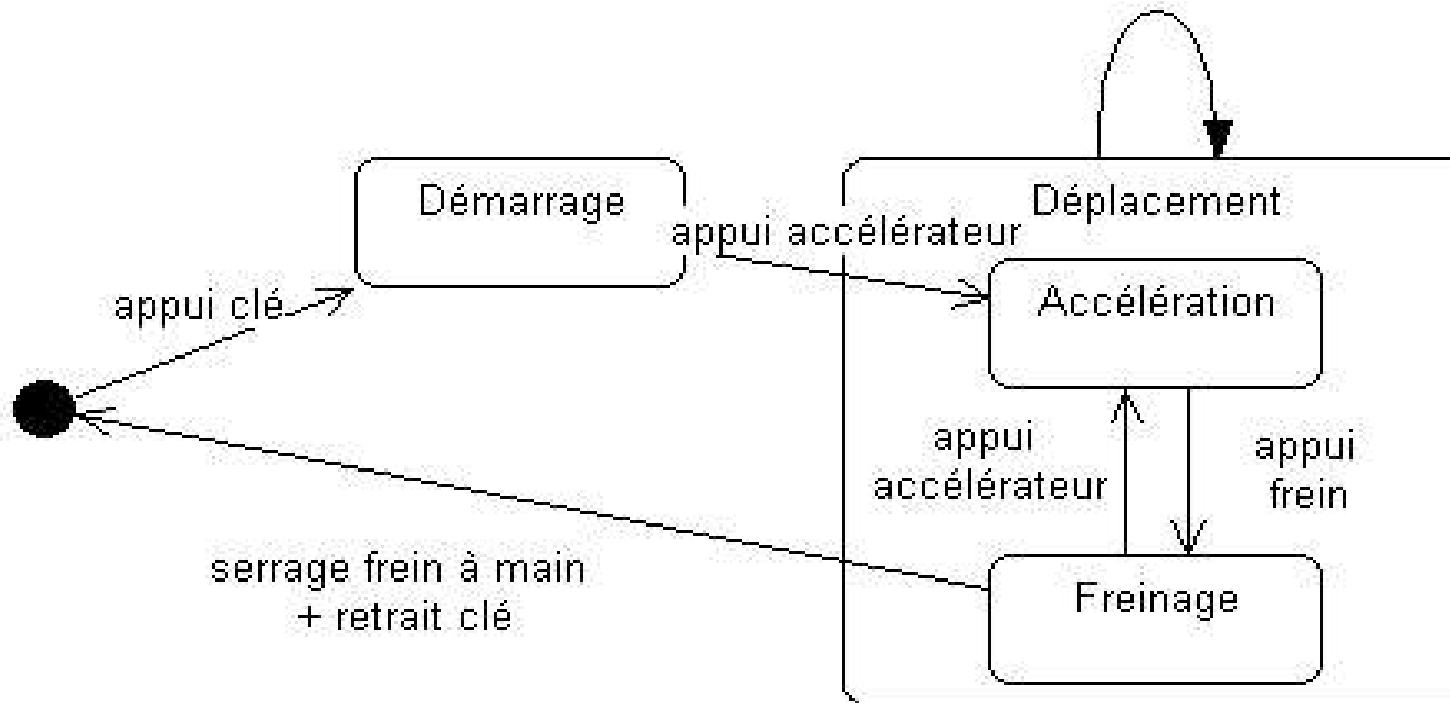
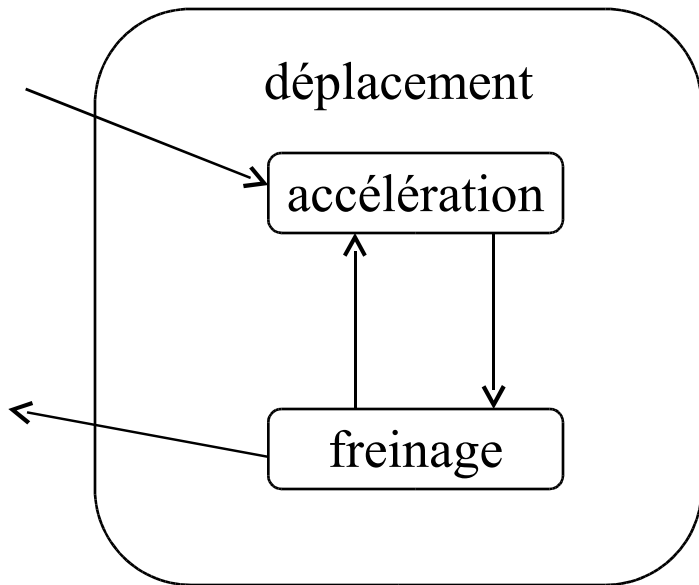
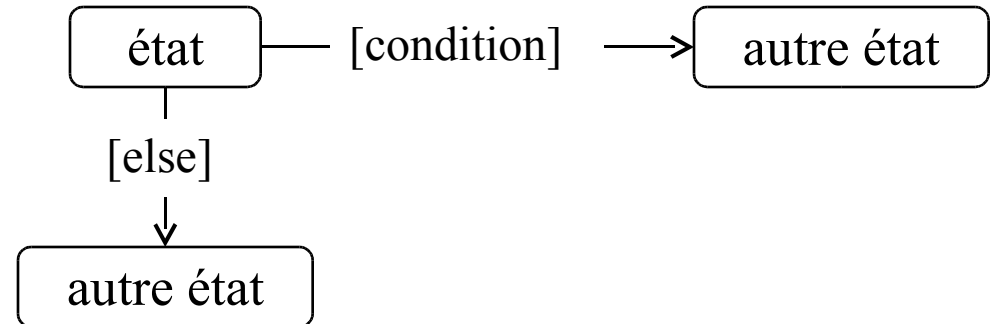
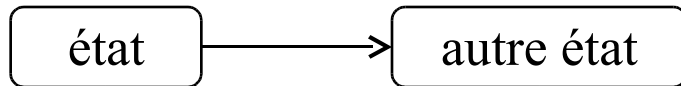
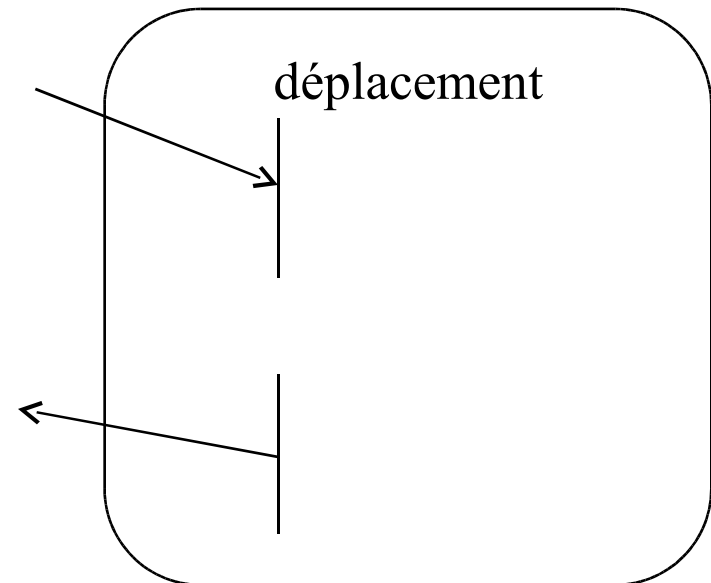


