

Le langage UML

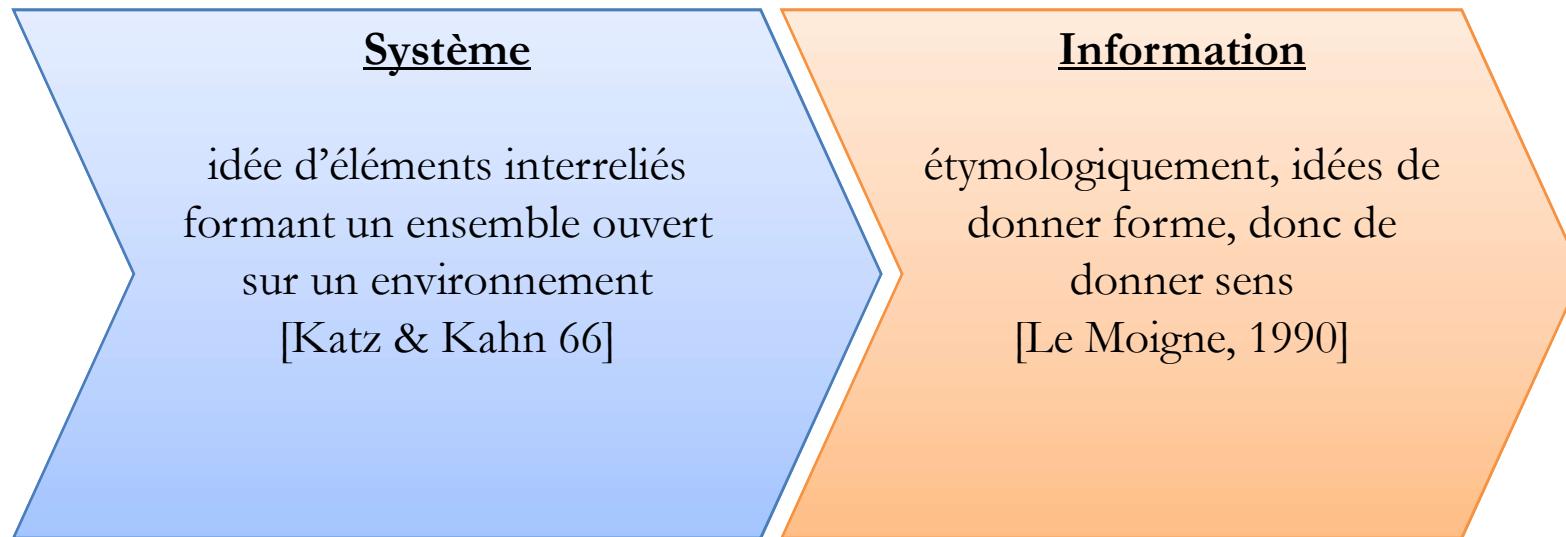
Faten Atigui

Maître de conférences – CNAM – Paris

Transparents préparés par Virginie Thion-Goasdoue

INTRODUCTION

Système d'information : Définition



Système d'information : Définition

[Reix 98] : « *Ensemble organisé de ressources : matériel, personnel, données, procédures permettant d'acquérir, de traiter, de stocker, de communiquer des informations (sous forme de données, textes, images, son, etc.) dans les organisations.* »



Moyens humains et techniques

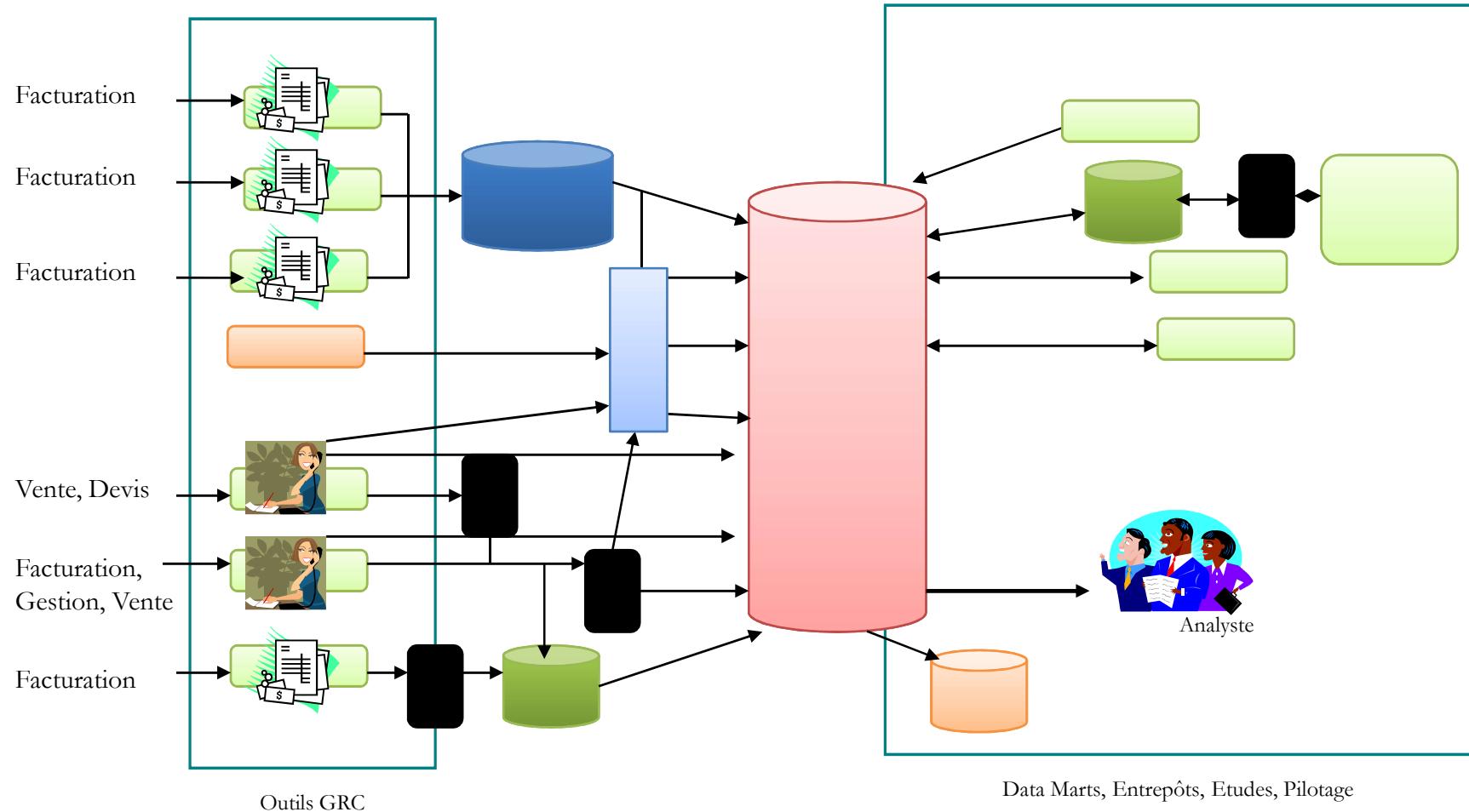
... inclut les procédures et méthodes manuelles

En entreprise, « système d'information » désigne souvent de façon abusive le « système informatique ».



Moyens informatiques

Exemple de Système d'Information



Ingénierie d'un système d'information

Ingénierie d'un SI

Concevoir (ou reconcevoir ou faire évoluer) et réaliser un système d'information.

Cours principalement focalisé sur la conception.

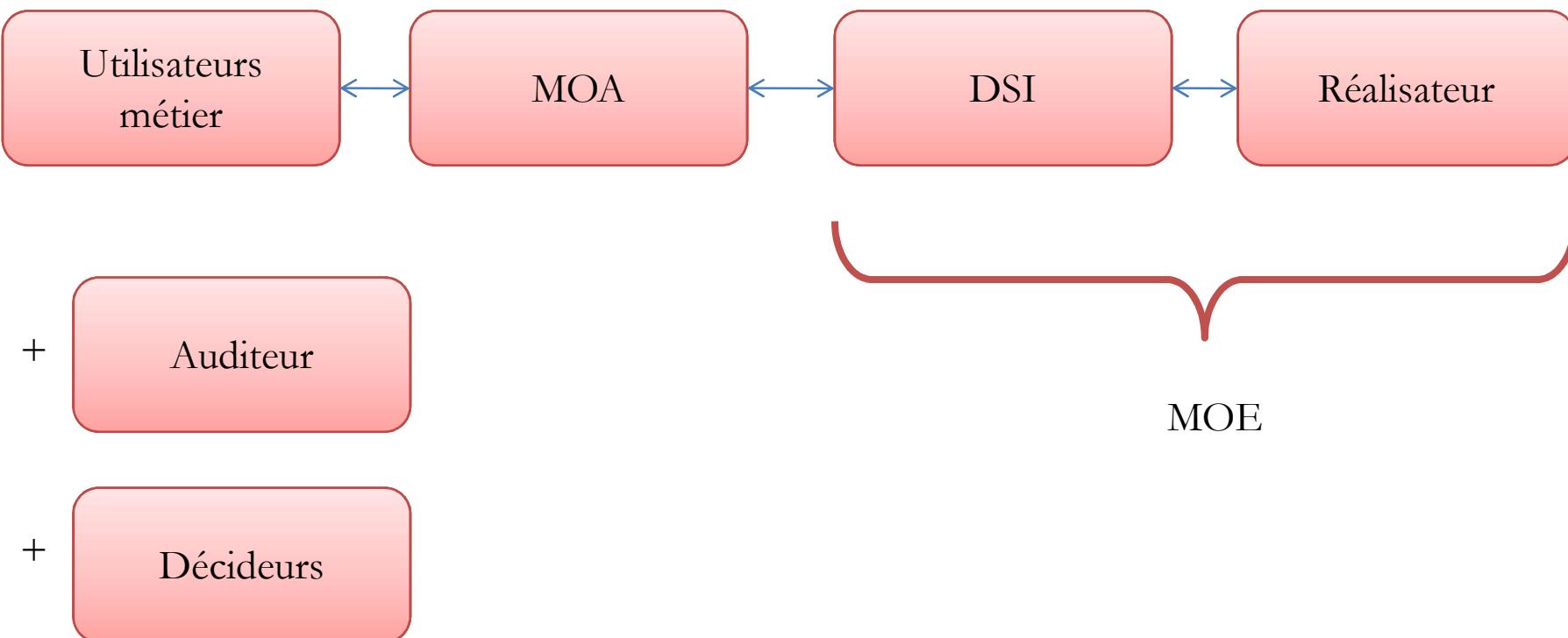
Conception

pour nous, obtenir un modèle du système d'information existant ou à mettre en œuvre.

Modéliser

Abstraction « formalisée » du monde réel

Qui intervient dans l'ingénierie d'un SI ?



Les systèmes d'information : MOA et MOE

MOA et MOE – ce sont des sigles couramment utilisés pour Maîtrise d’OuvrAge et Maîtrise d’OEuvre. Ce sont des entités organisationnelles.

Vocabulaire issu du BTP... MOA et MOE sont des termes empruntés au secteur des travaux publics.



MOA en construction - personne (morale) pour laquelle sont réalisés les travaux, entité porteuse des besoins.



MOE en construction - personne (morale) chargée par le maître d’ouvrage de concevoir le programme de restauration, de diriger l’exécution des marchés de travaux, et de proposer le règlement des travaux et leur réception.

Les systèmes d'information : MOE

MOE (Maîtrise d’OEuvre) - Réalisateur technique du projet, elle en conçoit la solution informatique. On peut voir la MOA comme son client. Elle est généralement composée de la DSI de l'entreprise et du réalisateur (des prestataires, sociétés de services, éditeurs et constructeurs).



Les systèmes d'information : MOA

MOA (Maîtrise d'OuvrAge) – Entité responsable de l'organisation et des méthodes de travail autour du SI, responsable de la bonne compréhension entre les métiers et la DSi.

⇒ La MOA se place « entre les métiers et la MOE ».

La MOA a entre les mains la décision, le financement, la structuration du projet métier. Mais à ne pas confondre avec les utilisateurs. La MOA est donneur d'ordre de la DSi.

Rôles MOA : décrit les besoins, le cahier des charges, établit le financement et le planning général des projets, fournit au MOE les spécifications fonctionnelles générales (le « modèle métier ») et valide la recette fonctionnelle des produits, coordonne les instances projets entre les utilisateurs métiers et la MOE, assure la responsabilité de pilotage du projet dans ses grandes lignes, adapte le périmètre fonctionnel en cas de retard dans les travaux, pour respecter la date de la livraison finale.



UML : (Unified Modeling Language)

pour qui ?

pourquoi ?

pour quoi ?

POURQUOI UML ?



Processus de développement logiciel

Processus de développement logiciel : séquence d'étapes, en partie ordonnées, qui concourent à l'obtention (ou à l'évolution) d'un système logiciel.



Principes applicables
à l'ingénierie d'un SI

Deux axes :

- développement technique
- *gestion du développement* ← hors périmètre du cours

Processus unifié

Processus unifié (*unified process*) : processus de développement logiciel utilisant le langage UML.

Deux points de vue :

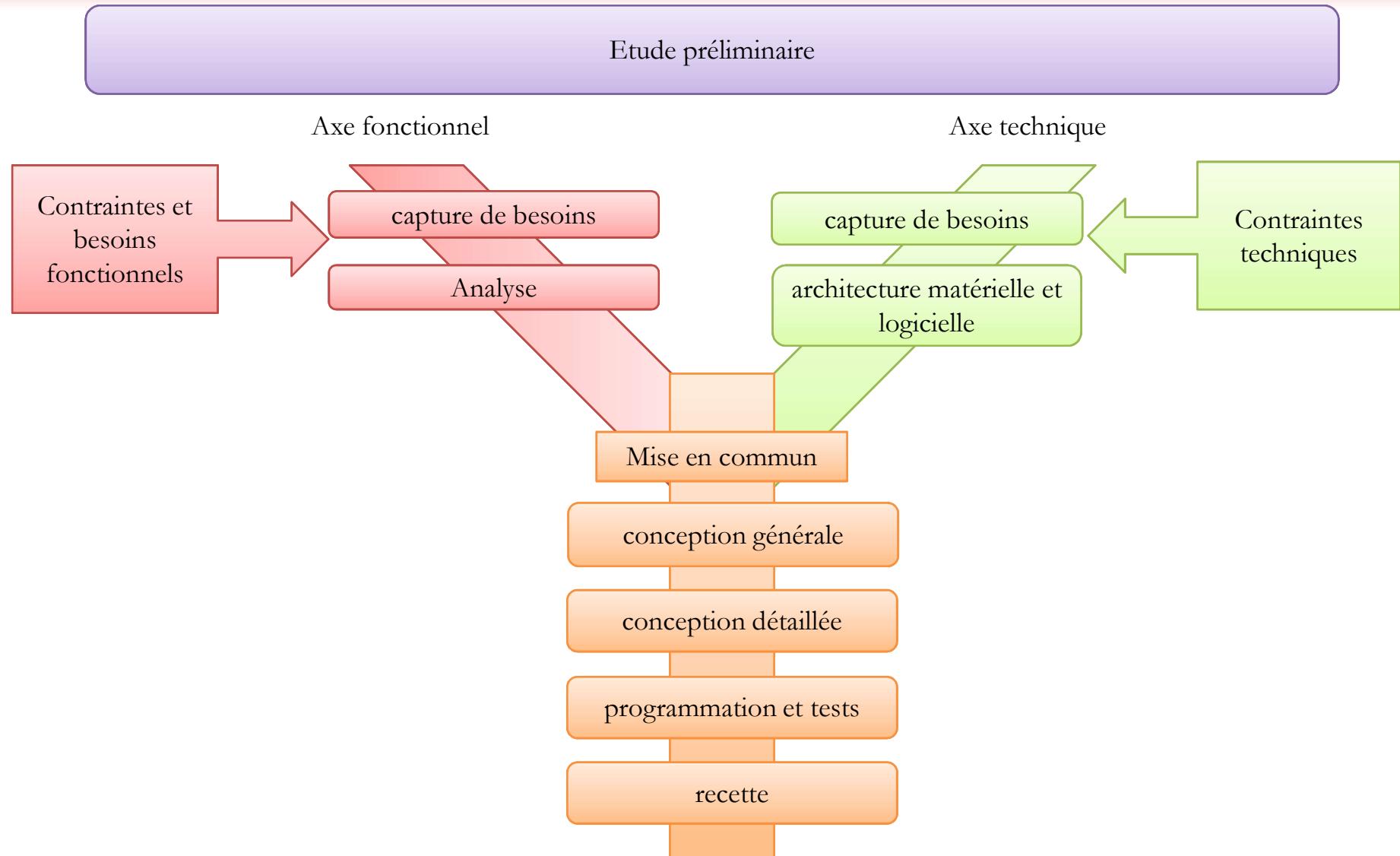
Fonctionnel – fonctionnalités que les utilisateurs attendent

la machine à café doit pouvoir faire couler du café, donner l'heure, elle doit aussi être programmable.

Technique – prise en compte des contraintes technique et intégration dans le SI de l'entreprise

la machine doit être de couleur inox pour s'accorder au frigo et au four, elle doit se brancher sur du 220 V.

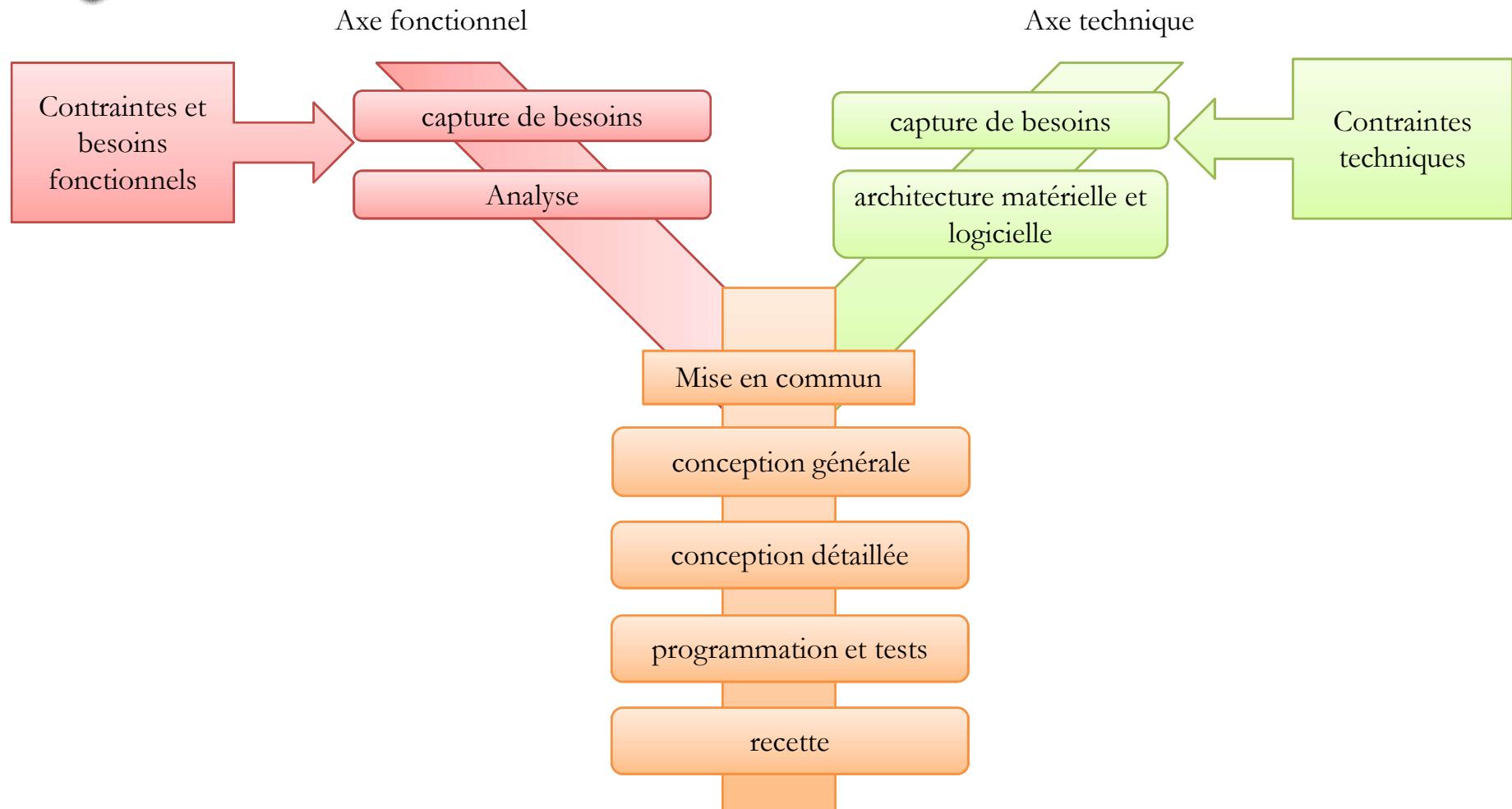
Processus unifié 2TUP (2 Tracks Unified Process)



Processus unifié 2TUP (2 Tracks Unified Process)

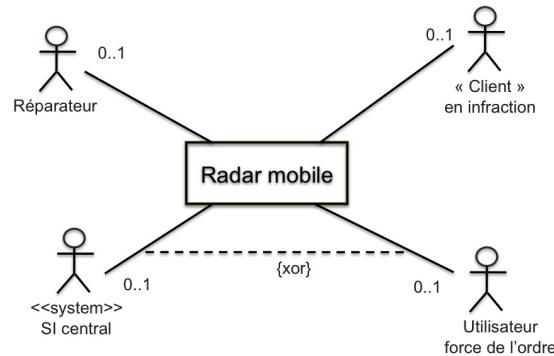
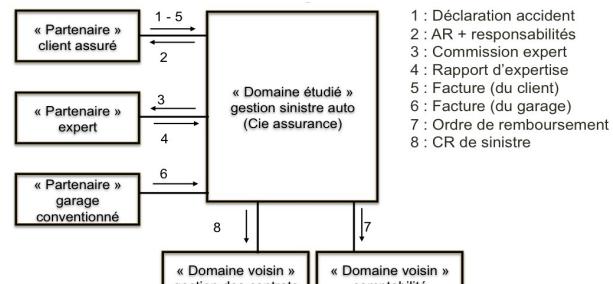


Etude préliminaire

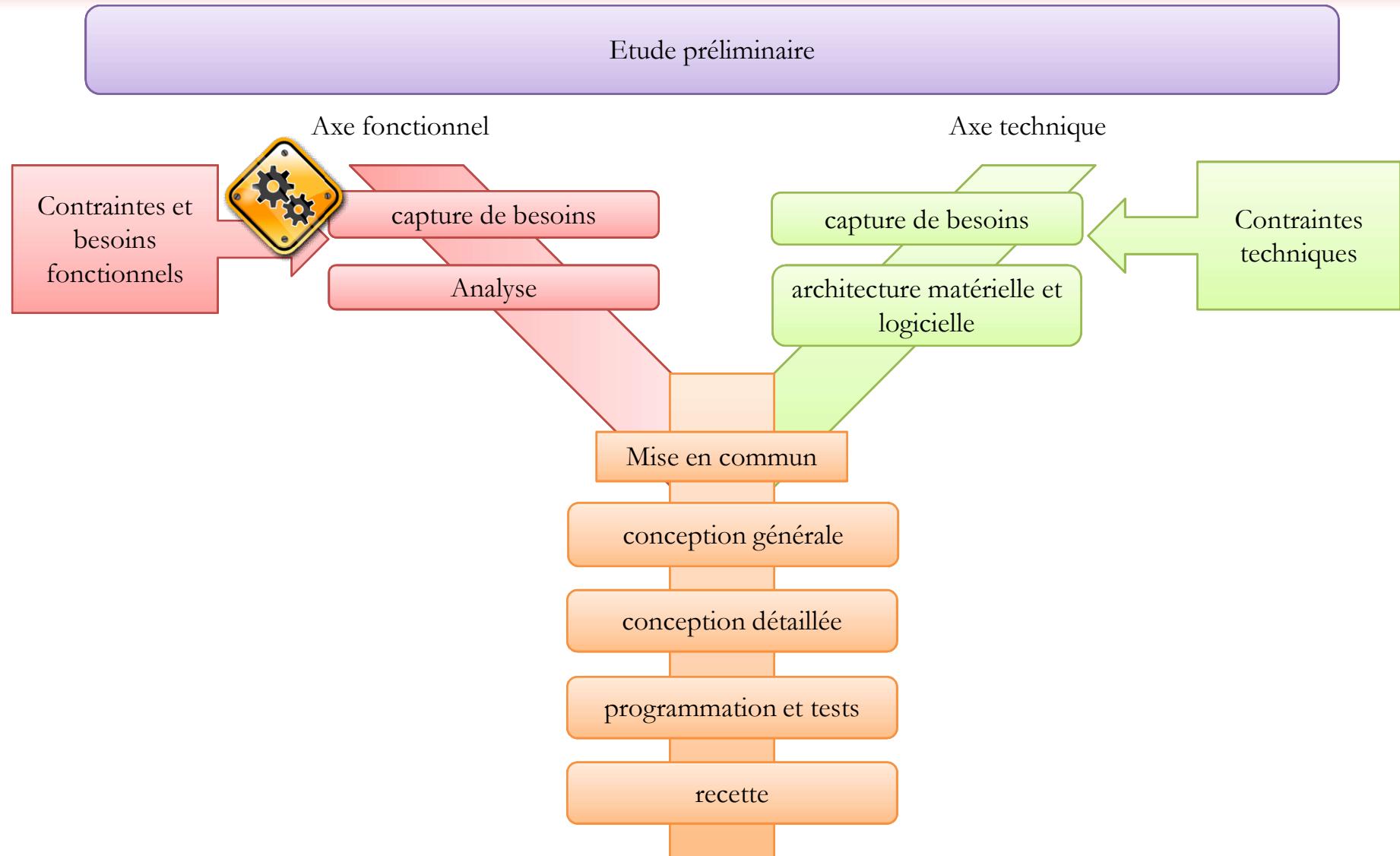


Etude préliminaire (cadrage)

- Appréhender et décrire le contexte : les acteurs, les problèmes, les interactions acteur/système (éventuellement via un audit du système actuel),
- Donner les premières orientations fonctionnelles et techniques,
- Définir un premier périmètre du système,
- Définir des priorités.