Diagramme d'états-transitions (2)

Eléments de sémantique :



Super Etat



Souches (pour simplifier l'écriture)

Diagramme d'états-transitions (3)

Ajout d'actions dans les états

État

entry / action : action exécutée à l'entrée de l'état

exit / action : action exécutée à la sortie de l'état

on événement / action : action exécutée à chaque fois que l'événement cité survient

do / action : action récurrente ou significative, exécutée dans l'état

Parallélisme

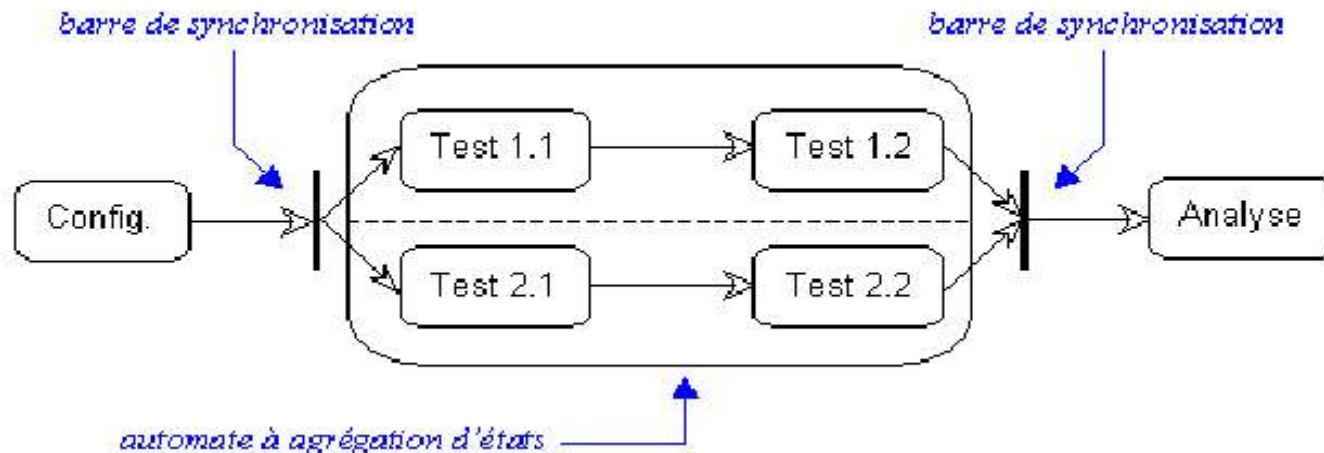


Diagramme d'états-transitions (4)

Evénements paramétrés

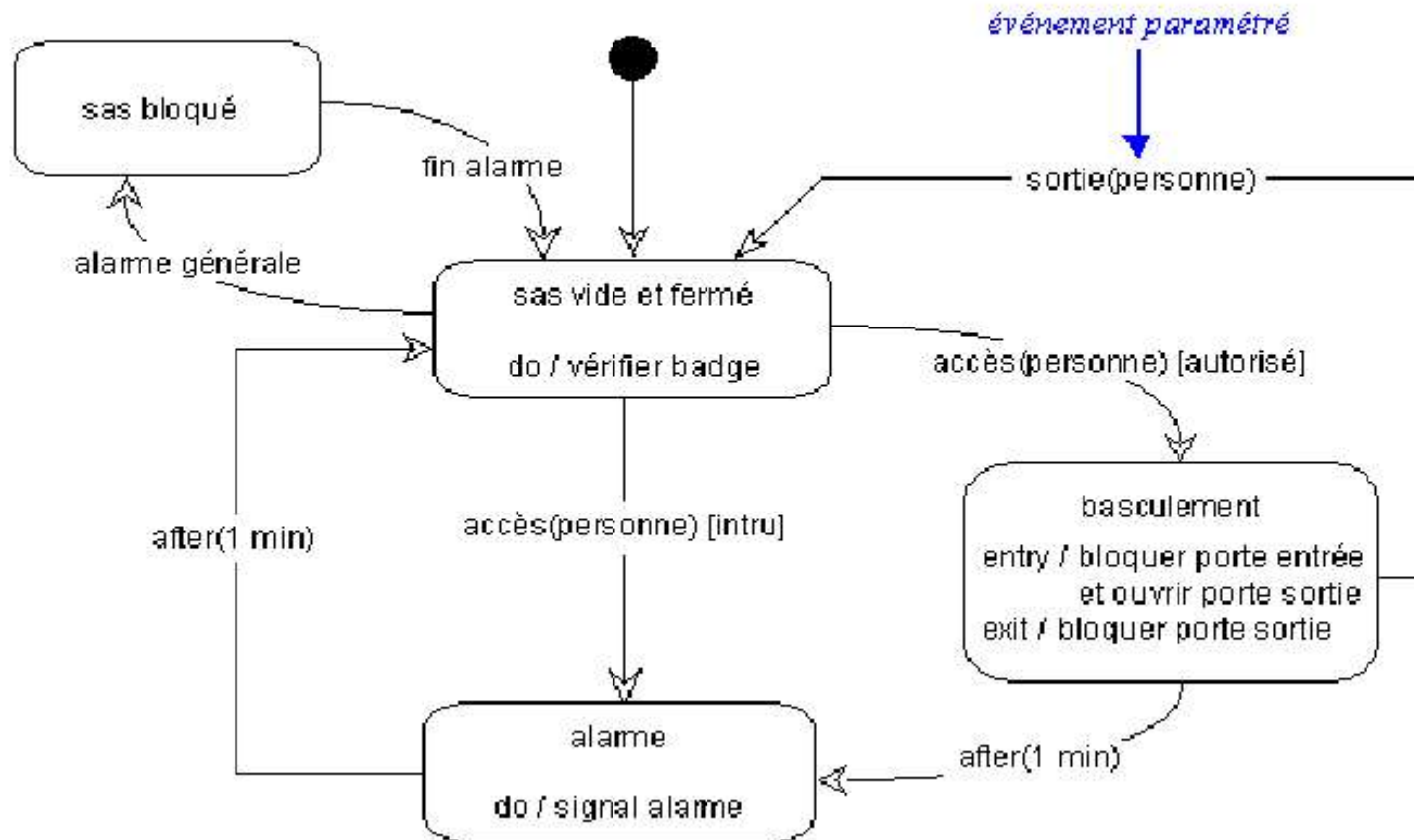
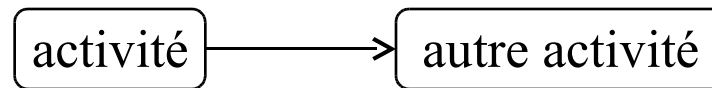
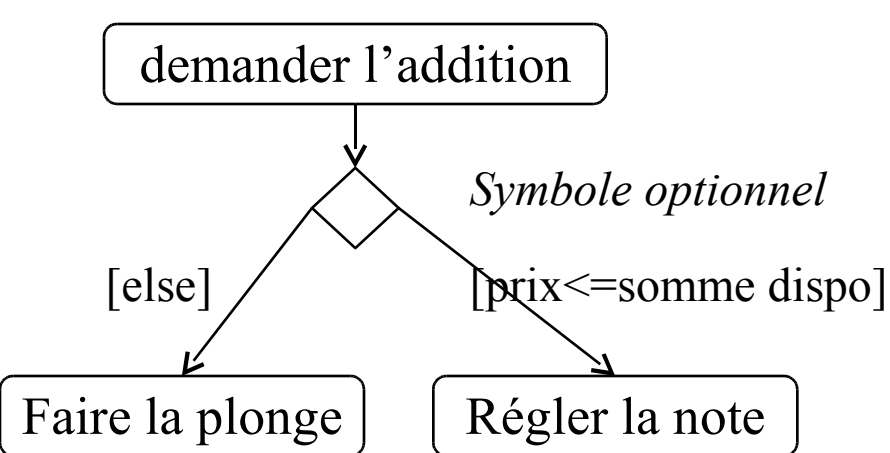


Diagramme d'activités

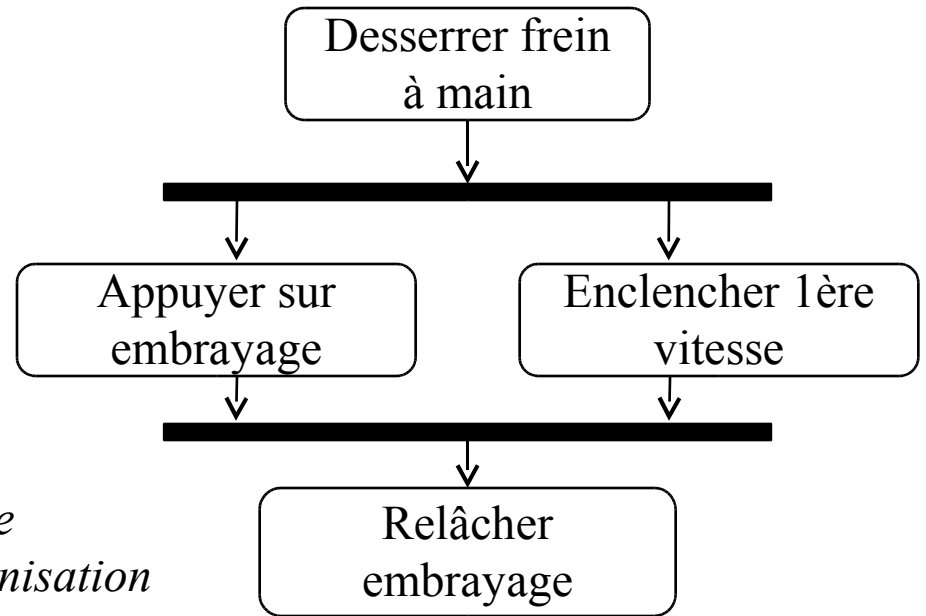
- Utilisé pour décrire l'algorithme d'une opération
- Focalisé sur les traitements internes et non sur la réception d'événements externes
- Sémantique :



La transition est automatique



Symbole optionnel



Barre de synchronisation

Vues de réalisation et des processus

- **Analyse**

- analyse du domaine

- diagramme de classes** obtenu par synthèse des ébauches précédentes et ajustements

- analyse de l'existant (matériel et logiciel)

- représentation par **diagramme d'objets, de classes...**

- **Conception de l'architecture : réalisation des modèles**

- architecture logicielle

- ajout éventuel d'objets, d'acteurs ...