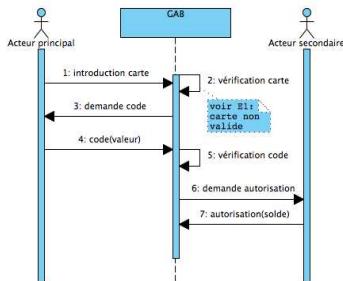


Processus unifié 2TUP (2 Tracks Unified Process)

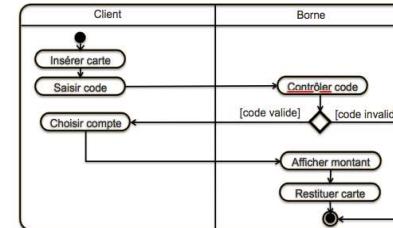


Capture des besoins fonctionnels : Les cas d'utilisation

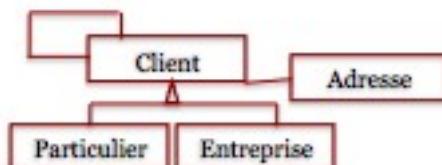
Recueillir de besoins utilisateurs



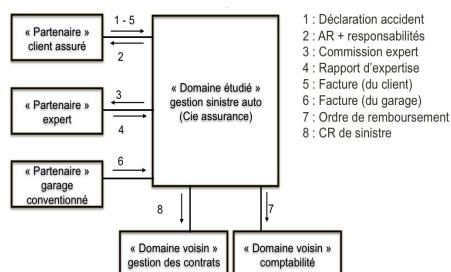
diag. de sequence système



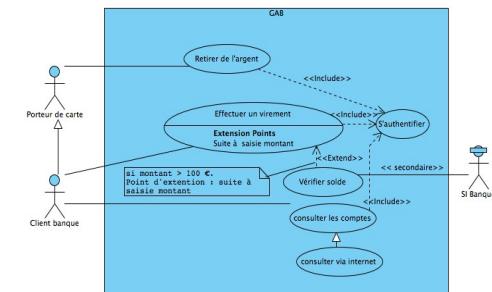
diag. d'activités ou d'états



diag. de classes

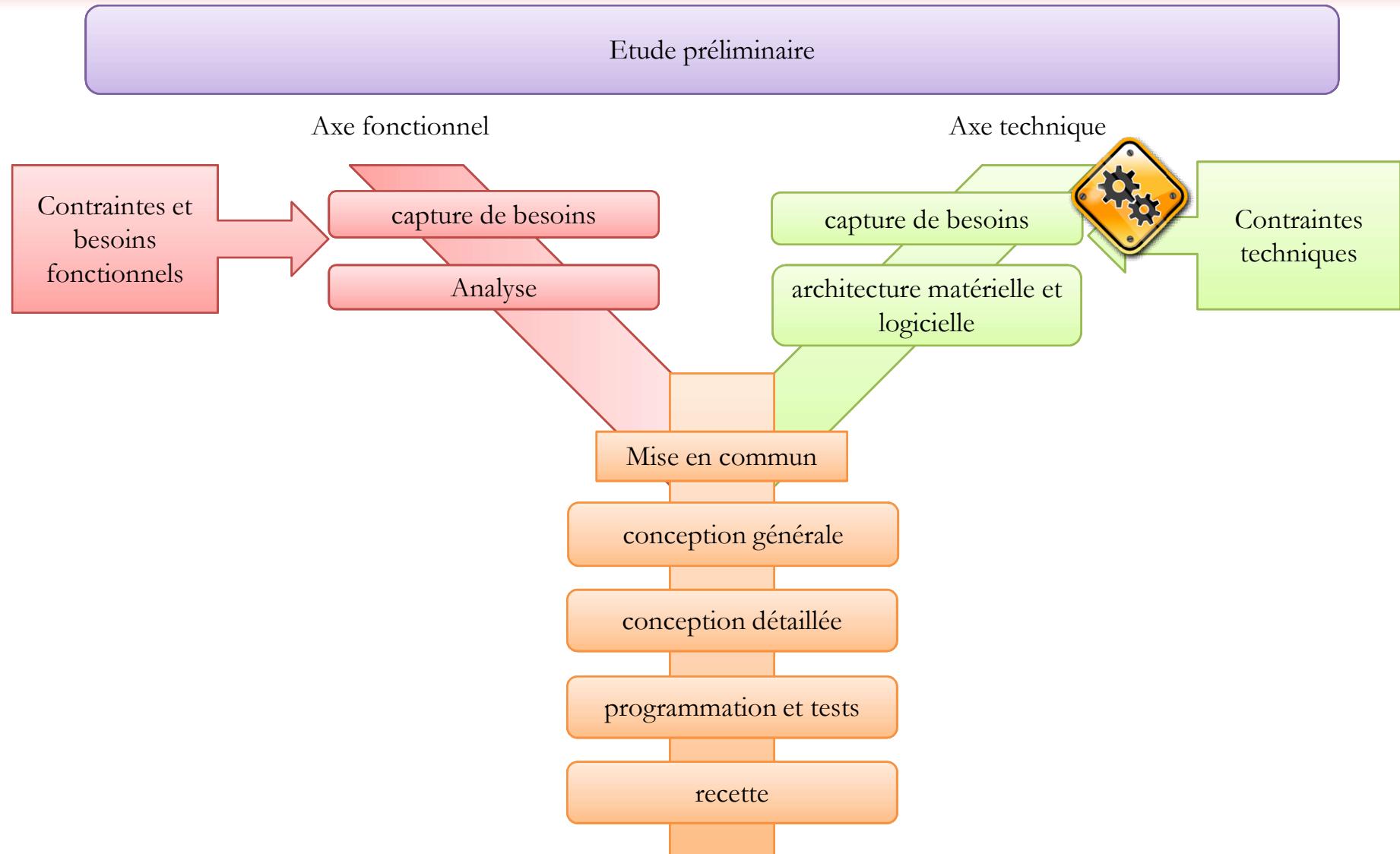


diag. de communication



diag. de cas d'utilisation

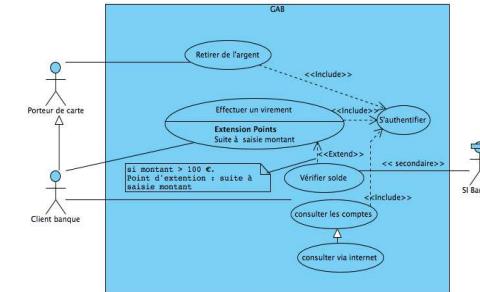
Processus unifié 2TUP (2 Tracks Unified Process)



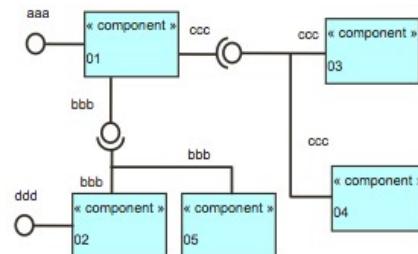
Capture des besoins techniques

Recueil des besoins et contraintes matérielles, logicielles

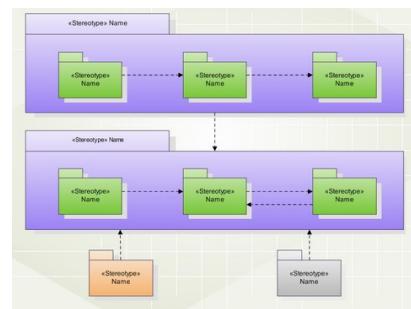
- Identification des CU techniques
- Configuration matérielle (serveurs, CP, etc)
- Premier diagramme de composants
- La description type d'architecture



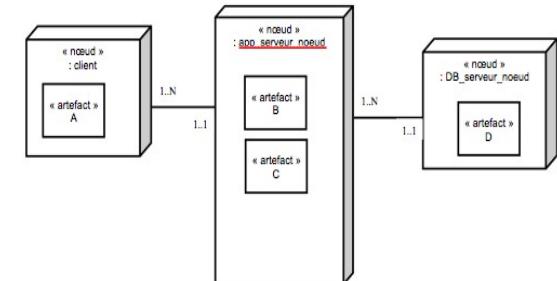
diag. de cas d'utilisation



diag. de composants

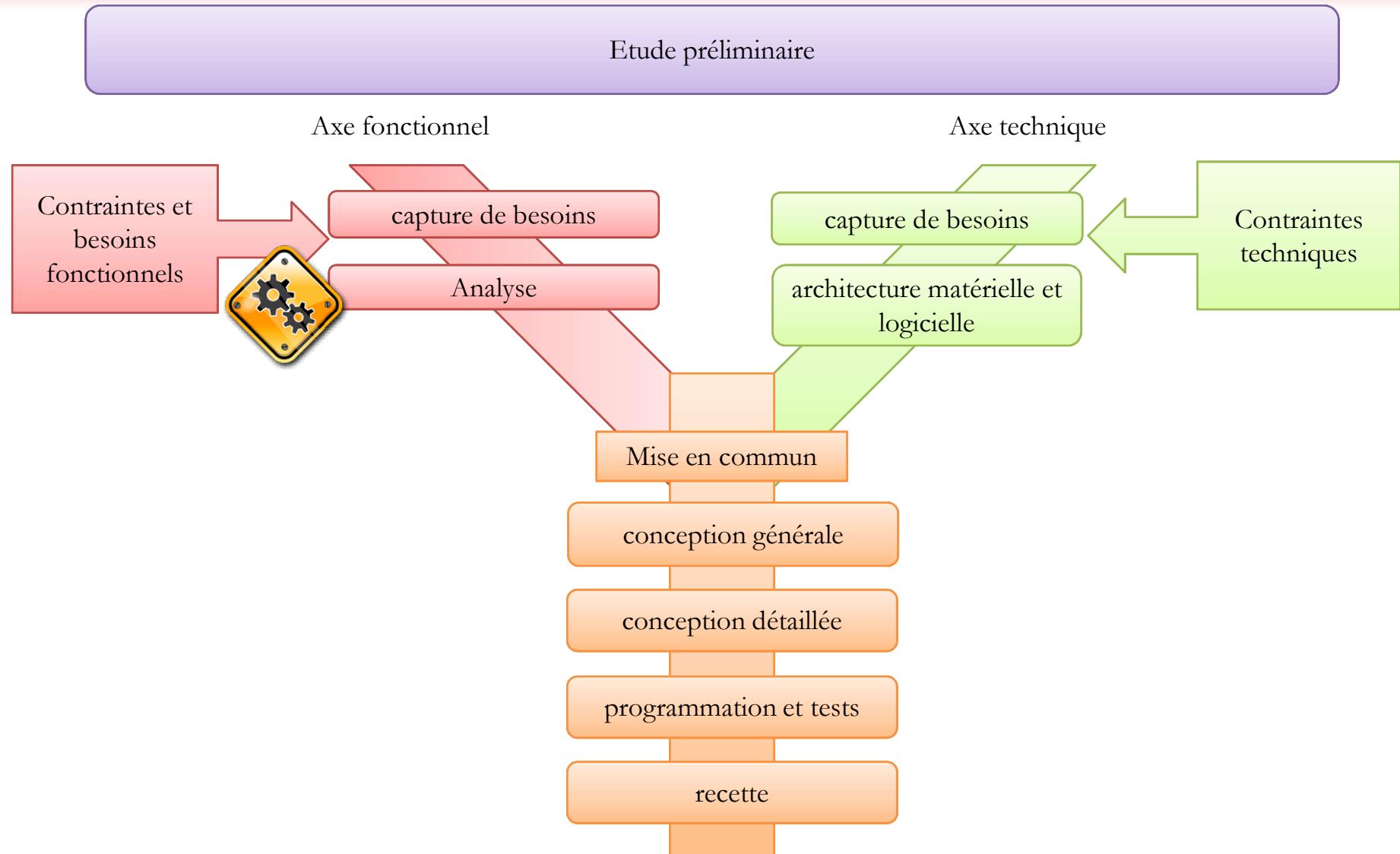


diag. de packages



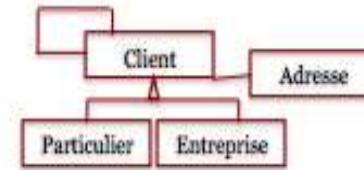
diag. de déploiement

Processus unifié 2TUP (2 Tracks Unified Process)



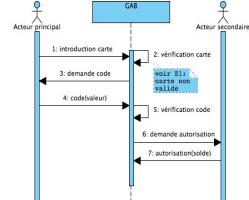
Analyse

- Développement du modèle statique

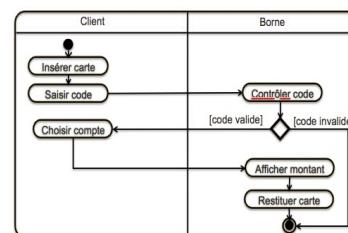


diag. de classes

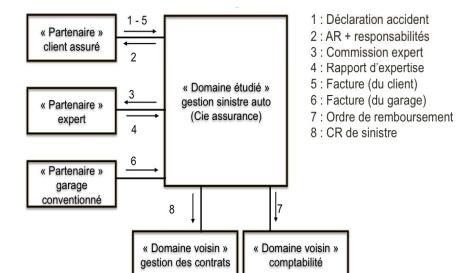
- Développement du modèle dynamique



diag. de séquence



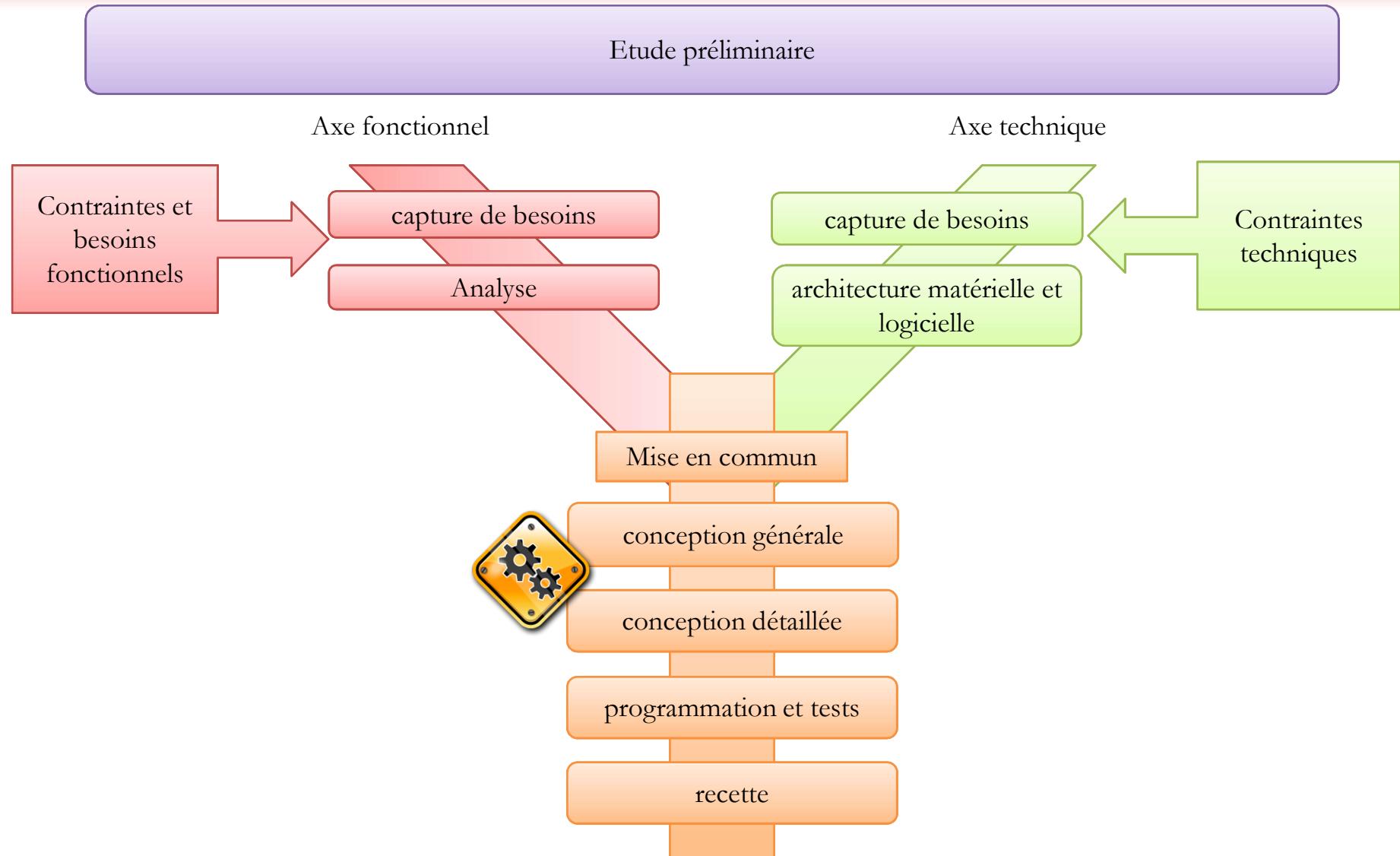
diag. d'activités ou d'états



diag. de communication

- Rapprochement des modèles

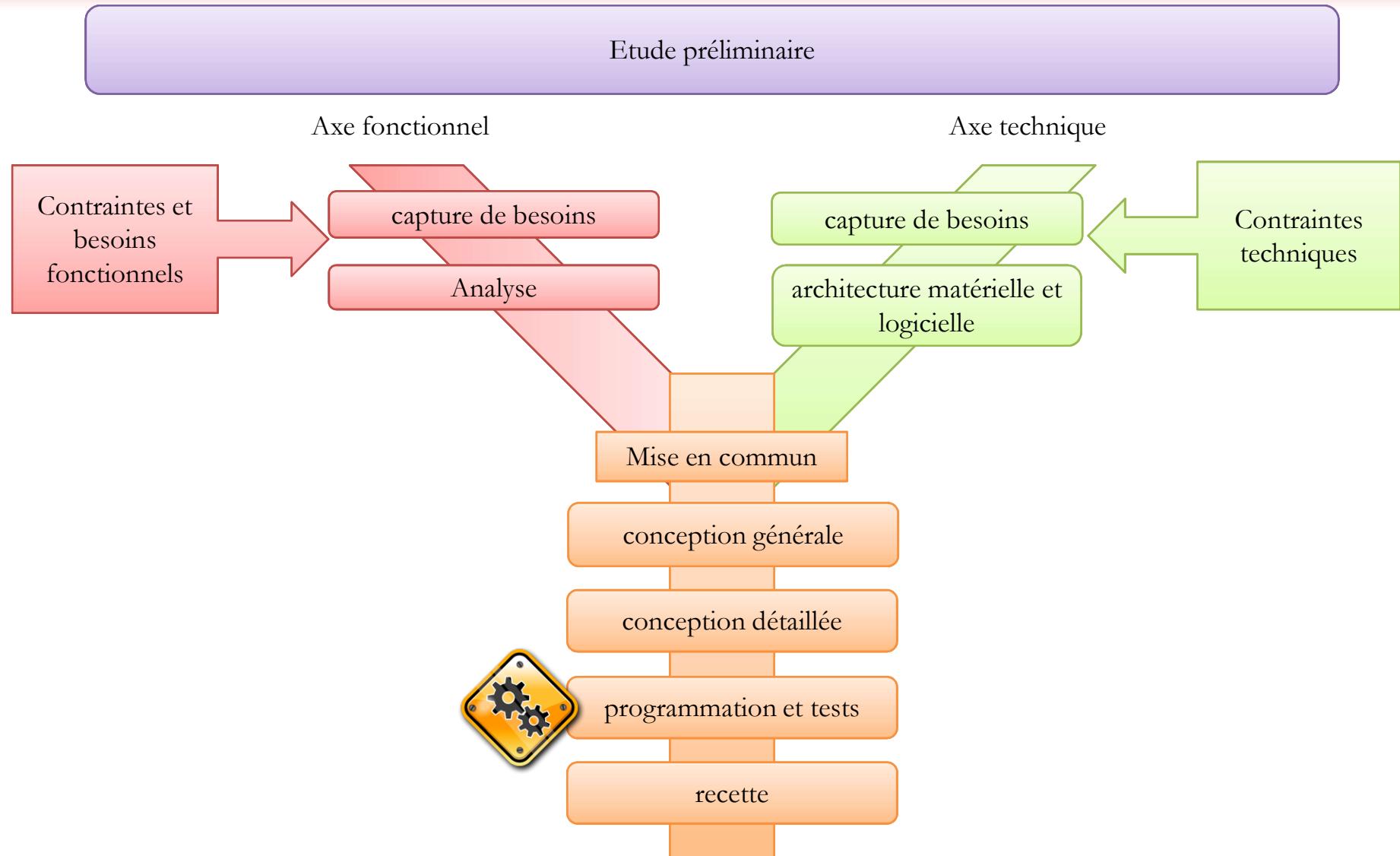
Processus unifié 2TUP (2 Tracks Unified Process)



Conception

Spécifications du futur système sous forme de documentation (éventuelle utilisation d'un AGL).

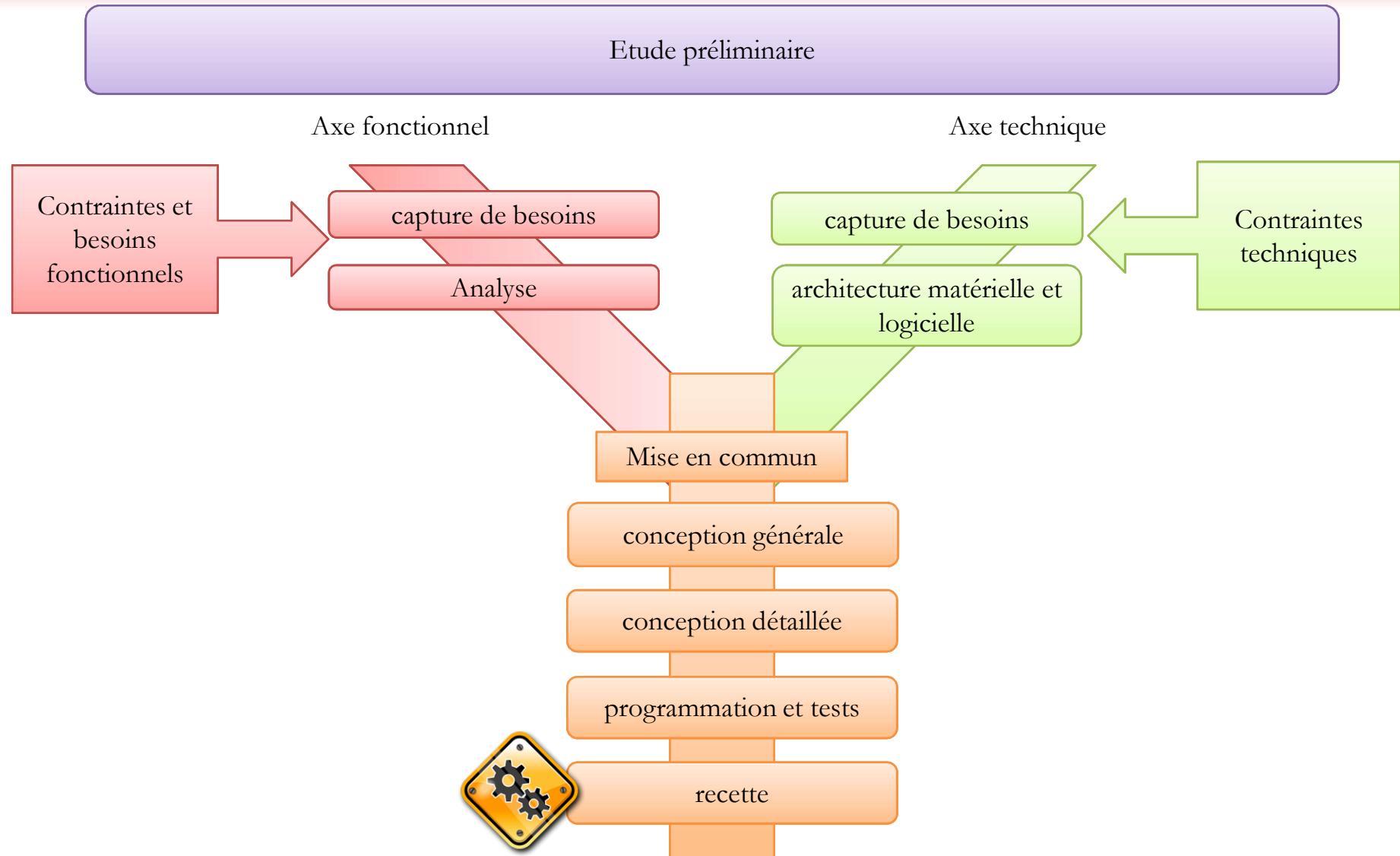
Processus unifié 2TUP (2 Tracks Unified Process)



Programmation

- Eventuellement parties de code générées par AGL
- Une grande partie des tests est menée à partir des spécifications

Processus unifié 2TUP (2 Tracks Unified Process)



Recettes

- Menée, entre autres, à partir des spécifications.

Pourquoi UML ?

Pour définir les contraintes et besoins, tomber d'accord sur la conception et définir les tests à effectuer, doivent pouvoir communiquer :

- les utilisateurs métier (commerciaux, secrétaires, caissières, clients de la banques, etc),
- les techniciens : spécialiste conception, développeurs (« codeurs »), les urbanistes c.-à-d. SSII, etc.
- les auditeurs,
- les décideurs,
- ...

UML = langage standard de communication

Pourquoi UML ?

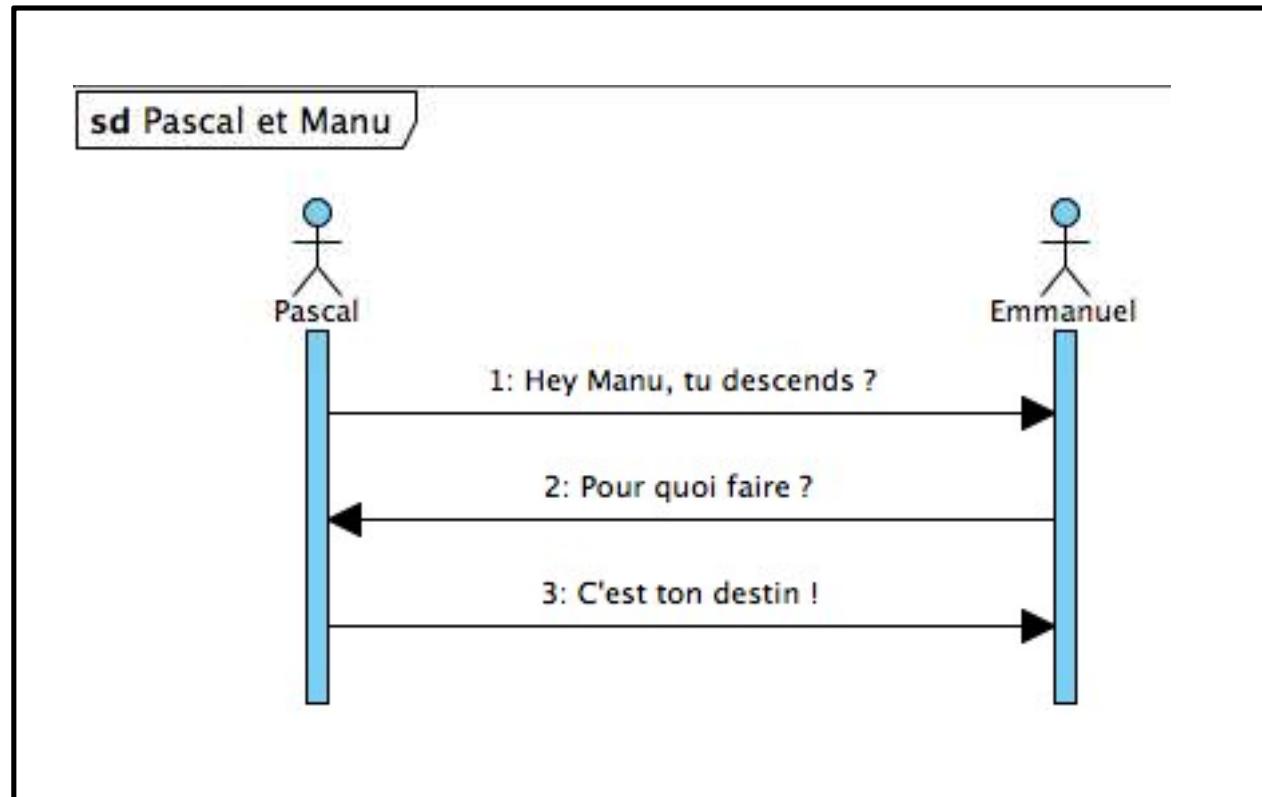
UML pour échanger

UML pour spécifier (représenter le système)

UML pour documenter

- Langage à base de représentations graphiques, les *diagrammes*, commentés et éventuellement enrichis de contraintes
- Permet des descriptions fonctionnelles, statiques et dynamiques
 - du (futur) système,
 - des utilisateurs face au système,
 - des processus métier des utilisateurs,
 - d'objets manipulés par le système,
 -

Langage à base de diagrammes



Mais attention (!)



- UML est un **langage** → pas une méthode !
savoir parler ne signifie pas qu'on sait à qui parler, à quel moment engager la conversation, et quoi dire
→ voir les processus de développement pour l'aspect méthodologique associé
- UML ne résout pas tous les problèmes de communication :
 - comme avec tout langage, on peut « mal s'exprimer » (erreur, imprécision, incohérence, ...)
 - précision → diagrammes complexes → difficiles à comprendre
 - compréhensibilité → diagrammes plus simples → sujets à interprétation (reste une part de subjectivité)

Donc attention (!)



- UML ne "guide" pas :
 - pas de conduite de projet,
 - pas de gestion de la qualité,
 - pas de gestion du changement,
 - pas de description des rôles,
 - etc

Les diagrammes : une véritable panoplie

- **Statiques**
 - diagramme de cas d'utilisation (*use case diagram*)
 - diagramme de classes (*class diagram*)
 - diagramme d'objets (*object diagram*)
 - diagramme de composants (*component diagram*)
 - diagramme de déploiement (*deployment diagram*)
 - (UML2) diagramme de packages (*package diagram*)
 - (UML2) diagramme de structure composite (*composite structure diagram*)
- **Dynamiques**
 - diagramme d'états-transition (*state machine diagram*)
 - diagramme d'activité (*activity diagram*)
 - diagramme de séquence (*sequence diagram*)
 - diagramme de communication (*communication diagram*) anciennement « de collaboration »
 - (UML2) diagramme global d'interaction (*interaction overview diagram*)
 - (UML2) diagramme de temps (*timing diagram*)
- **Fonctionnels**
 - diagramme de cas d'utilisation
 - diagramme d'activité
 - diagramme de séquence

Les diagrammes : une véritable panoplie

- Un même type de diagramme (p.e. diagramme d'activité)
 - peut être utilisé :
 - pour modéliser des concepts différents
 - à des moments différents du processus de développement (donc pour des objectifs différents)
 - à différents niveaux d'abstraction
 - peut ne pas être utilisé du tout dans le processus de développement

Les plus utilisés

- ◆ diagramme de cas d'utilisation (*use case diagram*)
- ◆ diagramme d'activité (*activity diagram*)
- ◆ diagramme de séquence (*sequence diagram*)
- ◆ diagramme de classes (*class diagram*) et packages (*package diagram*)
- ◆ diagramme de communication (*communication diagram*)

- ❖ diagramme de composants (*component diagram*)
- ❖ diagramme de déploiement (*deployment diagram*)
- ❖ diagramme d'états-transitions (*state machine diagram*)

Exemple d'utilisation :

Le diagramme de communication (simplifié) pour représenter un contexte statique

COMMUNICATION DIAGRAM (1/2)

Diagramme de communication

En analyse, permet de :

- décrire un contexte statique, le système étant représenté comme un objet au centre du diagramme, sans numérotation des messages,
- décrire une organisation (dynamique),
- formaliser un scenario (dynamique).

En conception, permet de :

- formaliser les enchaînements d'écrans d'une IHM (dynamique),
- décrire des mécanismes de synchronisation entre objets, l'enchaînement des appels entre les méthodes de classes (dynamique),
- ...

Diagramme de communication

Description d'un contexte (statique)

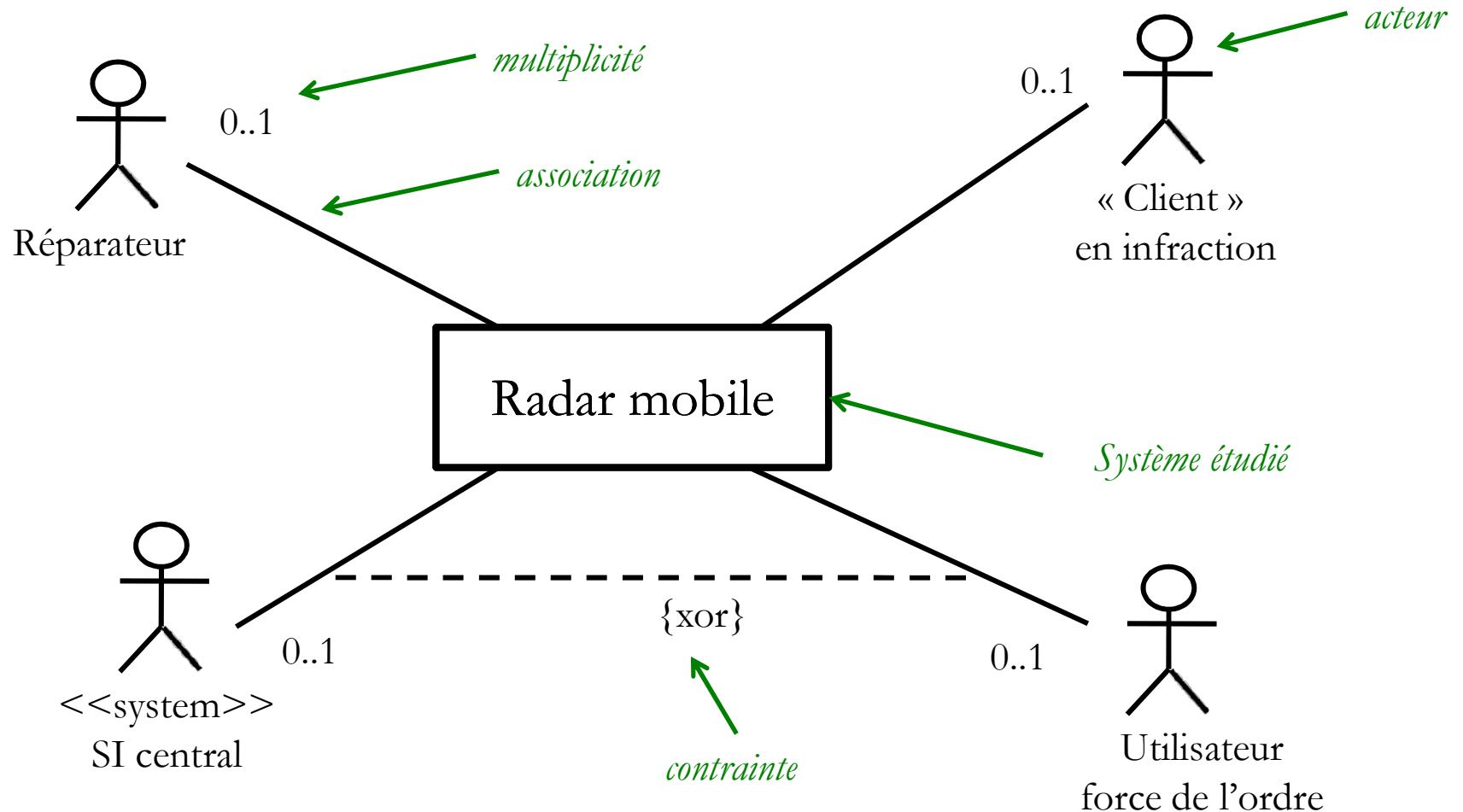
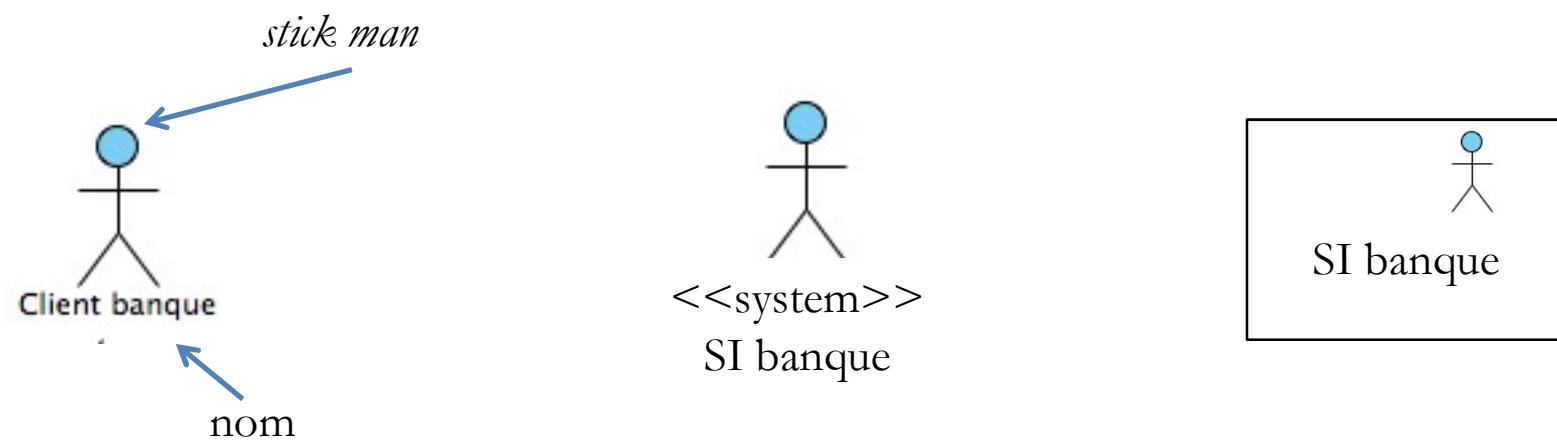


diagramme de cas d'utilisation

USE CASE DIAGRAM

- **Rôle** joué par une entité externe au SI (pas forcément humain) interagissant avec le système étudié
- Représentation :



Cas d'utilisation

Cas d'utilisation (use case) :

- Ensemble d'actions réalisées par le système
 - souvent en réponse à l'action d'un acteur,
 - produisant un résultat observable intéressant pour un acteur.
- Un CU modélise un service rendu par le système à un acteur.
Il spécifie ce que le système fait (ou devra faire) mais sans expliquer comment.

Exemple pour un distributeur de monnaie :

un « client » (acteur) peut « Retirer de l'argent » (cas d'utilisation)

Diagramme de cas d'utilisation

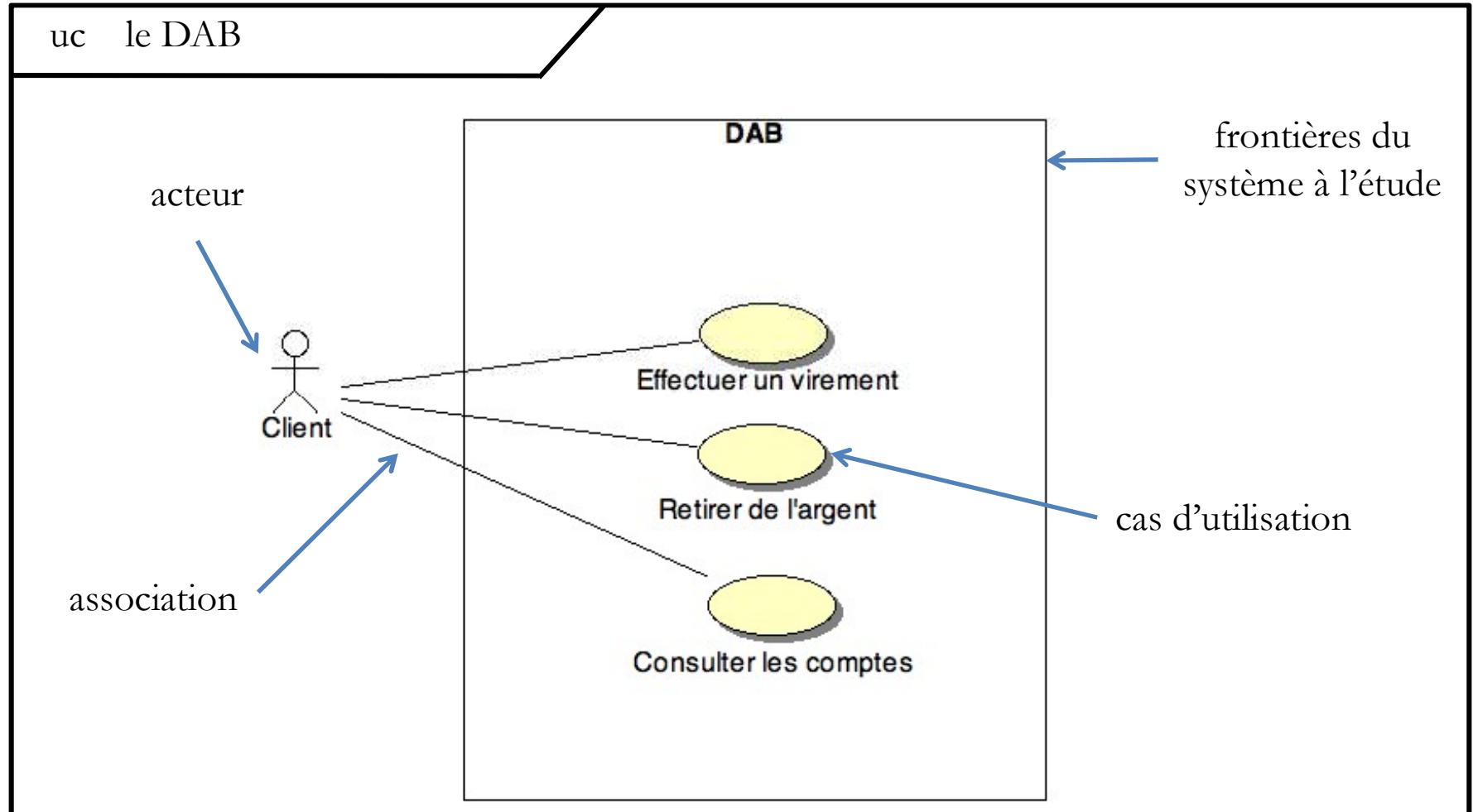
Caractéristiques :

- issu d'entretien avec les utilisateurs
- centré utilisateur
- haut niveau

Intérêts :

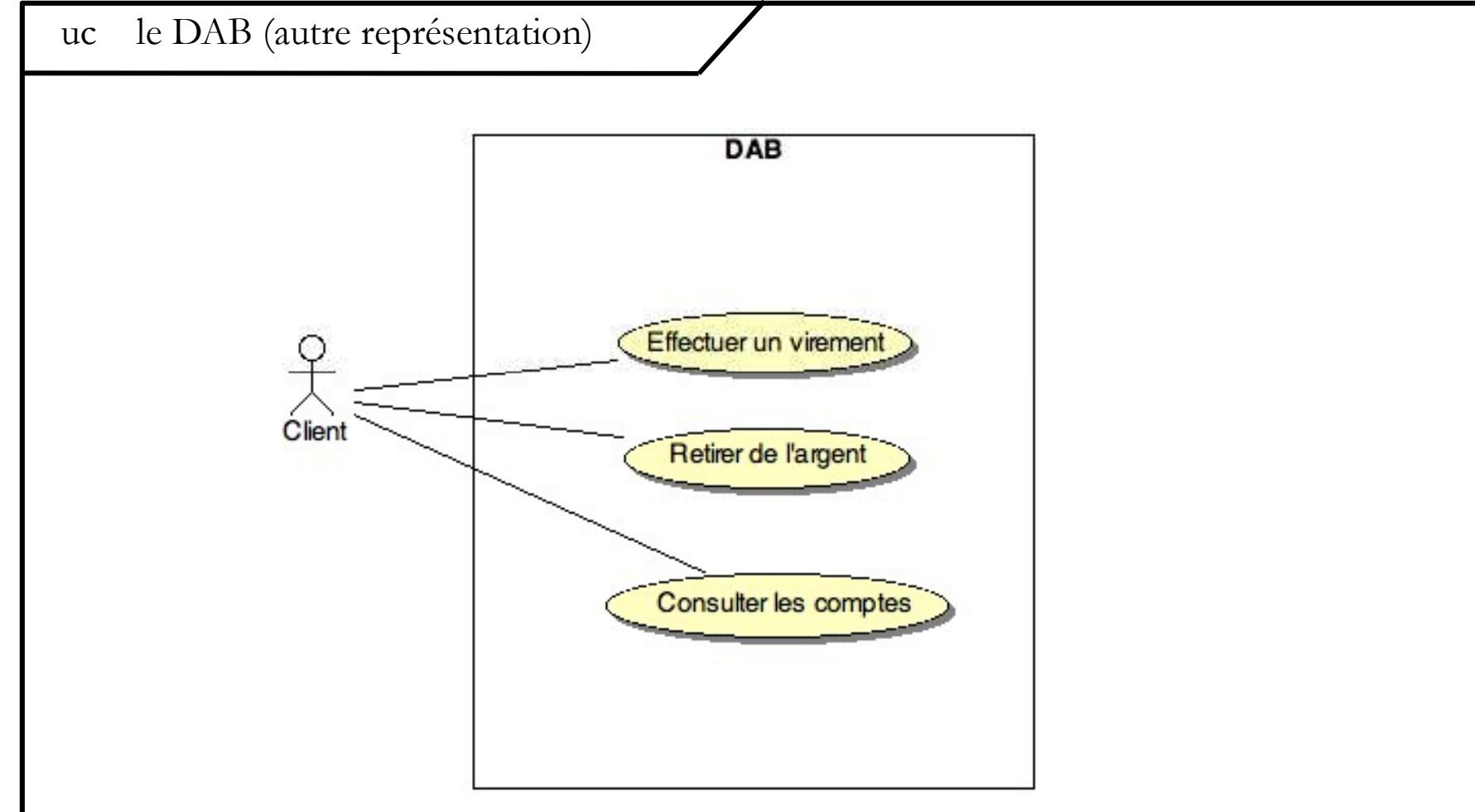
- exprime les besoins des utilisateurs
- délimite les frontières du système
- implique les utilisateurs dans la conception
- première base pour les tests fonctionnels

Diagramme de cas d'utilisation



BOUML

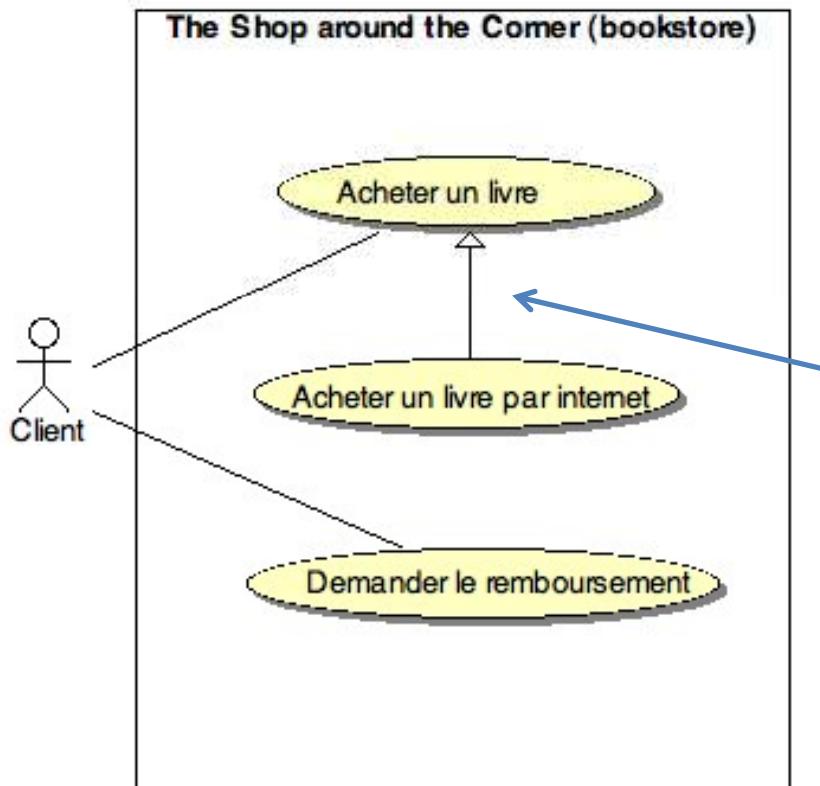
Diagramme de cas d'utilisation



BOUML

Diagramme de cas d'utilisation

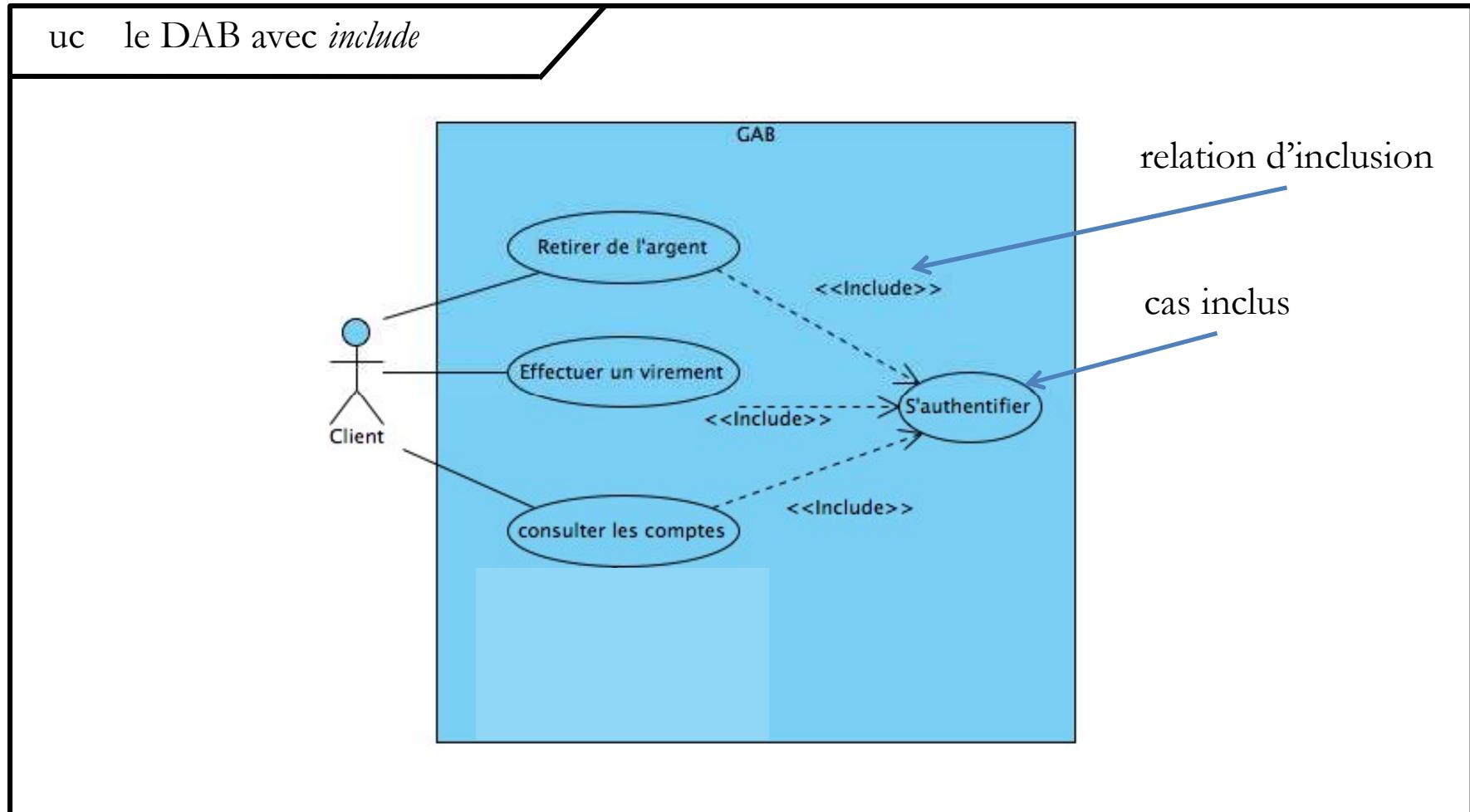
uc la librairie avec généralisation de cas