- modification éventuelle des modèles précédents

- vue logique : **structures en paquetages** qui contiennent des classes

Vue de déploiement

- architecture matérielle

- disposition physique des éléments et de leurs relations (connexions)
- **diagramme de composants**
- **diagramme de déploiement**

- réalisation

- génération du schéma de la base de données à partir des classes du domaine
- génération des écrans pour les interfaces
- réalisation des interactions à partir des diagrammes de collaboration

Diagramme de composants (1)

Montre les éléments physiques et leurs dépendances



Stéréotype :

`<<executable>>` : composant exécutable

`<<library>>` (bibliothèque) : ensemble de ressources

`<<table>>` : composant d'une bdd

`<<file>>` (fichier) : données ou code source

`<<document>>` : texte, ...

`<<application>>` : composant directement en contact avec l'utilisateur

...

Diagramme de composants (2)

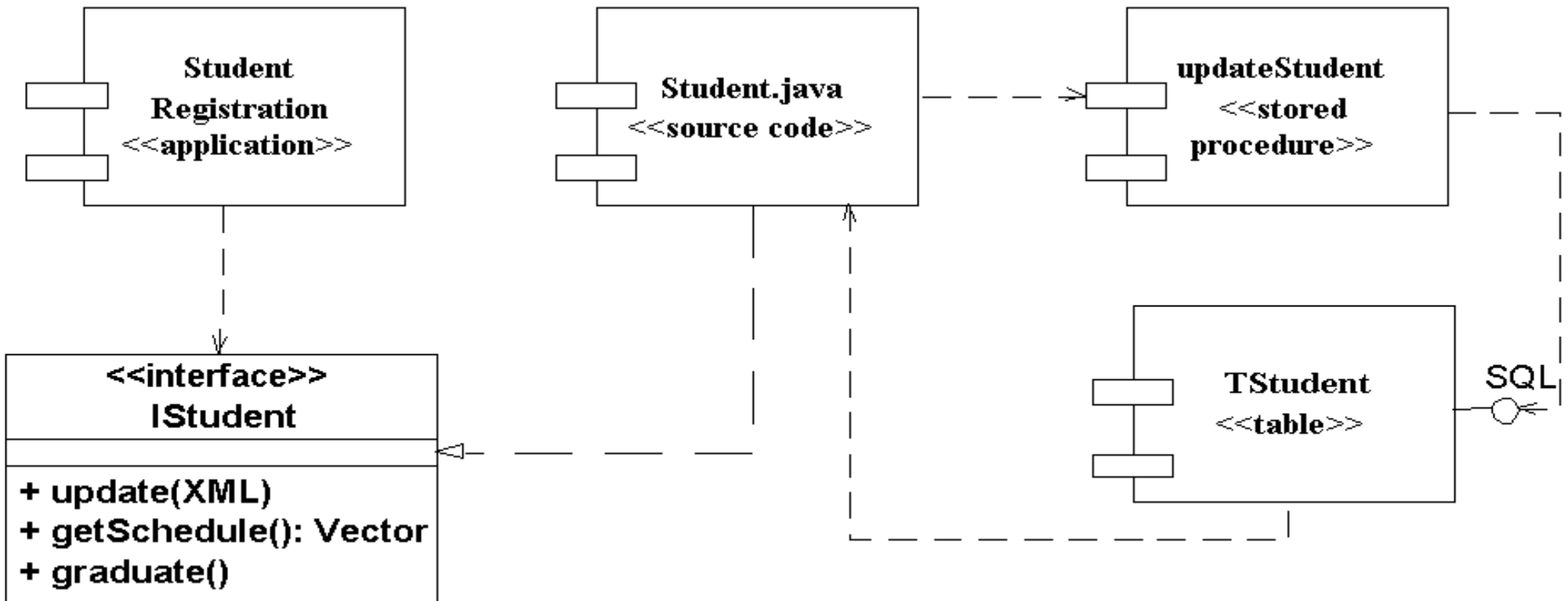


Diagramme de déploiement (1)

Description de la configuration matérielle du système
contient des noeuds : les ressources

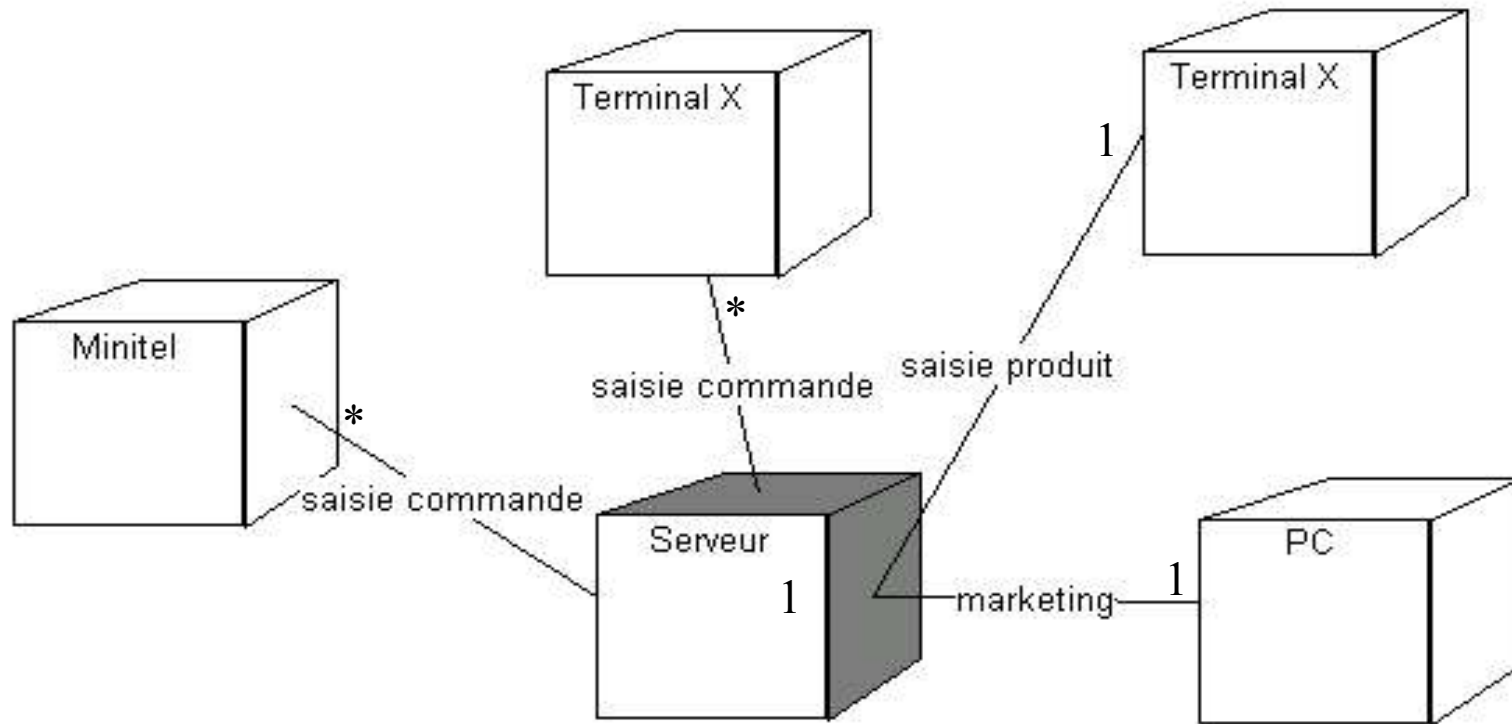


Diagramme de déploiement (2)