Généralisation (de bas en haut)
ou
Spécialisation (de haut en bas)

BOUML

Diagramme de cas d'utilisation



<< ... >> = stréotype

Diagramme de cas d'utilisation

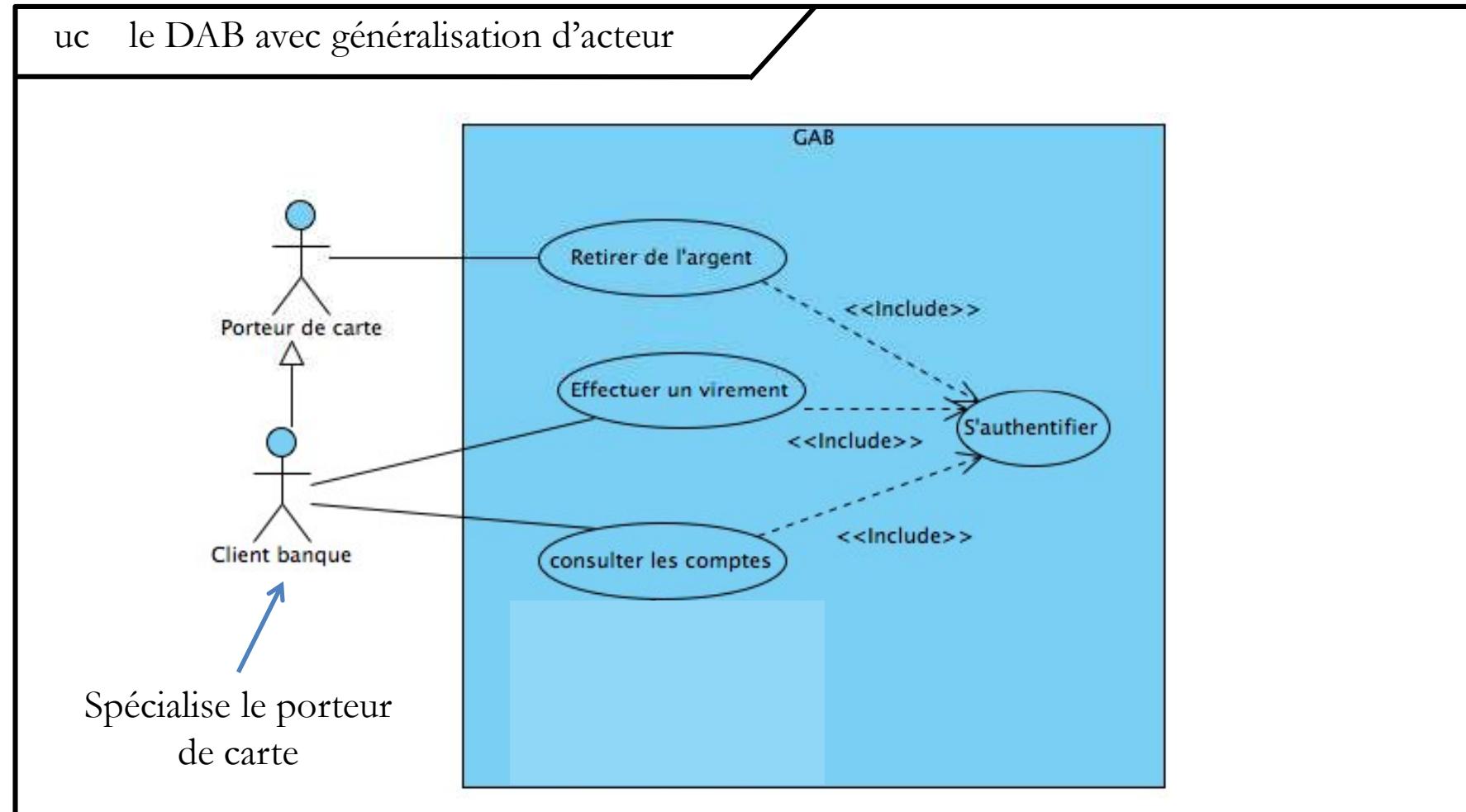


Diagramme de cas d'utilisation

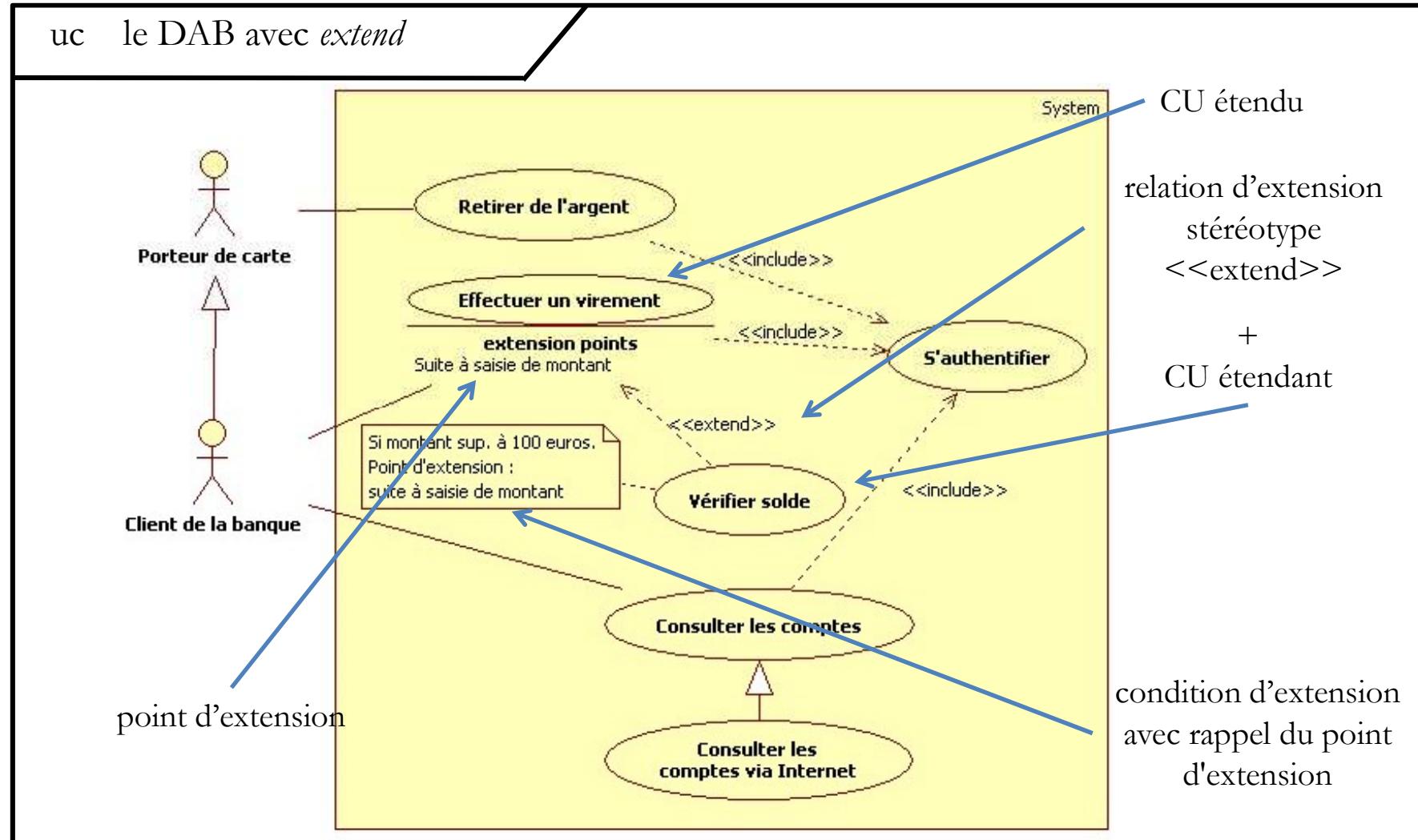


Diagramme de cas d'utilisation

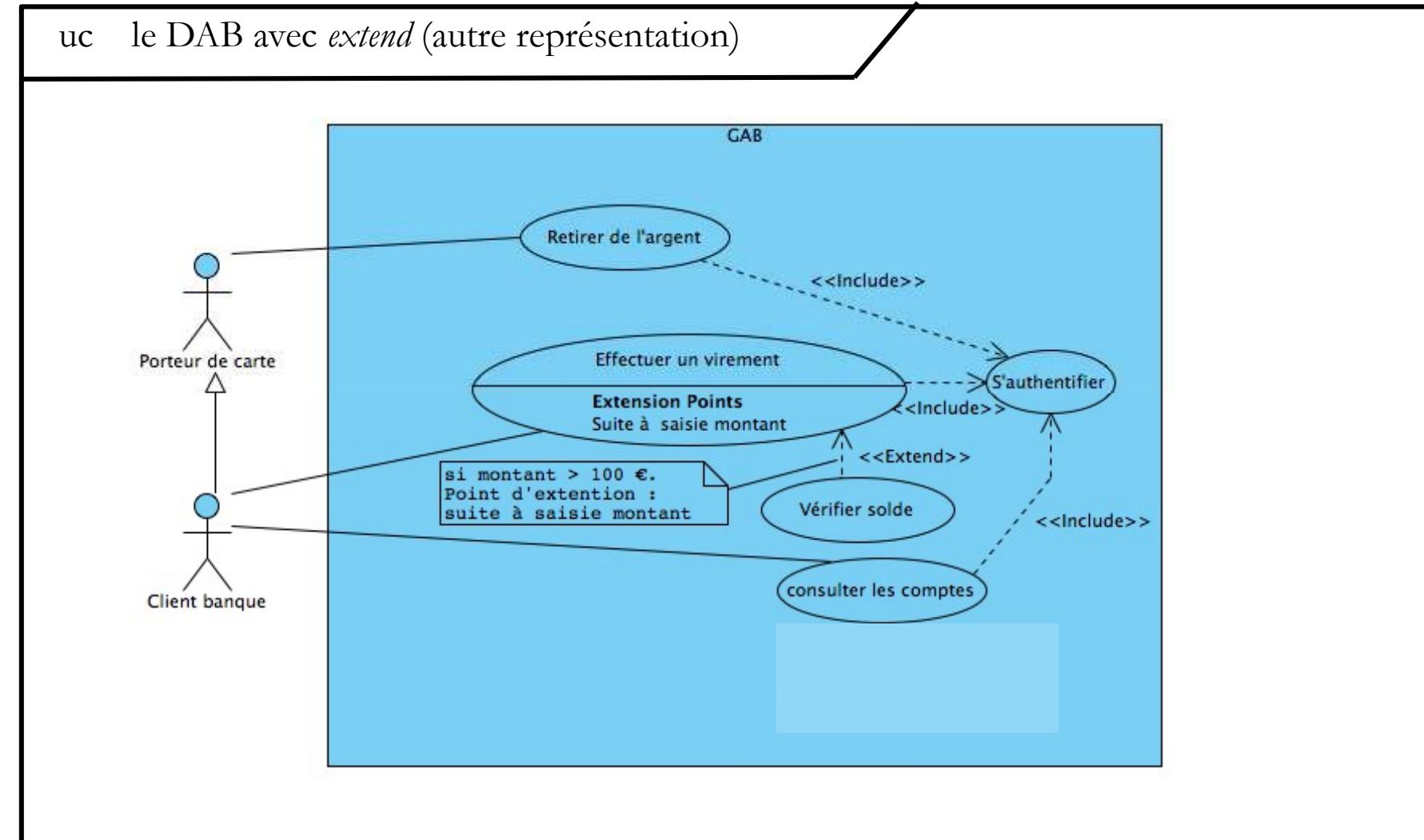


Diagramme de cas d'utilisation

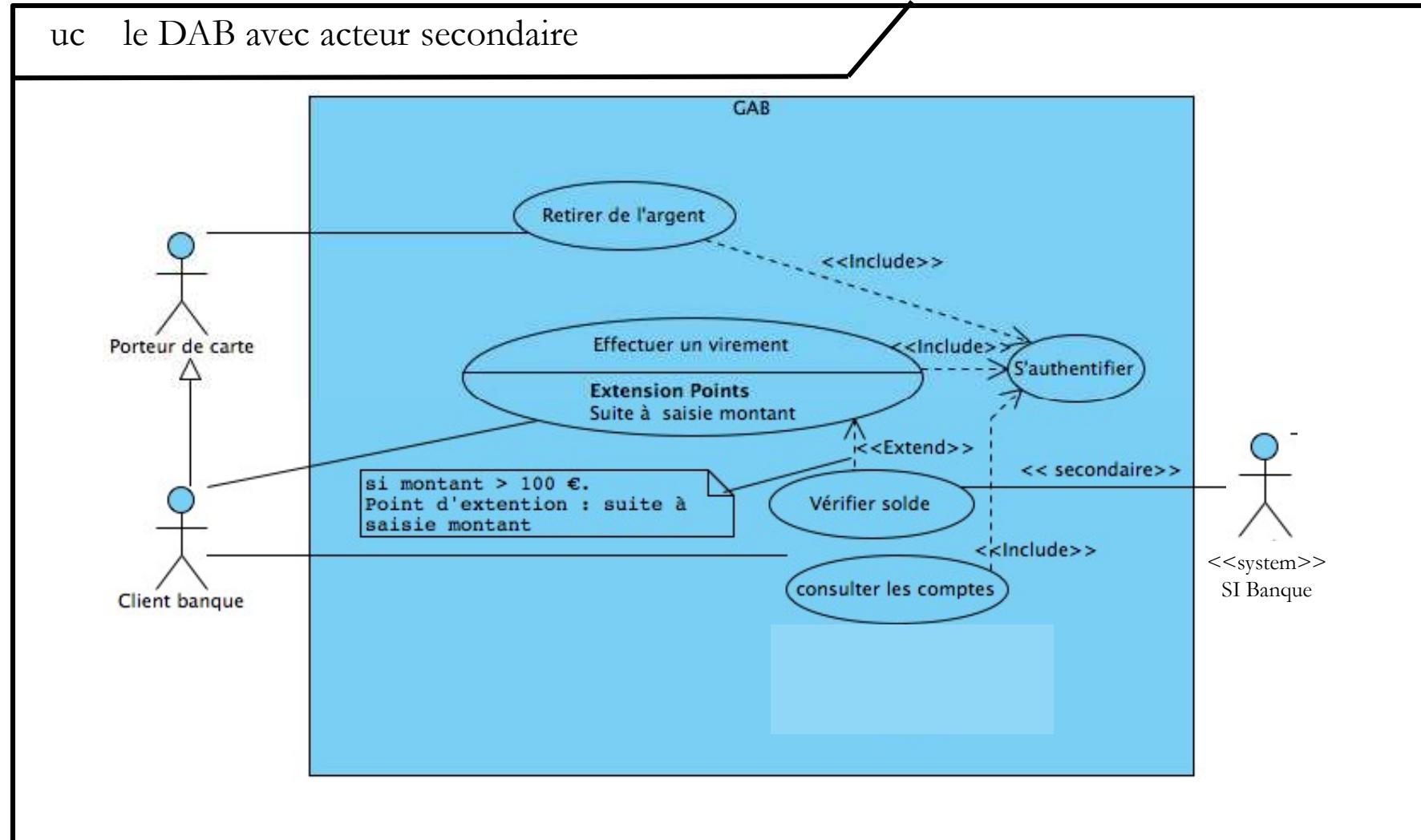


Diagramme de cas d'utilisation

- Attention au niveau de détail : un CU n'est pas une fonction
- Remarques importantes (pour le DCU mais valable pour tous les autres types diagrammes)
 - il existe des variantes (p.e. notation des acteurs), des compléments (p.e. multiplicité ici), d'autres composants, etc
 - on ne définit pas « le » diagramme mais UN diagramme. Il tient compte de votre vision, de vos objectifs et de ceux à qui (et ce à quoi) est destiné le diagramme.

Diagramme de cas d'utilisation

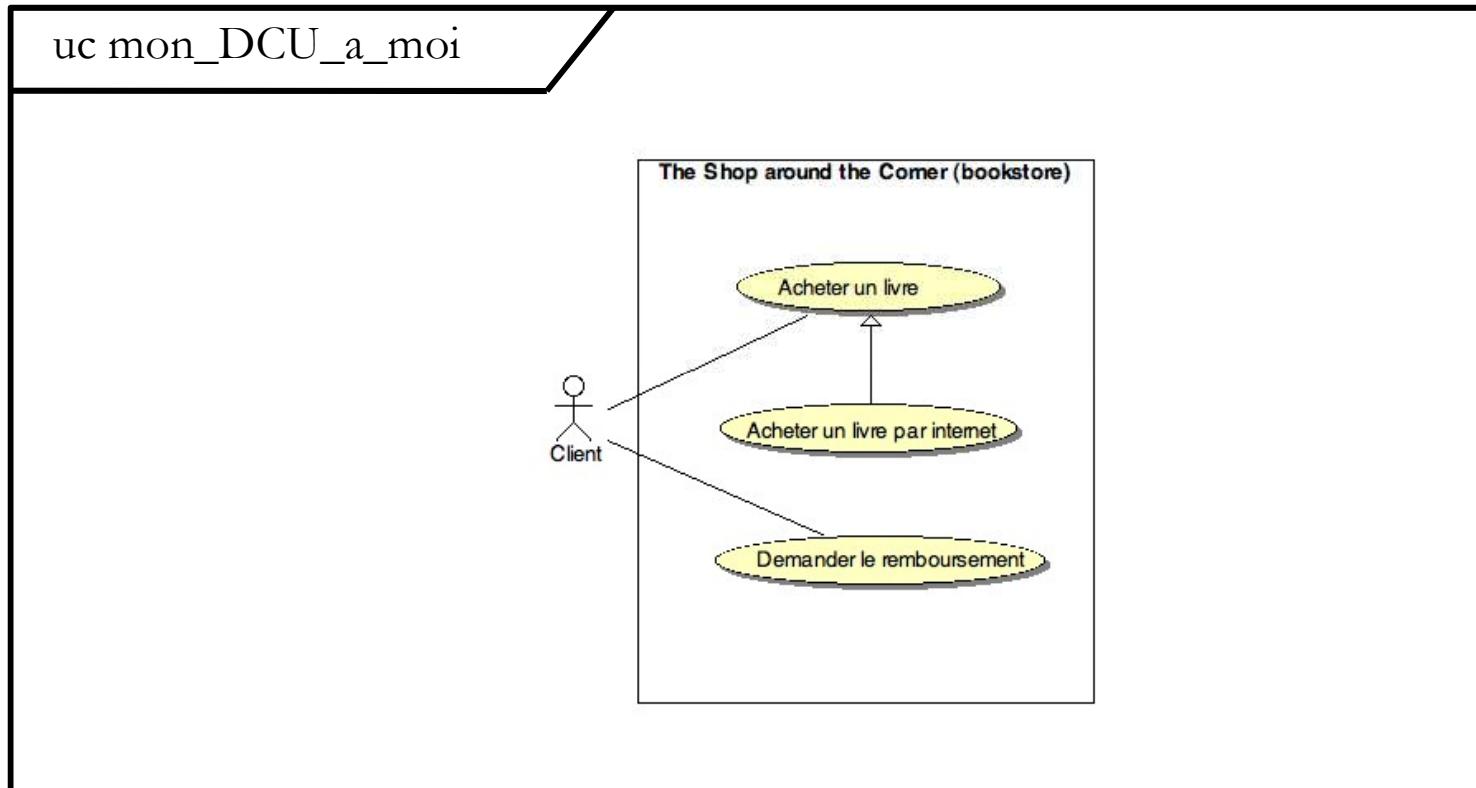


diagramme de classes

CLASS DIAGRAM

*partie de ce cours en partie très fortement
inspirée du cours UML 2 de Jacques Callot*

class mon_DC_a_moi

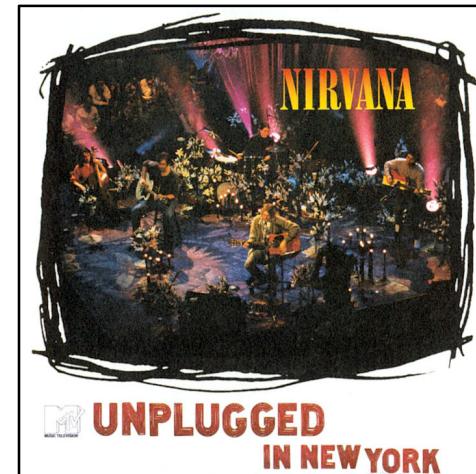
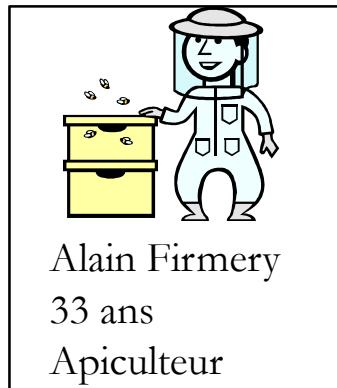
Diagramme de classe

- Modélisation des objets du systèmes et leurs relations,
 - Montre la structure interne du SI,
 - Vue statique.
-
- Si utilisé pour analyse : indépendant du langage de programmation utilisé (comme tous les autres diagrammes d'ailleurs),
 - Si utilisé pour conception, la présentation et le formalisme peuvent dépendre du langage de programmation.
-
- Peut également permettre de représenter le modèle d'une base de données.

Objet

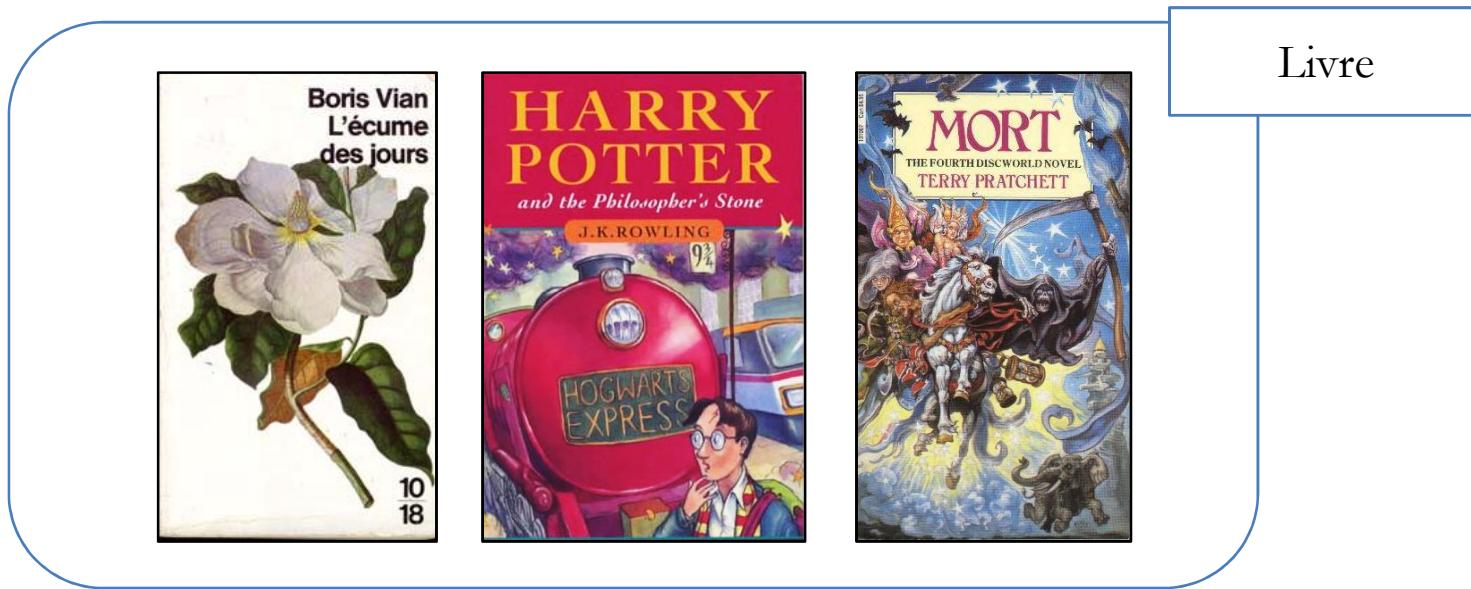
Avant la classe... les objets

- Un objet est la représentation d'une abstraction d'une "chose" concrète ayant des limites très claires et un sens précis dans le contexte du problème étudié.
- Chaque objet a une identité et peut être distingué des autres par son existence inhérente et non grâce à la valeur de certaines de ses propriétés.