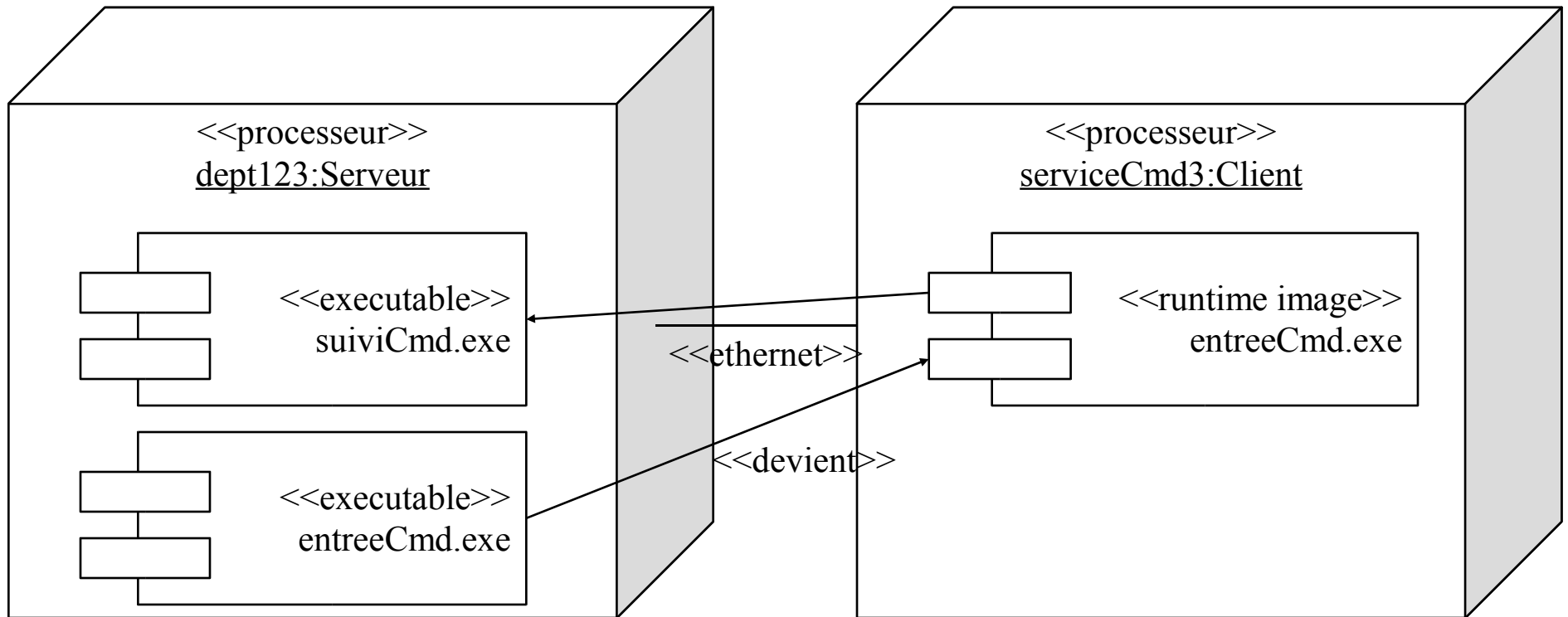
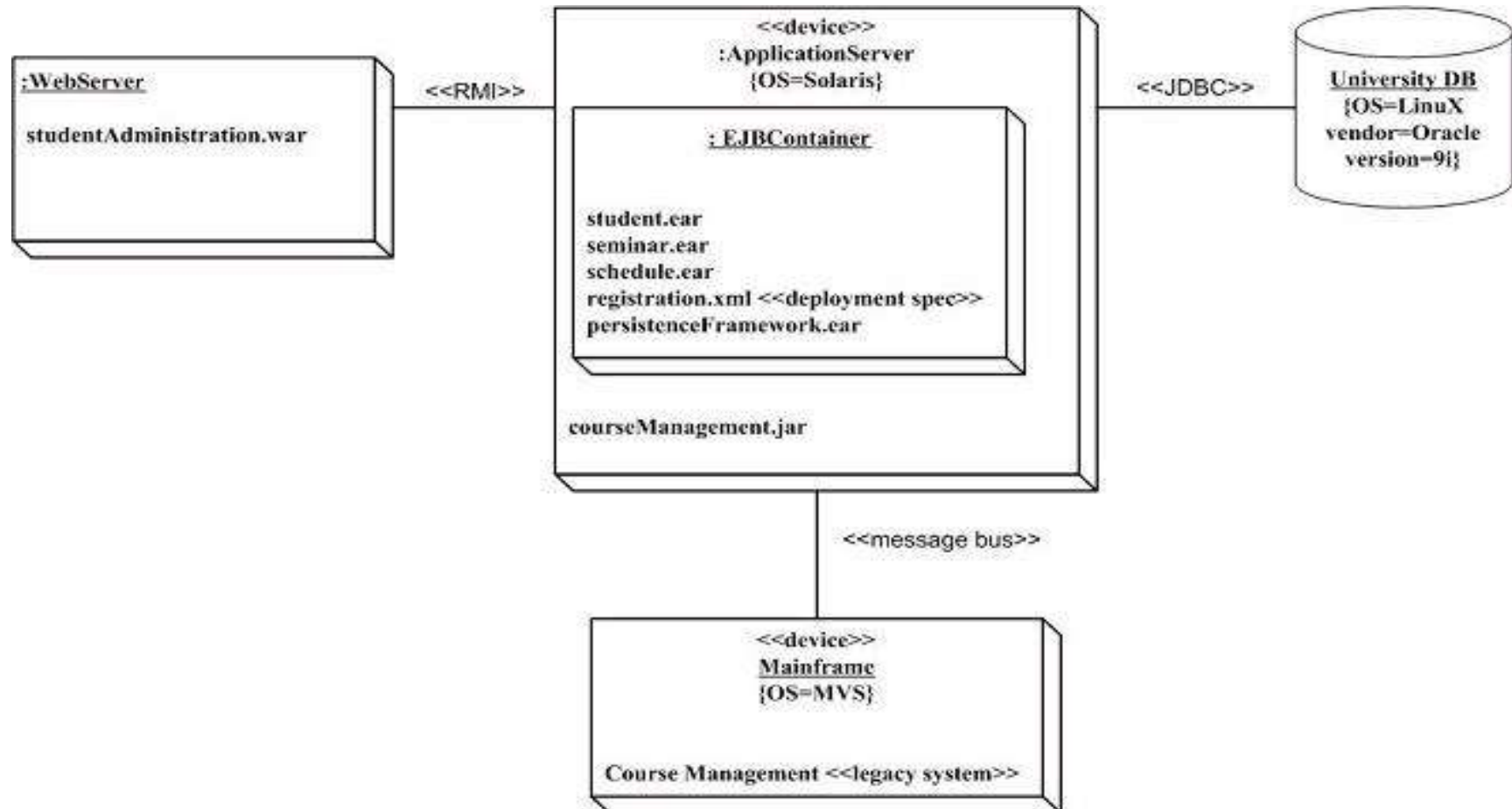