Classe

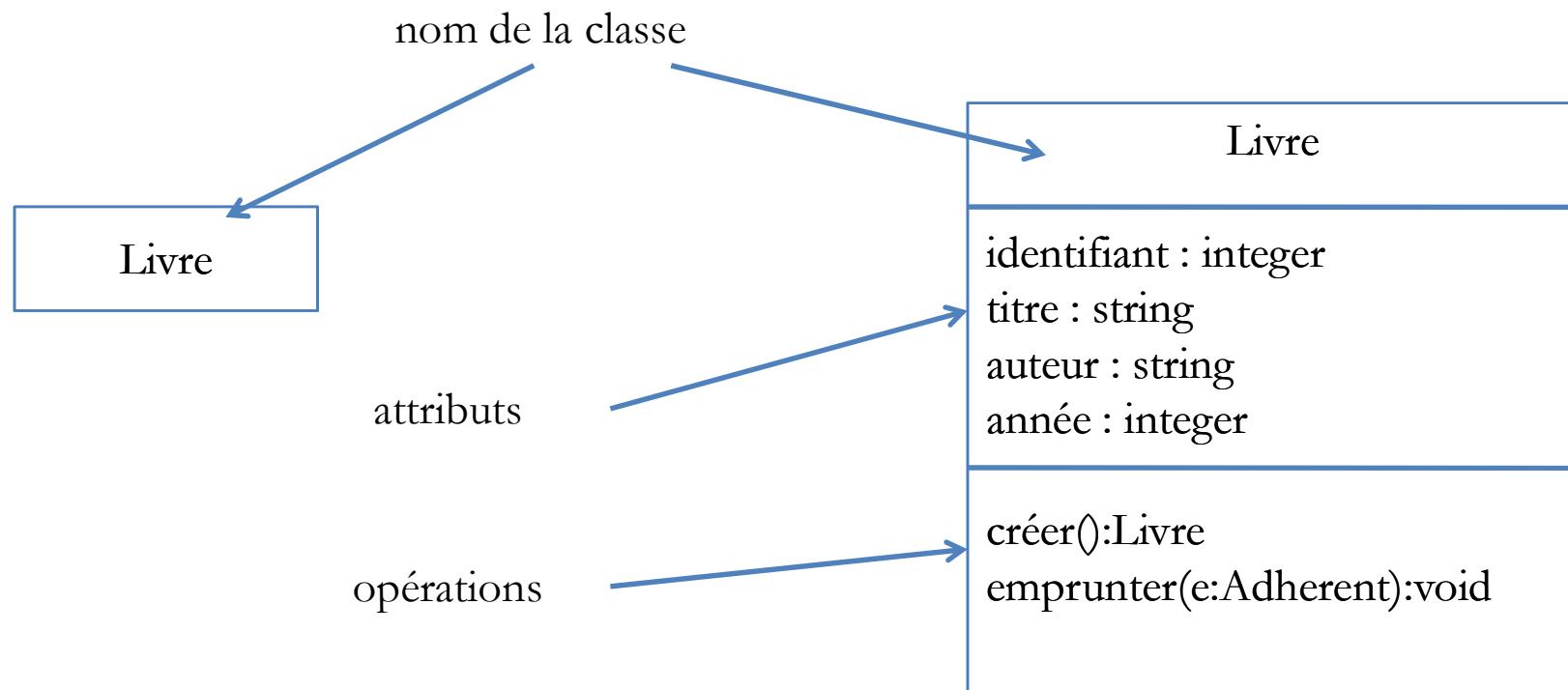
Une **classe** est l'abstraction d'un ensemble d'objets de même nature (\Leftrightarrow un objet est une instance de classe)



L'écume des jours est un objet.

L'écume des jours est une **instance** de la classe *Livre*.

Classe : représentation



ici, deux niveaux d'abstraction différents

Classe : attribut

Un **attribut** est une donnée qui est gérée pour une classe donc pour tous les objets (instances) de cette classe.

Classe : opération

Une **opération** est un élément de comportement contenu dans une classe (ou de conception des objets tels que création, destruction, etc.). C'est une action effectuée sur ou par les objets d'une classe.

Une opération peut :

- Accéder (consultation - mise à jour) aux attributs de son propre objet,
- Invoquer une autre opération de son propre objet,
- Invoquer une opération d'un autre objet (de la même classe ou d'une autre classe).

Une opération ne peut pas (ou plutôt ne doit pas) :

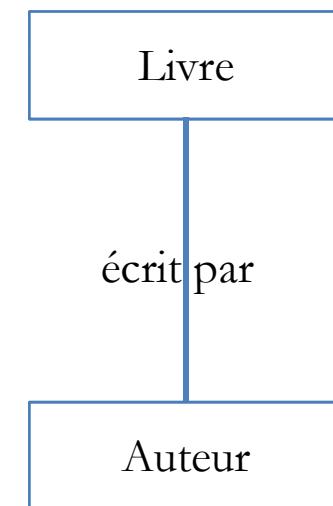
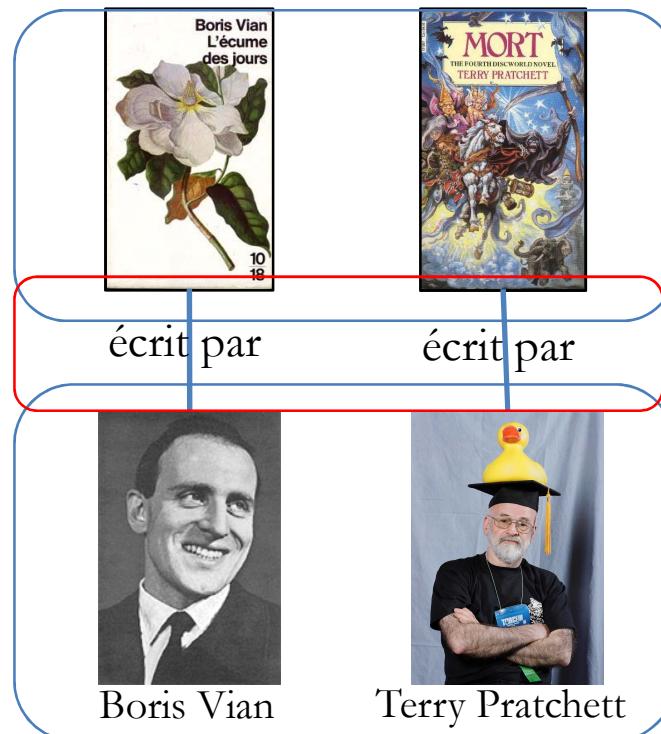
- Accéder directement (consultation - mise à jour) aux attributs d'un autre objet.

Si une opération d'un objet A doit accéder aux attributs d'un objet B, elle doit invoquer une opération de l'objet B qui fera cet accès et restituera les informations à l'objet A.

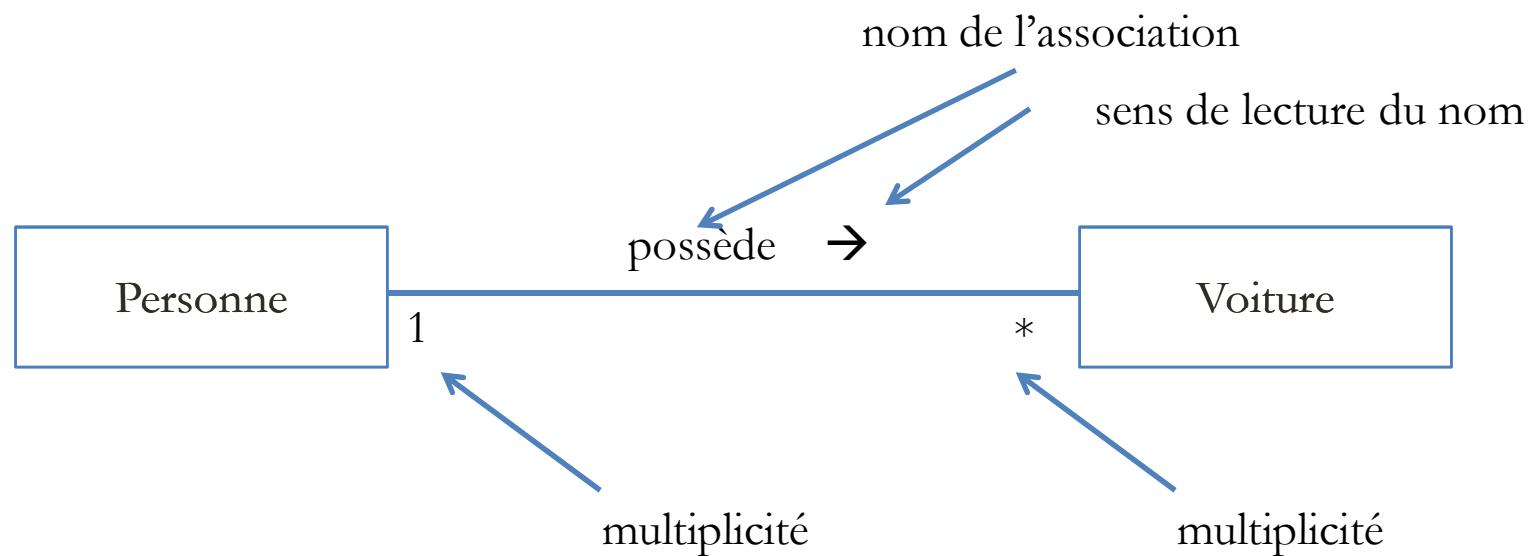
Association

Avant l'association... le lien.

- Un lien est une connexion physique ou conceptuelle entre des instances de classes. Une association est l'abstraction d'un ensemble de liens ayant une structure et une sémantique commune.
- Un lien est une instance d'association.



Association : représentation



Association : multiplicité

n..m : entre n et m (min-max)

0..1 : zéro ou un

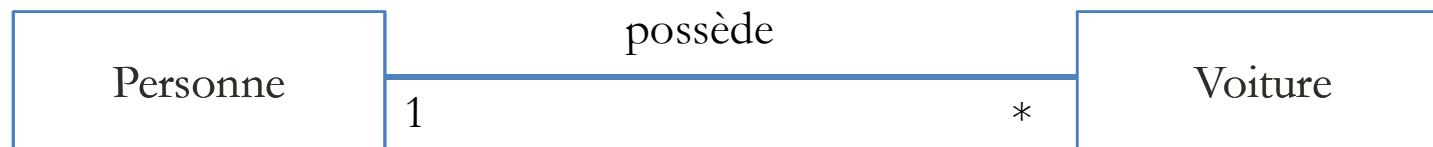
1..* : au moins un

* : équivalent à 0..*

1 : exactement un

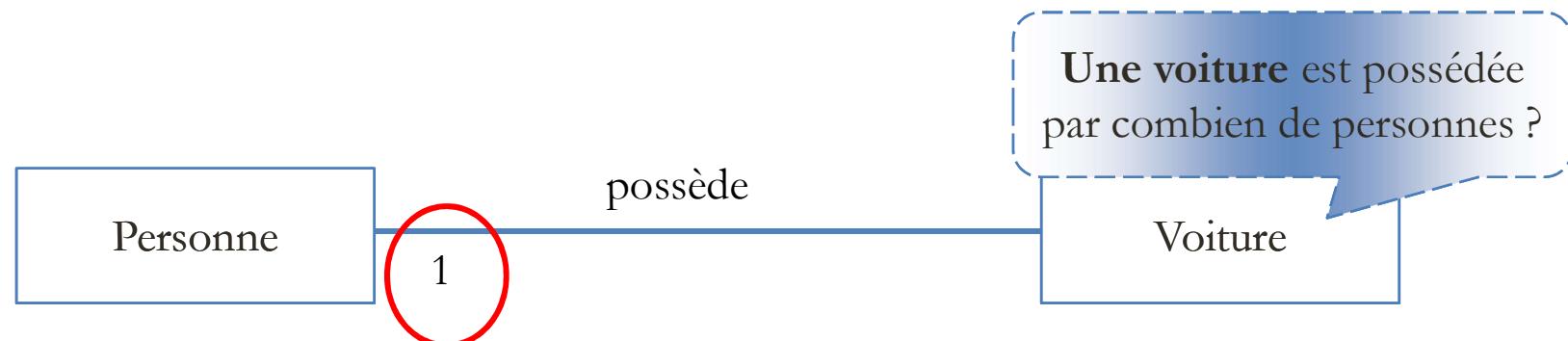
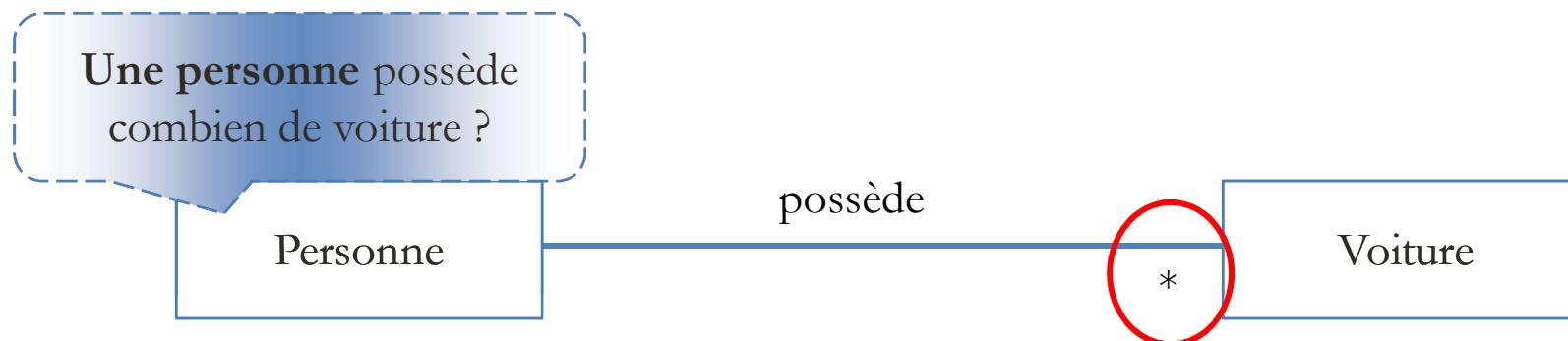
2 : exactement deux

... etc.



Association : multiplicité

Les deux multiplicités se lisent indépendamment l'une de l'autre.

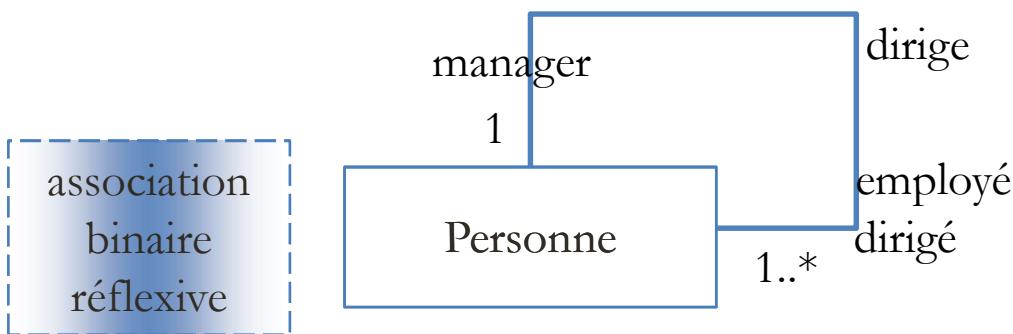
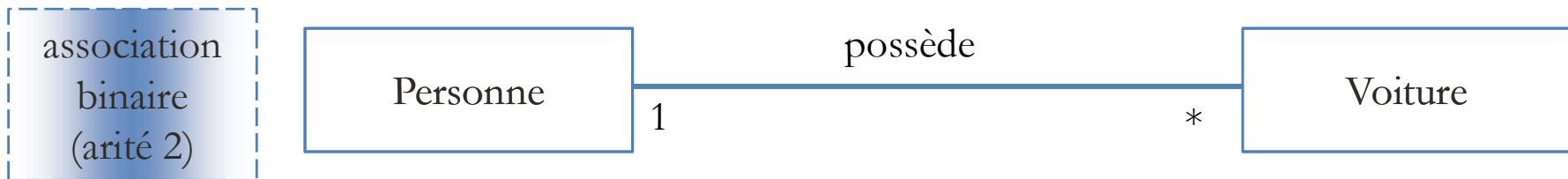


Association binaire : elle associe deux classes.

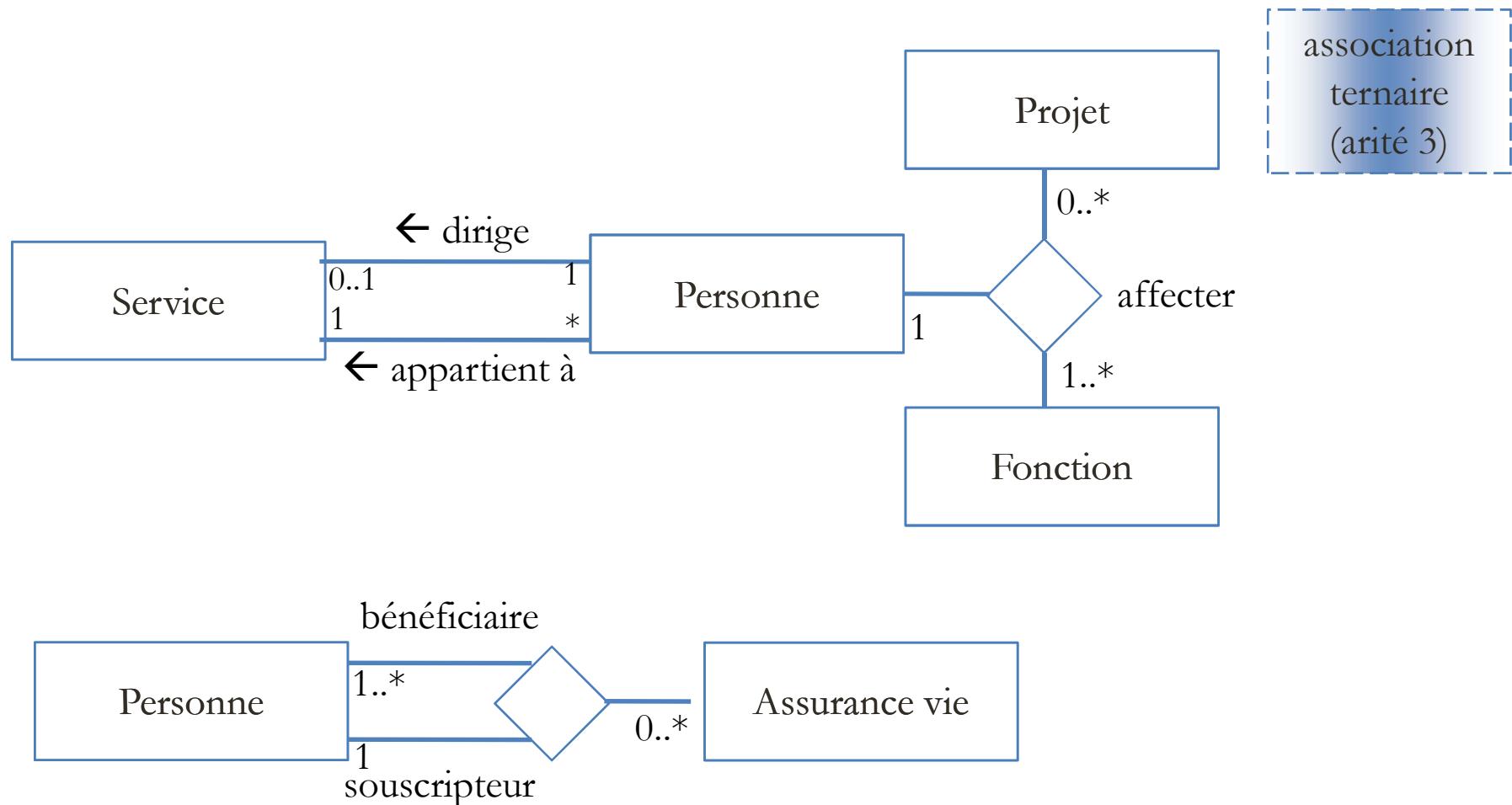
Si elle associe une classe à elle même (chaque lien lie un objet de la classe à (généralement) un autre objet de la même classe), il est indispensable d'indiquer les rôles joués par la classe au travers de chaque branche de l'association réflexive

Association n -aire où n est l'arité de la relation : elle associe n classes

Arité d'une association



Arité d'une association



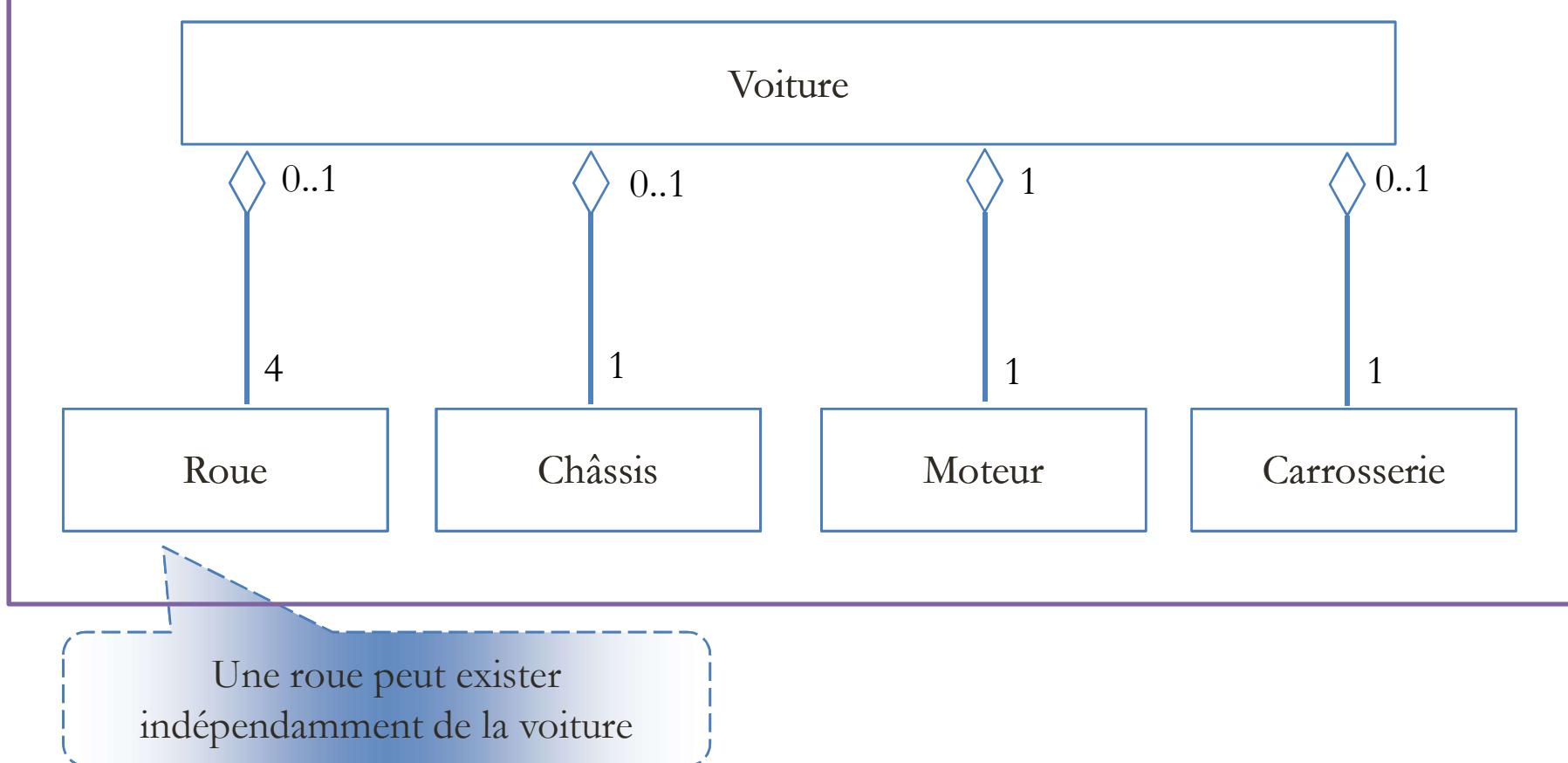
Agrégation et composition

Une **agrégation** est une association exprimant une relation de contenance.



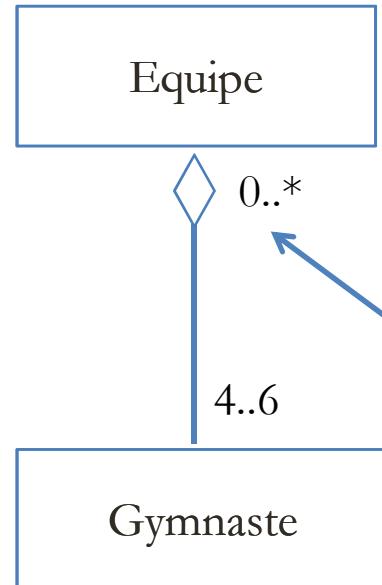
Agrégation et composition

DC : agrégation



Agrégation et composition

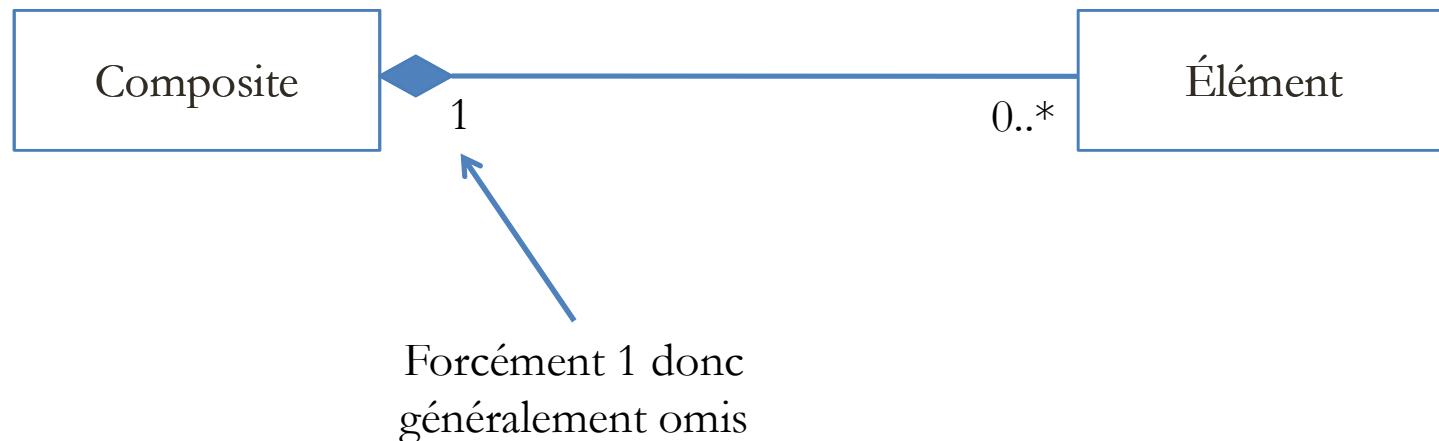
DC : agrégation



Notez la multiplicité : l'élément agrégant peut intervenir dans plusieurs agrégats

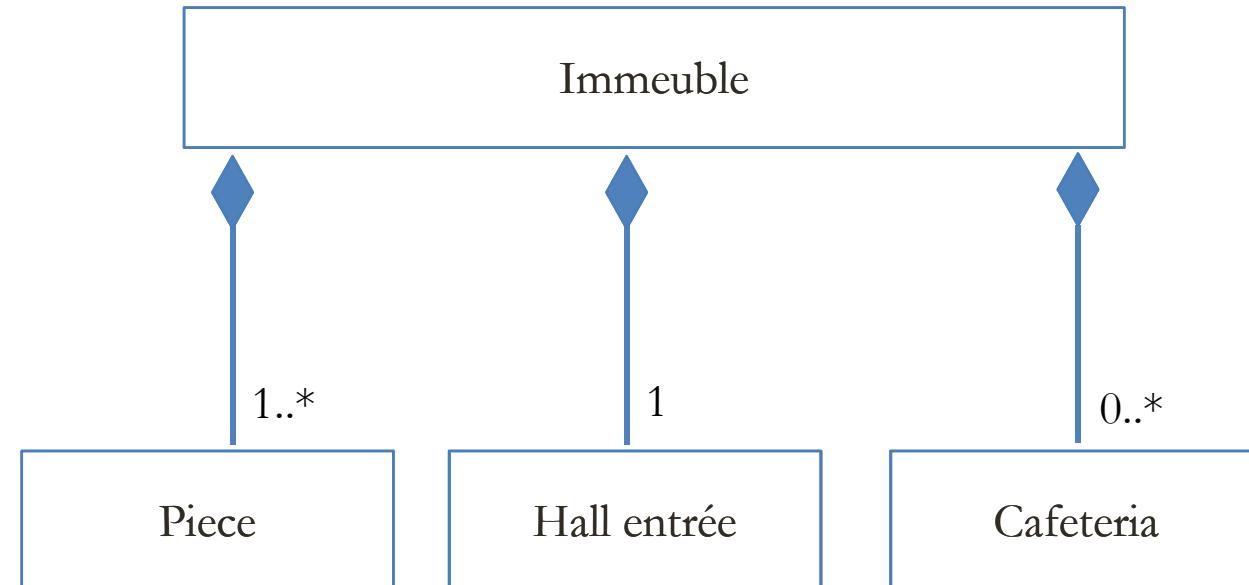
Agrégation et composition

Une composition est une agrégation plus forte impliquant que la destruction du composant agrégat entraîne la destruction de tous les éléments le composant (question de cycles de vie des objets).



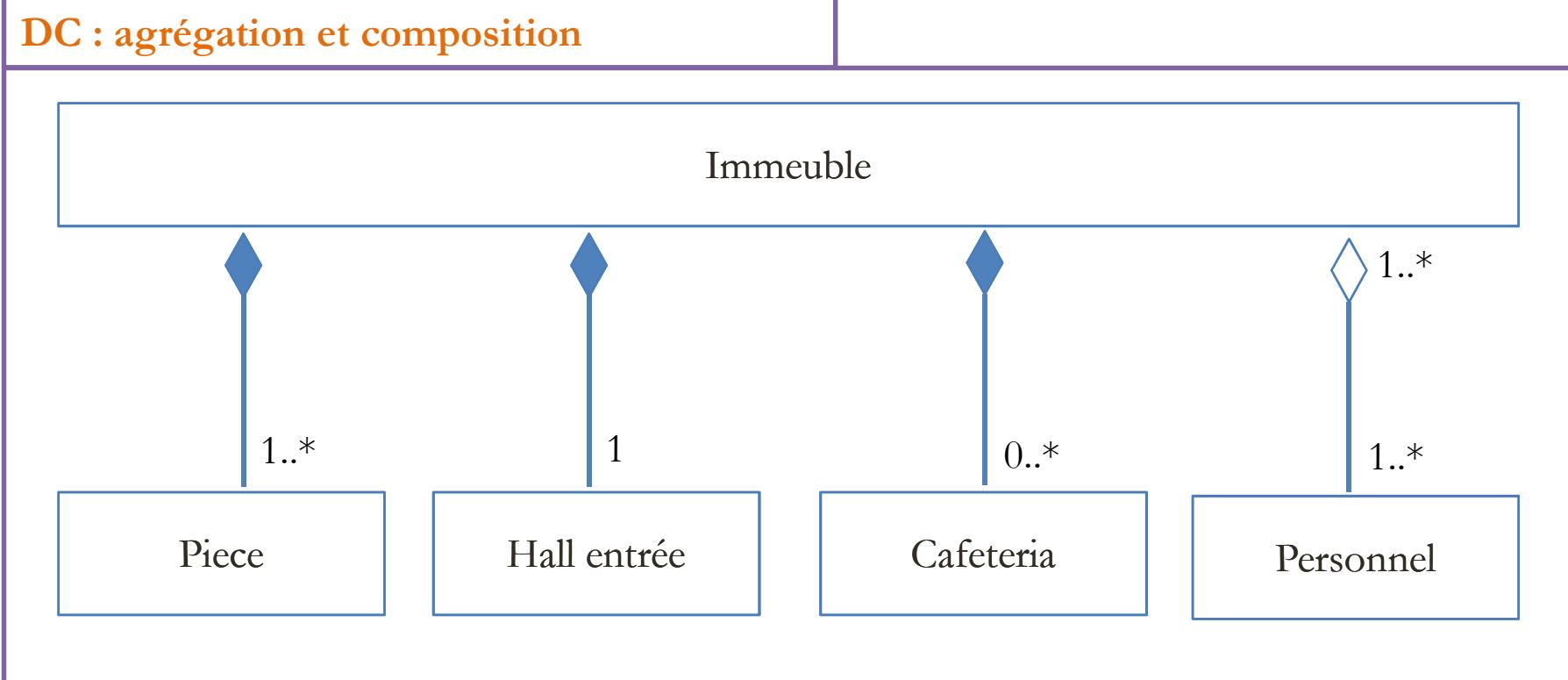
Agrégation et composition

DC : composition

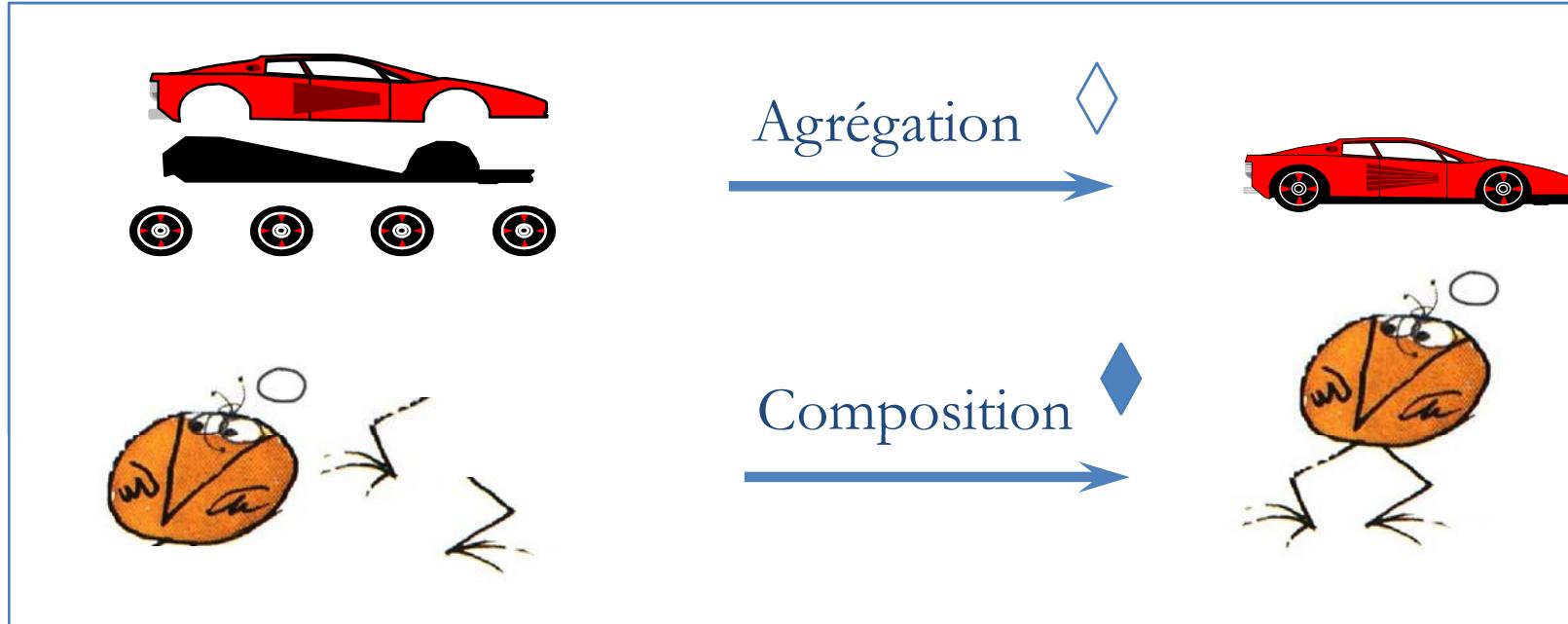


Les composants n'existaient pas avant l'immeuble.
Si l'immeuble disparaît, ses composants disparaissent aussi.

Agrégation et composition

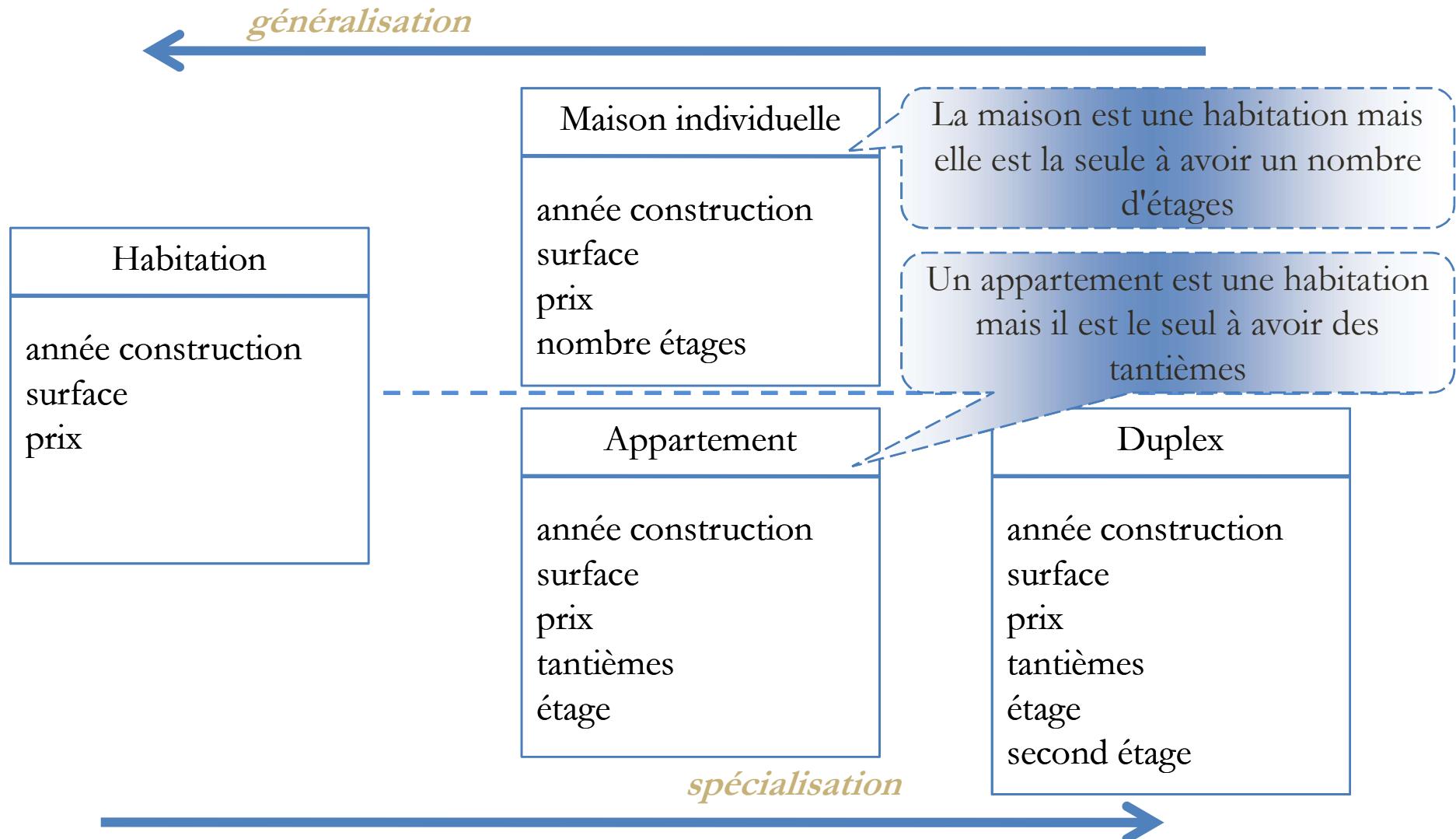


Agrégation vs composition

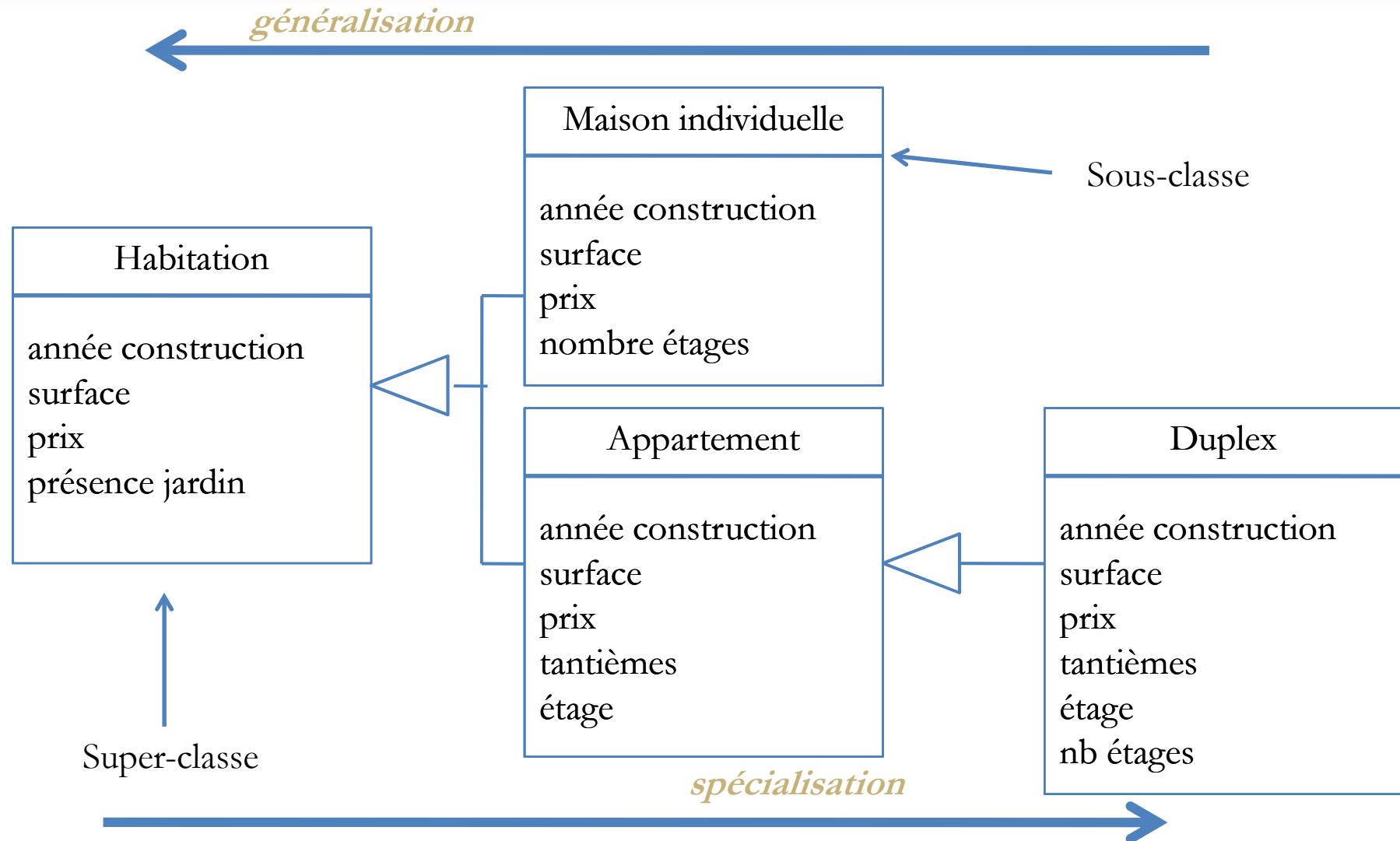


Choisir entre agrégation et composition → se poser la question du cycle de vie des éléments composants

Généralisation/specialisation



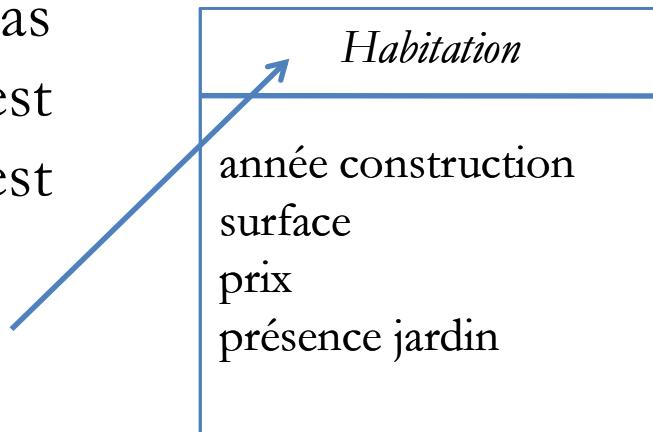
Généralisation/spécialisation



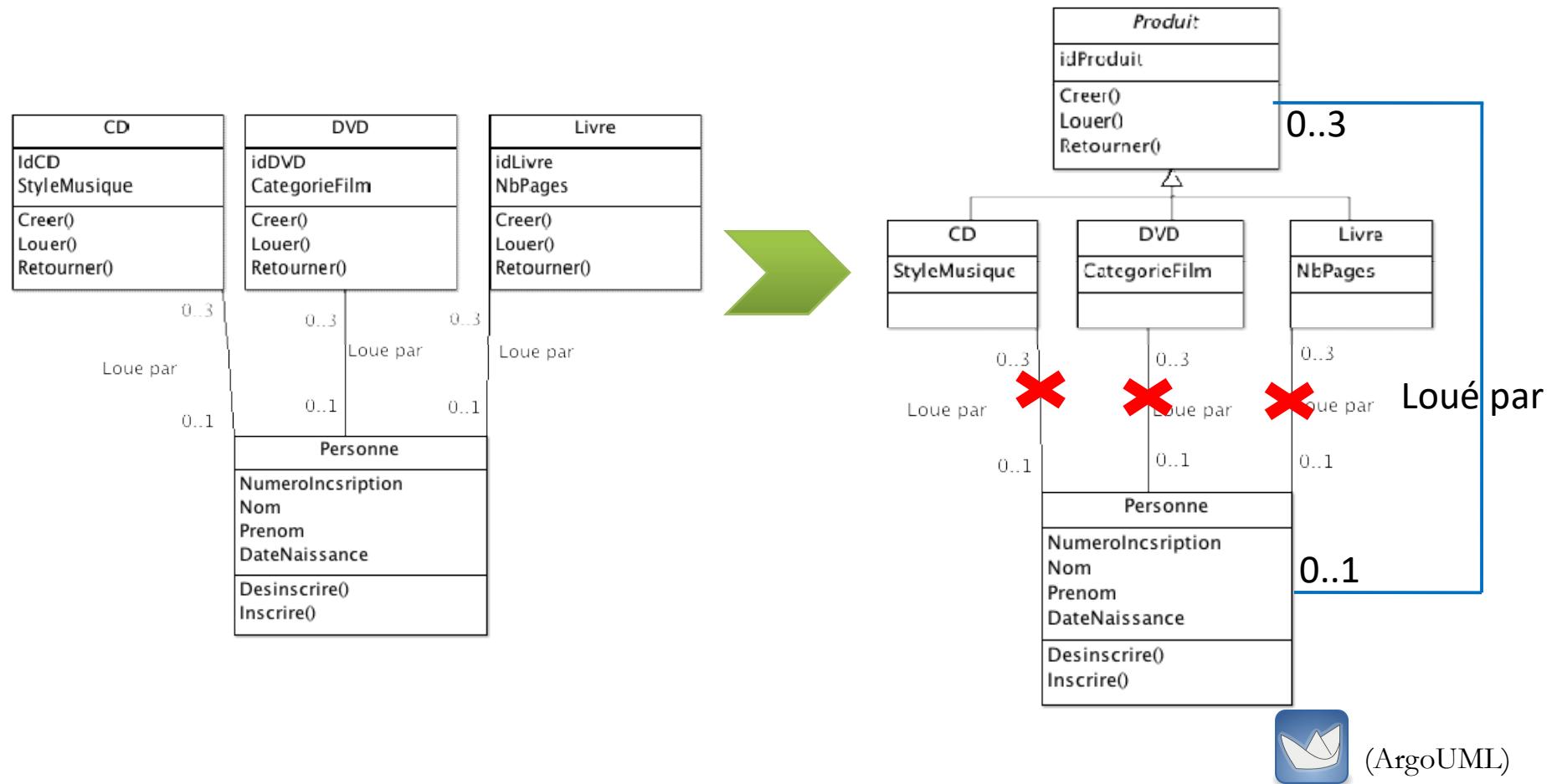
Généralisation/spécialisation

- La sous-classe hérite des attributs, opérations et associations avec multiplicités de la super-classe. Elle la spécialise.
- La généralisation peut n'être utilisée que pour factoriser des propriétés communes dans le diagramme.

Dans ce cas, la super-classe n'est pas instanciée dans le système. Elle est appelée **classe abstraite**. Son nom est simplement noté en italique.