Diagramme de déploiement (3)



Correspondances entre diagrammes

- les modèles sont complémentaires : il est difficile de valider un diagramme isolément
- ils doivent être utilisés et affinés de manière parallèle
- progression du modèle le + abstrait et le + général vers les modèles les + concrets (techniques) et les + détaillés

Livres et Revues

- «Modélisation objet avec UML», P.A. Müller, Eyrolles, 1997
- «De Merise à UML», Kertani et al., Eyrolles, 1998
- «Penser objet avec UML et Java», M. Lai, Interéditions, 1998
- «UML and C++: a practical guide to object-oriented development», R. Lee, W. Tepfenhart, Prentice-Hall, 1997
- «Le génie logiciel orienté objet. Une approche fondée sur les cas d'utilisation», I. Jacobson, Addison-Wesley, 1993
- «Analyse et conception orientées objet», G. Booch, Addison-Wesley, 1994
- «OMT. Modélisation et conception orientées objet», J. Rumbaugh et al., Masson & Prentice Hall, 1995
- «Analyse des systèmes : de l'approche fonctionnelle à l'approche objet», P. Larvet, Interéditions, 1994
- «UML in Action», Communications of the ACM, vol. 42, n°10, Oct. 1999
- Ingénierie des Systèmes d'Information, vol.5, n°5, 1997.

Internet

Les sites suivants contiennent de la documentation sur UML:

- <http://www.coget.com/uml/>
- <http://www.objekttechnik.se/uml/>
- <http://www.rational.com/> (produit AGL UML : Rose)
- <http://www.ics.uci.edu/pub/arch/uml/> (produit AGL : Argo/UML)
- <http://uml.free.fr>
- <http://www.tcom.ch/Tcom/Cours/OOP/Livre/LivreOOPTDM.html>

Outils supportant UML

- LOREx2
- Rational: Rose
- Aonix: Software Through Pictures
- Cayenne: ObjectTeam
- Platinum Technologies: Paradigm Plus
- MetaCase: MetaEdit+
- Popkin: SA/Object Architect
- MarkV: ObjectMaker
- Tendril Software: StructureBuilder
- Select Software Tools
- Riverton Software: HOW
- Visible Systems: EasyER
- Adaptive Arts: Simply Objects
- ObjectiF
- Object International: Together/J
- Advanced Software Technologies: Graphical Designer Pro
- MagicDraw UML
- MicroGold Software: With Class
- Object Domain
- Object Insight: Javision
- Project Technology: BridgePoint
- Poseidon
- ArgoUML
- Visio
- Visual Paradigm for UML 2.2