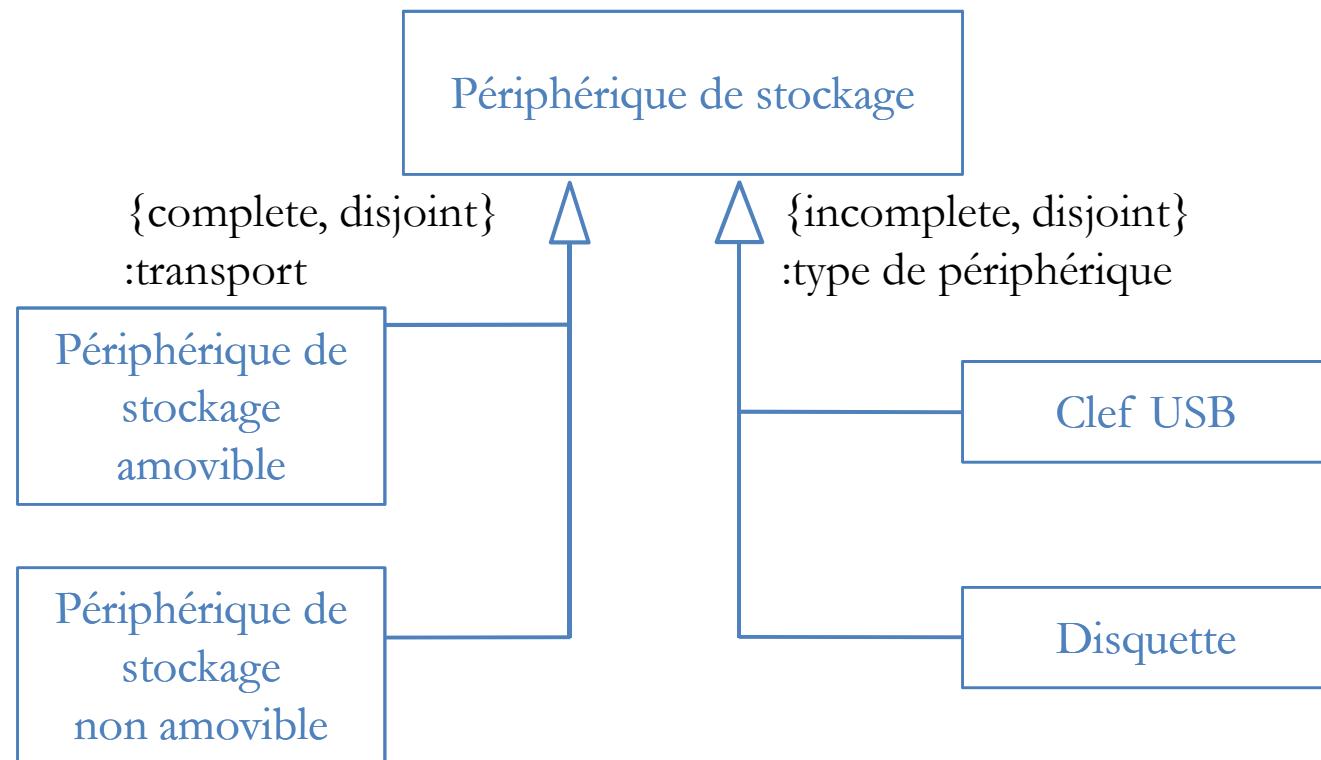


Généralisation/spécialisation



Généralisation/spécialisation

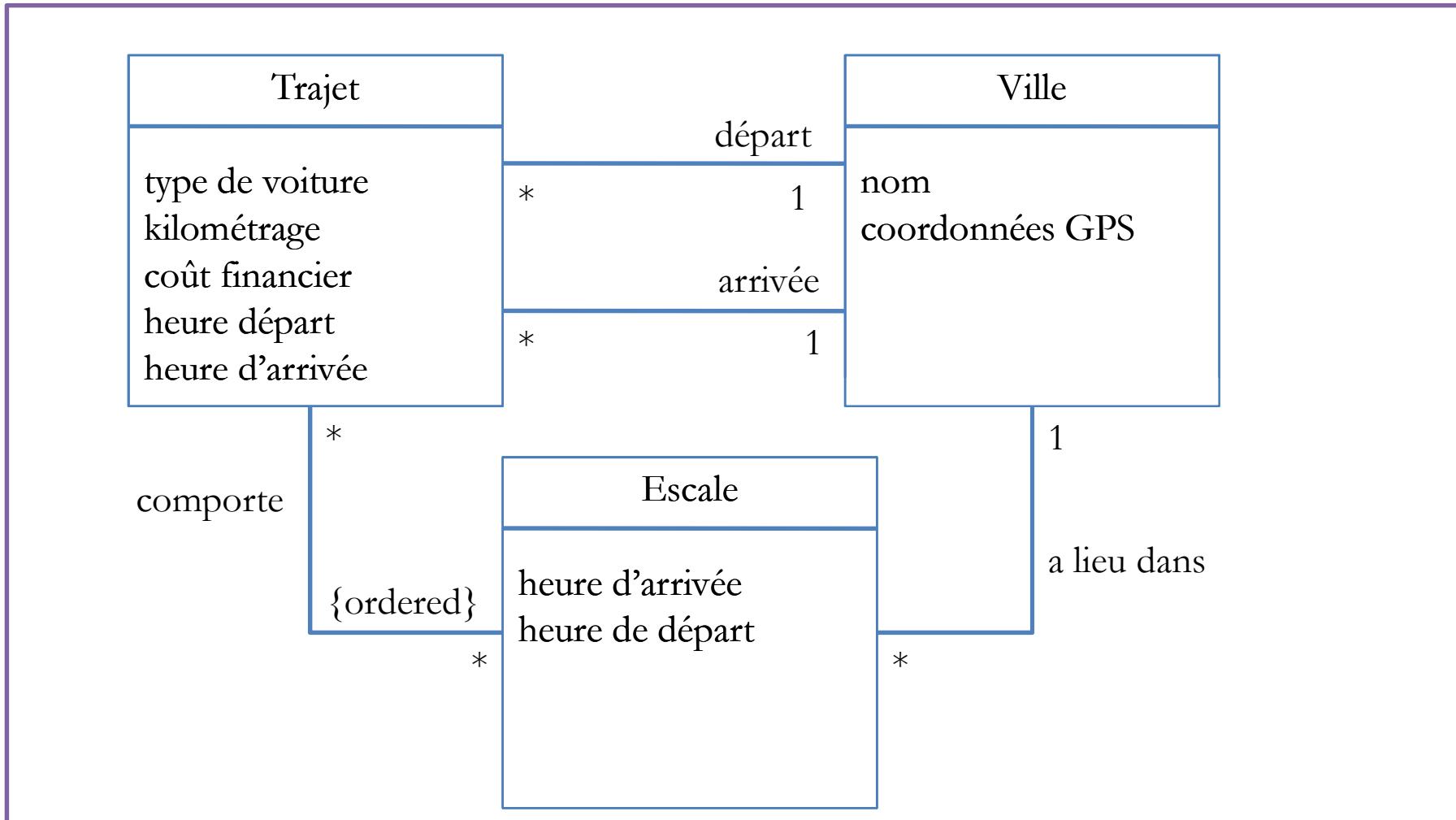
Vous pourrez aussi trouver (UML2)
Plusieurs relations de généralisation sur une même classe.



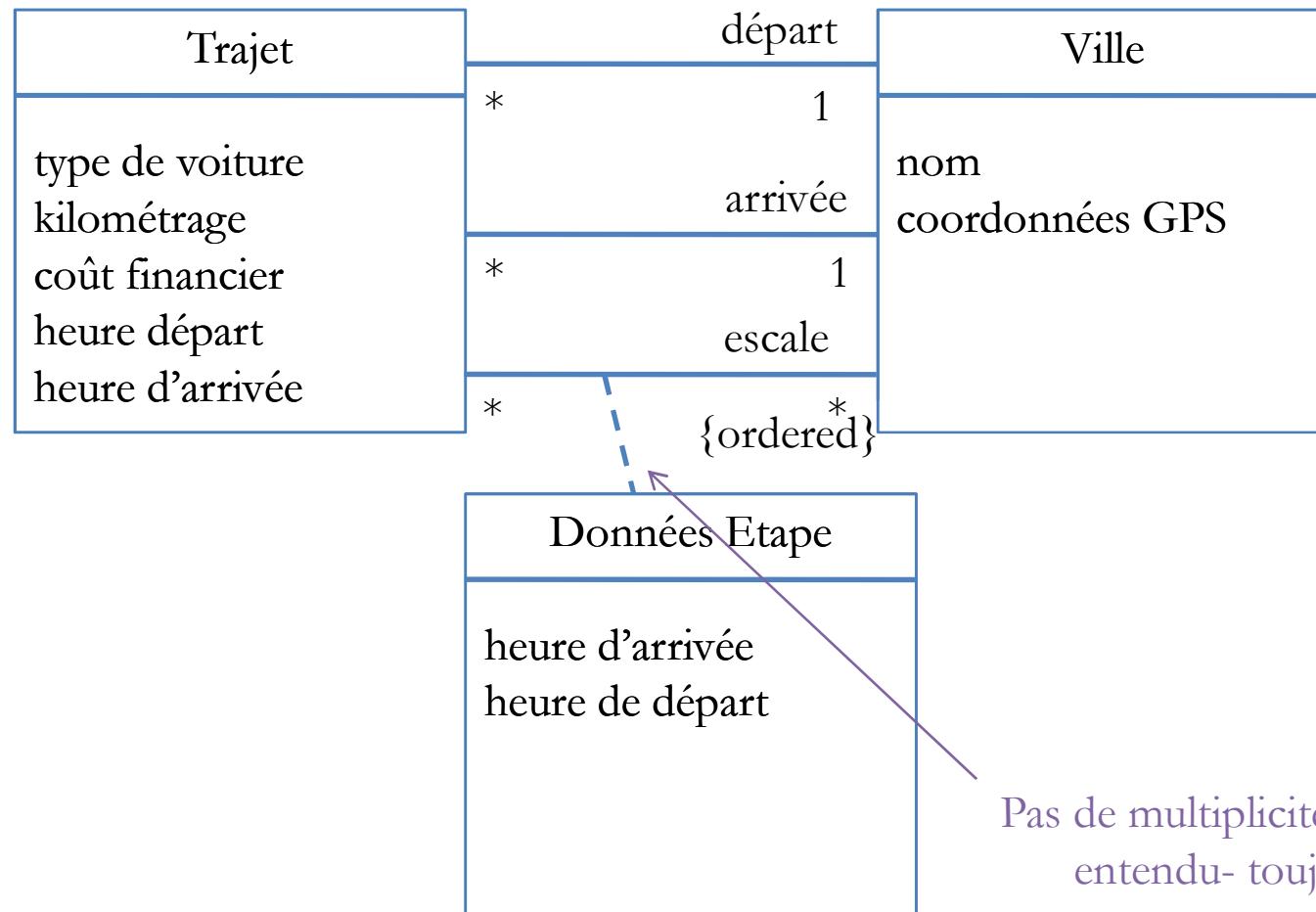
Conseils pour la généralisation/spécialisation

- On ne spécialise que lorsque cela apporte un réel gain.
- Éviter la spécialisation multicritères : ne faire la spécialisation que sur le critère le plus représentatif.
- Éviter l'héritage multiple (une sous-classe est sous-classe de deux classes différentes).
- Si le diagramme de classes est utilisé pour une implantation, le critère de spécialisation doit être explicité comme attribut de la sur-classe.

Classe-association



Classe-association



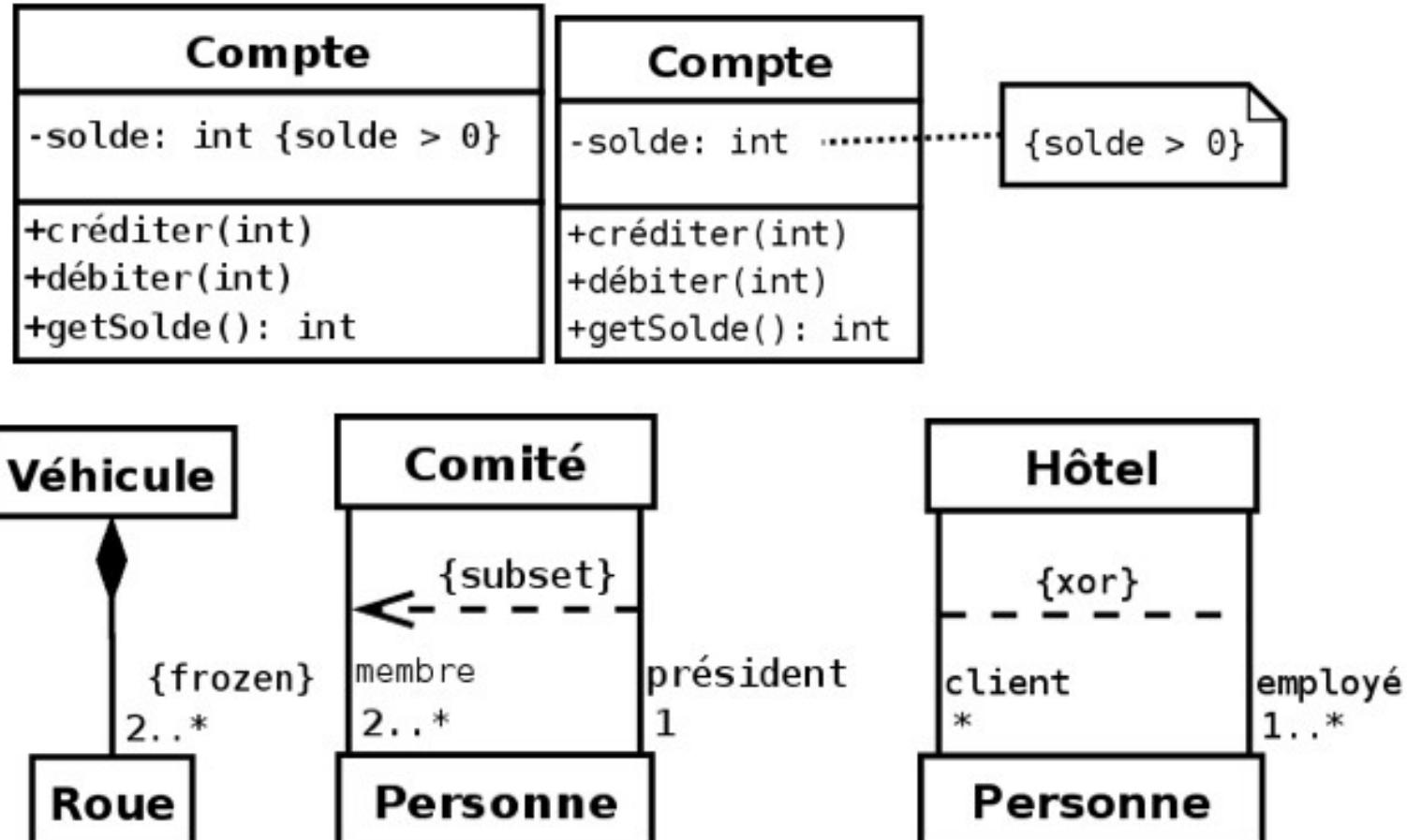
Contraintes

- Condition ou restriction sémantique posée sur un ou plusieurs éléments de modélisation.
- Elles peuvent être exprimées :
 - en langage naturel (il existe des contraintes pré-définies),
 - en OCL (formalisme au pouvoir d'expression ++),
 - dans le langage de programmation.

Pas vraiment « au choix » car dépend de la finalité du diagramme

Nous n'étudierons pas OCL
dans ce cours

Contraintes



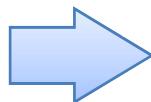
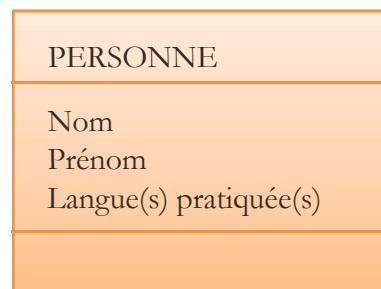
Contraintes



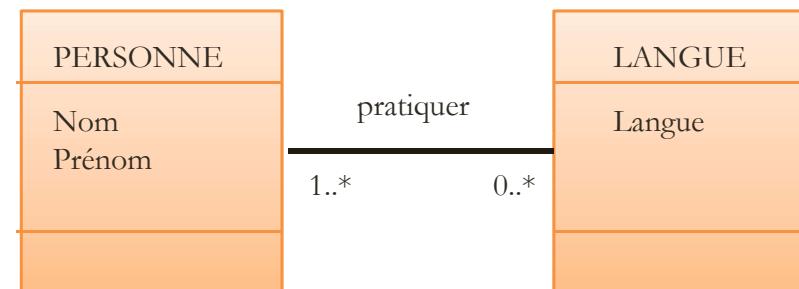
Diagramme de classes : Conseil de construction (1)

- Tout attribut dans une classe doit avoir une valeur unique pour chaque instance

NON :



OUI :



Exemple de problème engendré :
Si diagramme utilisé pour conception d'une BD relationnelle, cette classe donnerait une relation qui ne serait pas en 1NF

Diagramme de classes : Conseil de construction (2)

Problème ?

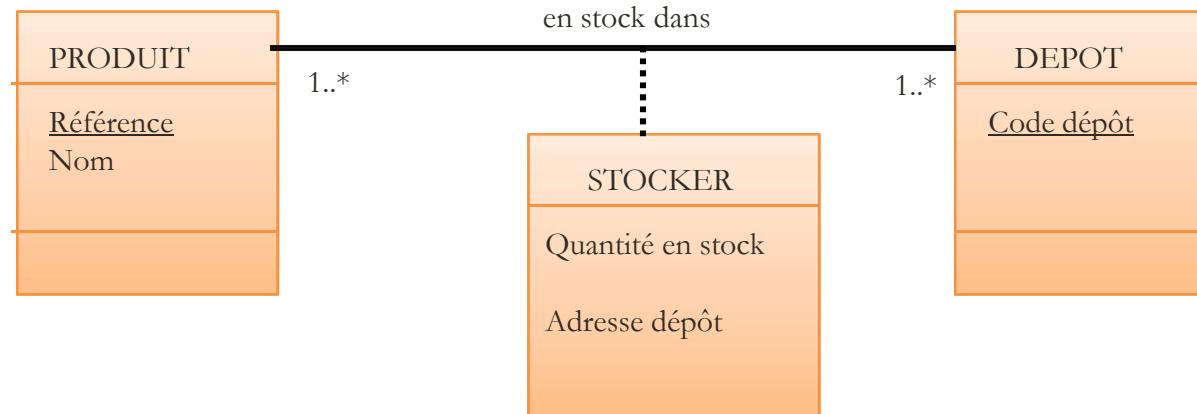
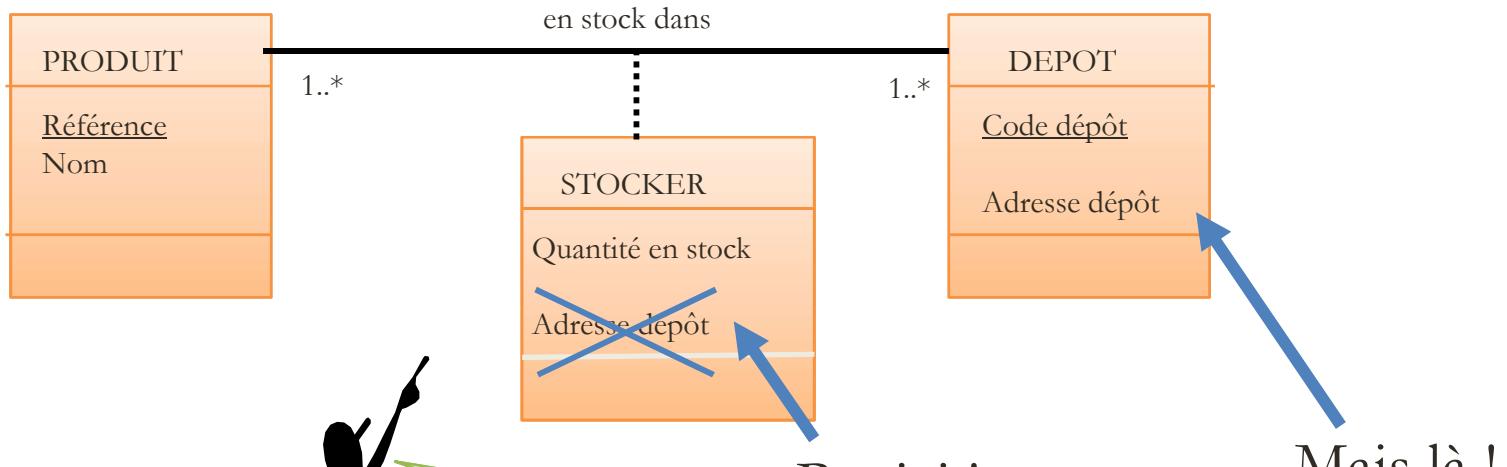


Diagramme de classes : Conseil de construction (2)

- Placer les attributs là où ils sont en dépendance fonctionnelle



C'est un principe général : on met les attributs dans la classe dont ils dépendent



Exemple de problème engendré :
Si diagramme utilisé pour conception d'une BD relationnelle, la relation STOCKER donnerait une table qui ne serait pas en 2NF

2NF... Qu'est-ce que c'est déjà ?



Dépendance fonctionnelle : définition

Une dépendance fonctionnelle (df) indique une dépendance entre *valeurs d'attributs d'une même relation*.

portée : une df est définie sur une relation R.

syntaxe : une dépendance fonctionnelle *se note* $X \rightarrow Y$ et *se lit* X détermine Y.

X et Y sont des *ensembles* d'attributs (éventuellement vides) tels que $X \cup Y \subseteq \text{att}(R)$.

sémantique intuitive : la df $X \rightarrow Y$ associée à une relation R indique que *tous* les tuples de R ayant la même valeur sur X ont aussi la même valeur sur Y.

Dépendance fonctionnelle : définition

notation : $t[U]$ est la valeur du tuple t sur l'ensemble d'attributs U .

sémantique formelle : Si $X \rightarrow Y$ est une dépendance fonctionnelle définie sur une relation R alors toute instance *valide* r du schéma de R satisfait $X \rightarrow Y$ ($r \models X \rightarrow Y$) :

$$\forall t_1, t_2 \in r \text{ tq } t_1[X] = t_2[X], \text{ on a } t_1[Y] = t_2[Y].$$

exemple : Une relation **Employé(Nom, Prénom, Fonction, Service)** et la dépendance fonctionnelle **{Nom,Prénom} → {Service}**

⇒ une personne est associée à un seul service (mais peut avoir plusieurs fonctions dans ce service).

2NF : définition et exemple

Définition :

une relation R avec son ensemble F de df est en 2NF ssi

1. R est en 1NF
2. tout attribut n'appartenant pas à une clef de R ne dépend pas d'un sous-ensemble d'attributs d'une clef.

exemple : soit la relation

Cours (Resp:chaîne, Matière:chaîne, Tel:chaîne, Intitulé:chaîne, VolumeH:entier)

avec la df : Resp → Tel.

Cours est en 1NF mais n'est pas en 2NF à cause de cette df.

2NF : définition et exemple

Cours	Resp	Matière	Tel	Intitulé	VolumeH
	Bidoit	BD	4567	Bases de données	20
	Bidoit	BD	4567	BD1	24
	Goasdoué	BD	7890	Introduction aux BD	18
	Rousset	IA	6789	IA1	24

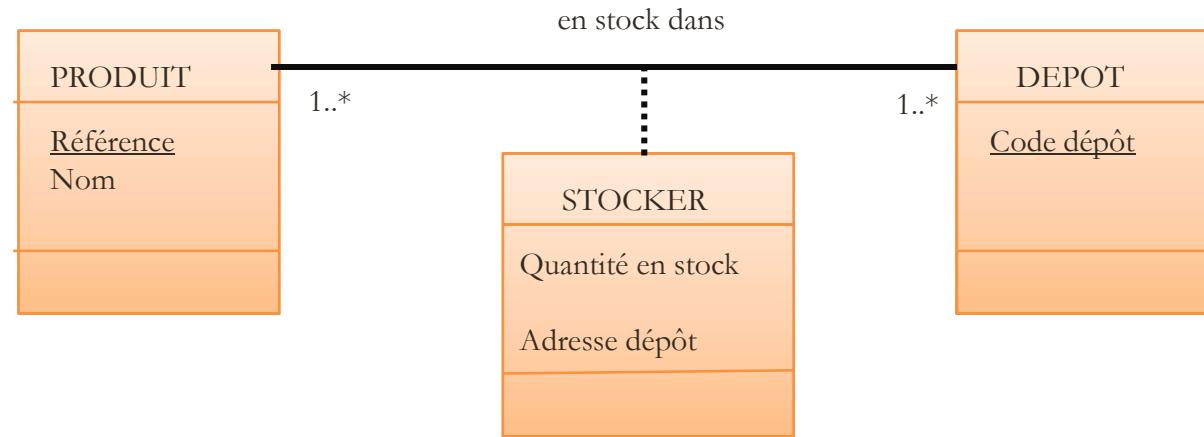
anomalies :

- redondance due à la df : si une personne est responsable de N cours, on stocke N fois le même numéro de téléphone.
- insertion : on ne peut pas stocker le numéro de téléphone d'une personne qui ne serait pas responsable d'un cours
- suppression : si une personne n'est plus responsable d'un cours, on ne stocke plus son numéro de téléphone

2NF... Qu'est-ce que c'est déjà ?



Diagramme de classes : Conseil de construction (2)



Que se passe-t-il lors du passage au relationnel ? (simplifié, voir cours de BD pour les détails)

PRODUIT(Référence, Nom)

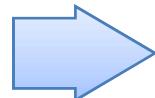
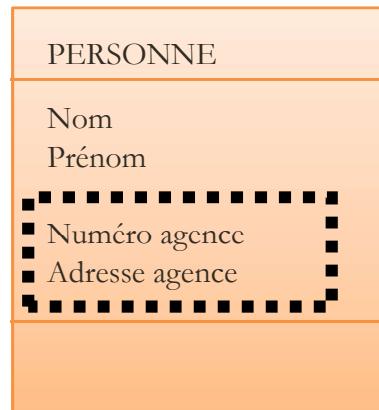
DEPOT(Code dépôt)

STOCKER(Référence, Code depot, Quantité, Adresse_depot)

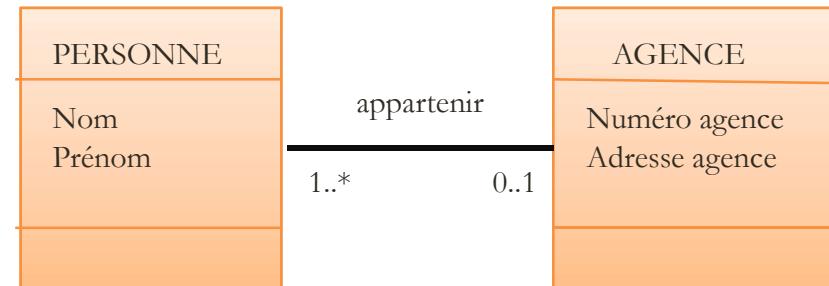
Diagramme de classes : Conseil de construction (3)

- Ne pas inclure une classe dans une autre classe

NON :



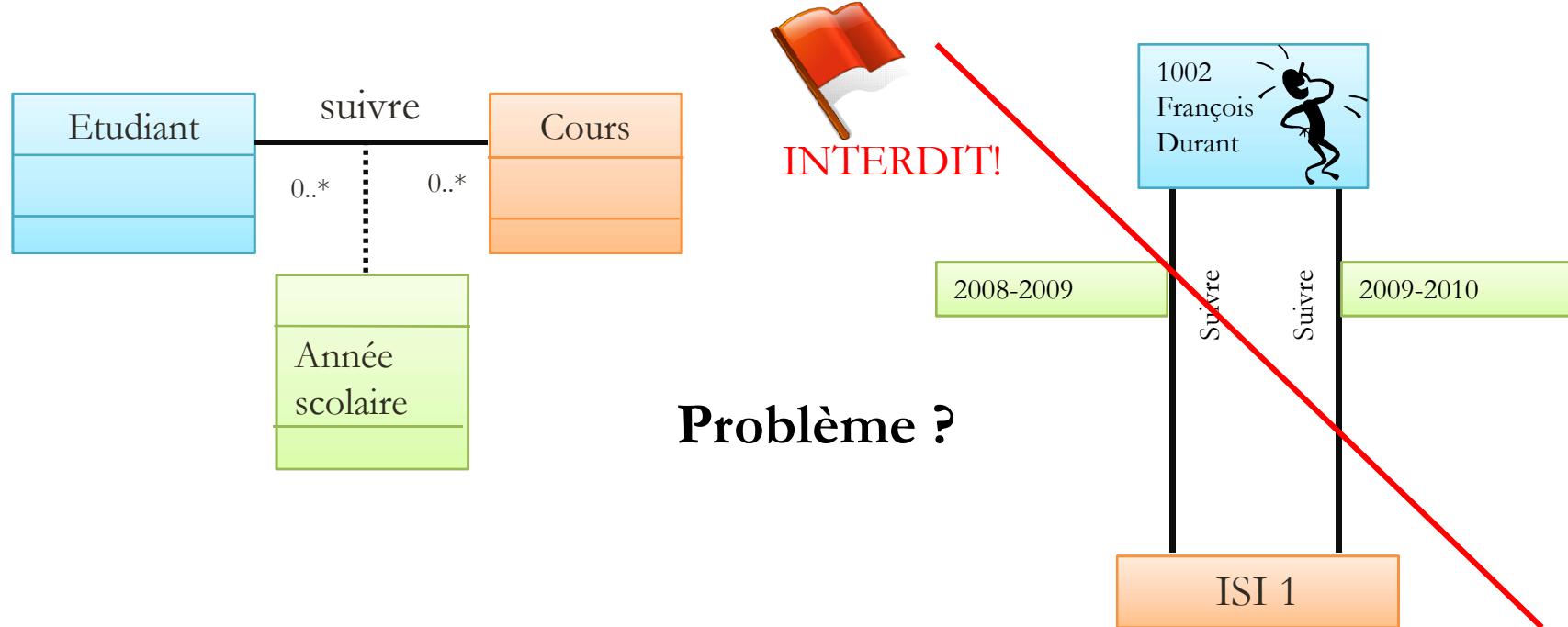
OUI :



Une telle classe implantée dans une BD relationnelle donnerait une table qui ne serait pas en 3ème Forme Normale

Diagramme de classes : Impératif de construction (4)

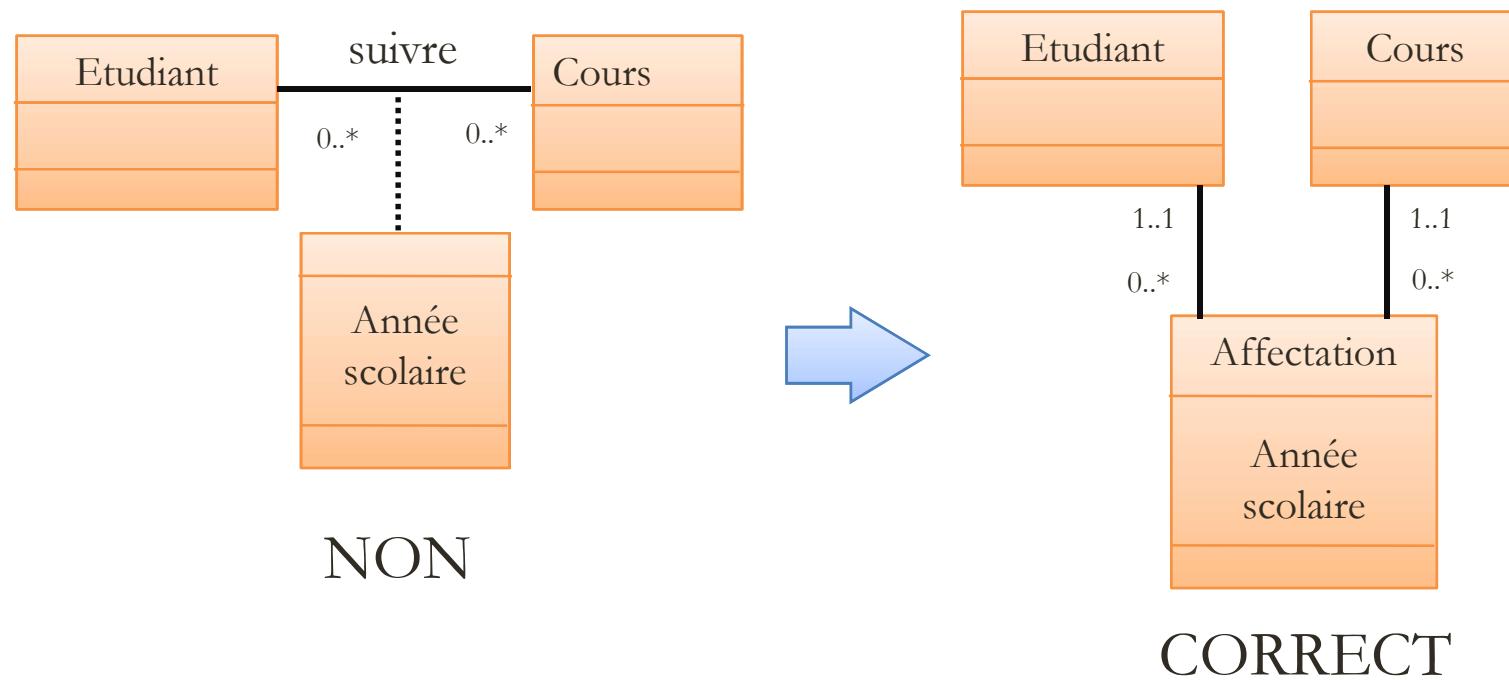
- On ne peut pas avoir deux liens entre les mêmes instances de classes au sein de la même association donc



Idée sous-jacente : l'année scolaire faire partie de la clé, elle ne peut pas apparaître dans une classe-association

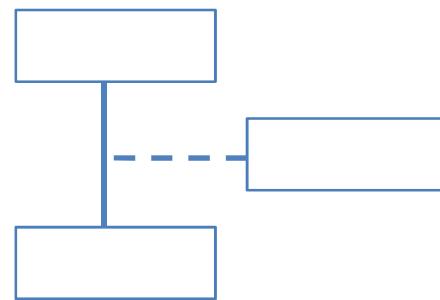
Diagramme de classes : Impératif de construction (4)

- On ne peut pas avoir deux liens entre les mêmes instances de classes au sein de la même association donc

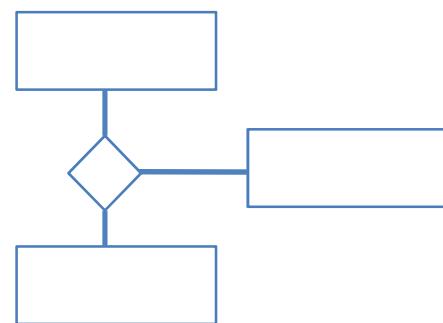


J'ai trois classes à associer...

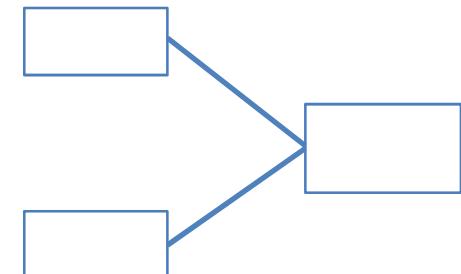
J'ai trois classes à associer.



classe association ?



association ternaire ?

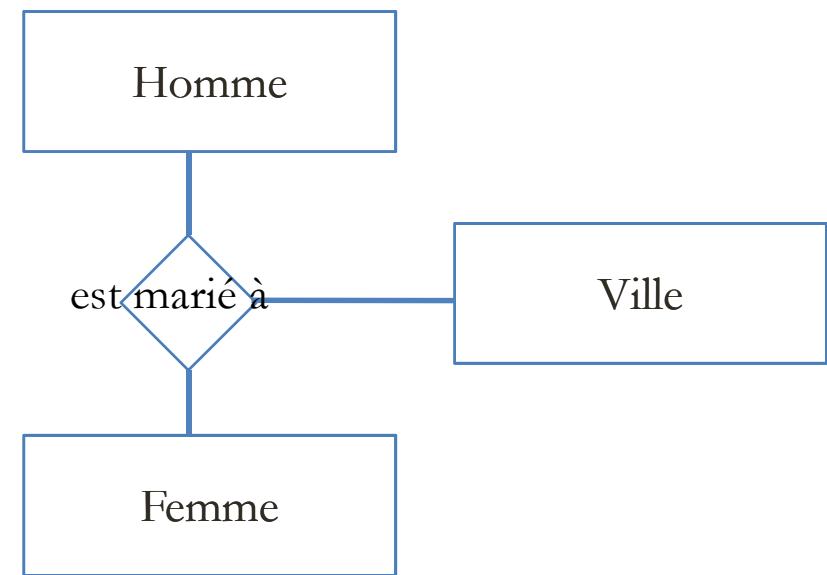
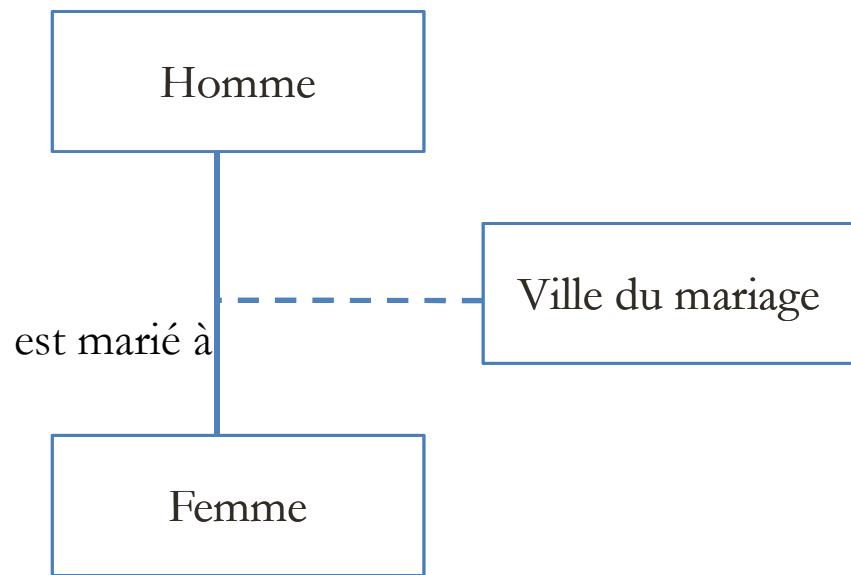


deux associations ?



Classe-association ou association ternaire ?

Exemple 1

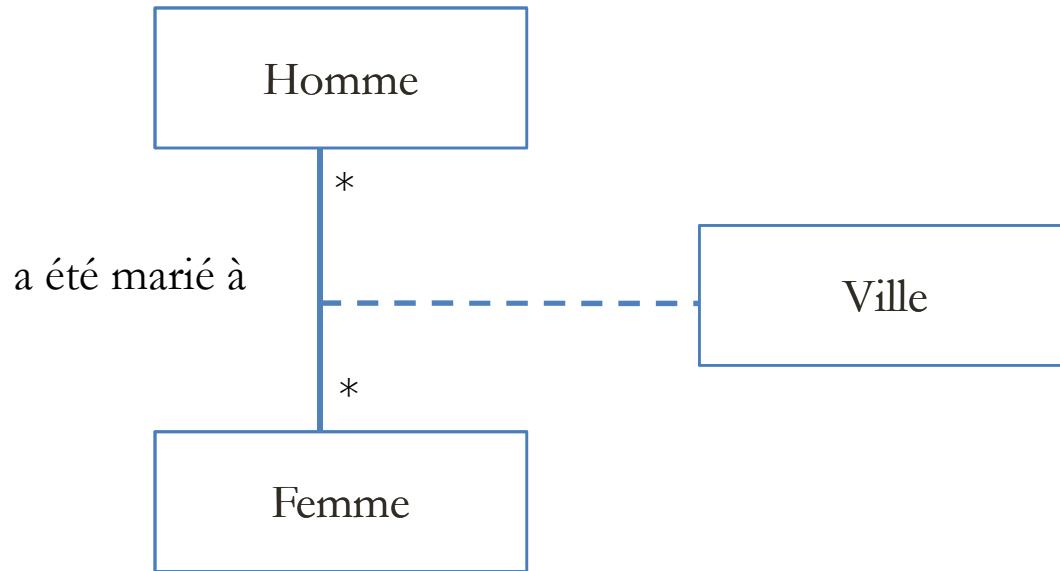


Quand utiliseriez-vous l'une ou l'autre de ces représentations ?

Exercice au passage : poser les multiplicités

Classe-association ou ... ?

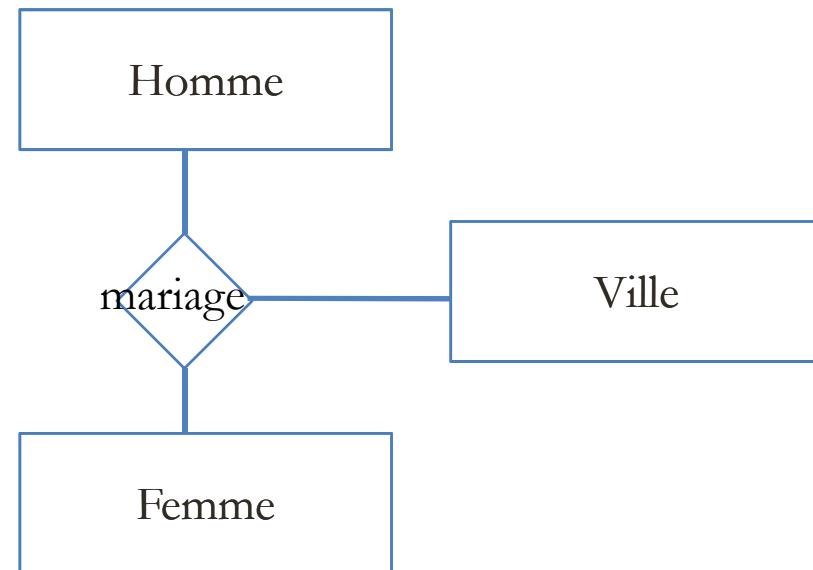
Exemple 2



Sachant qu'on ne peut pas avoir deux liens entre les mêmes instances de classes au sein de la même association

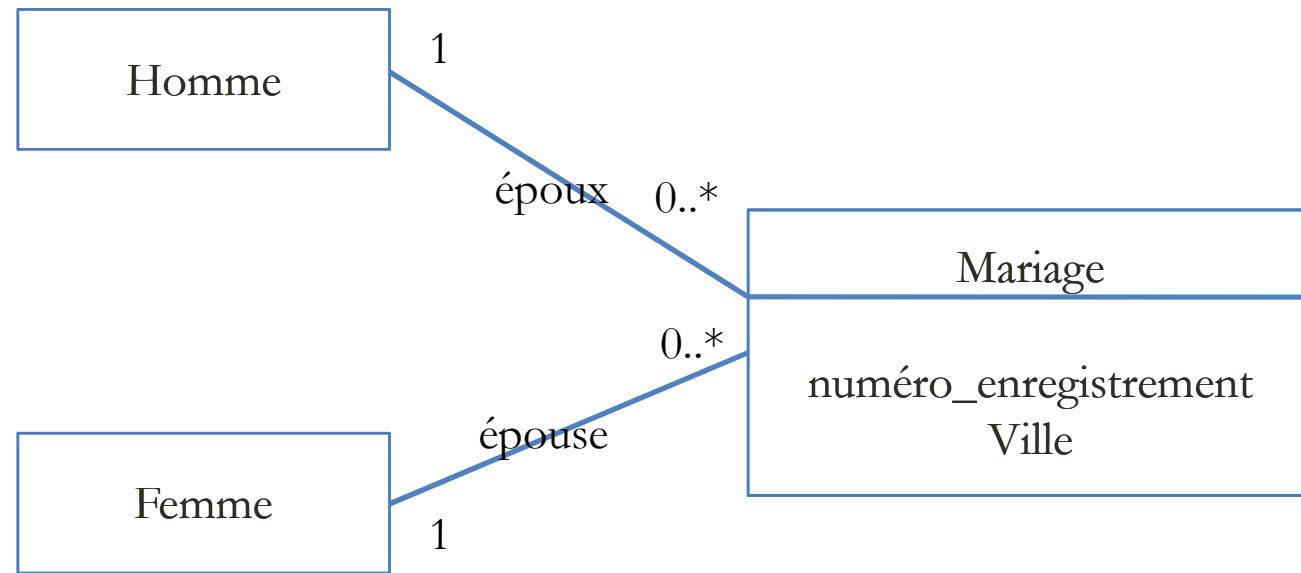
Qu'implique cette représentation ?

Classe-association ou association ternaire ?



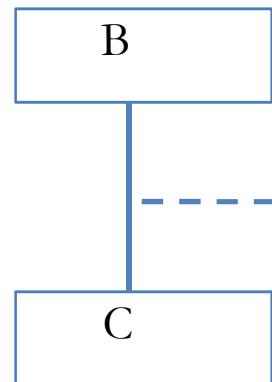
Le problème est-il résolu ?

Classe-association ou deux associations ?

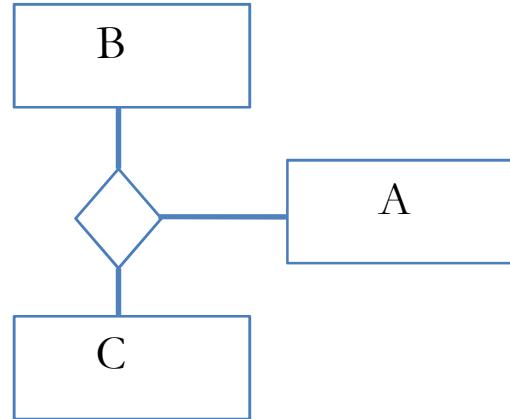


Problème résolu !

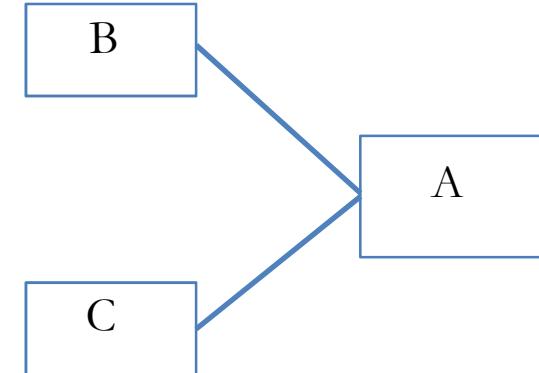
J'ai trois classes à associer...



classe association ?



association ternaire ?



deux associations ?

- A n'existe que via l'association
- On ne peut pas associer deux fois un couple (b,c)

A existe indépendamment de B et C

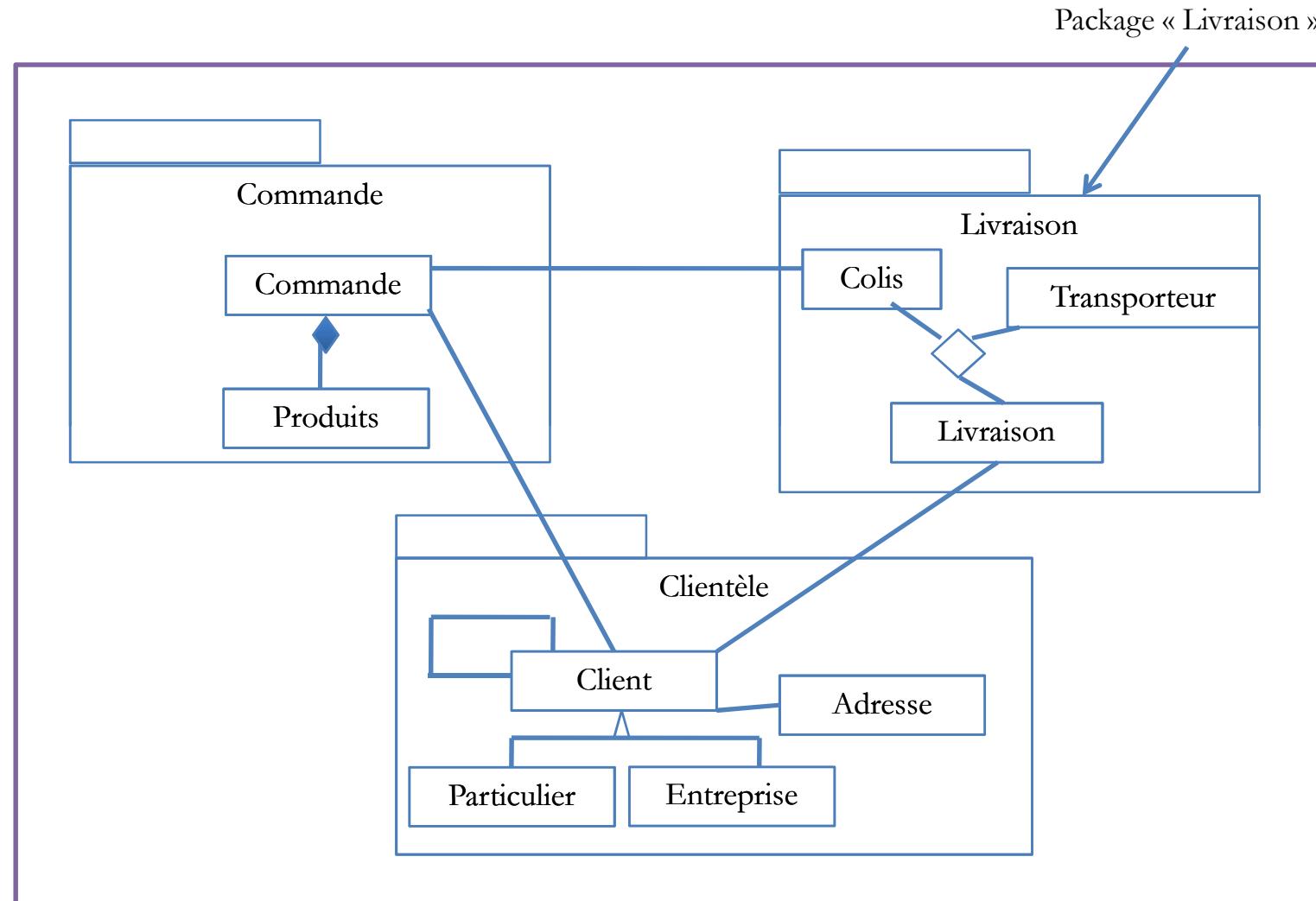
On ne peut pas associer deux fois
un triplet (a,b,c)

A choisir entre une ass. ternaire et deux ass. binaires en phase d'analyse, privilégier deux ass. binaires (plus lisible). → Négociation exactitude vs. lisibilité.

Paquetage

- Mécanisme de regroupement des éléments du diagramme.

Paquetage



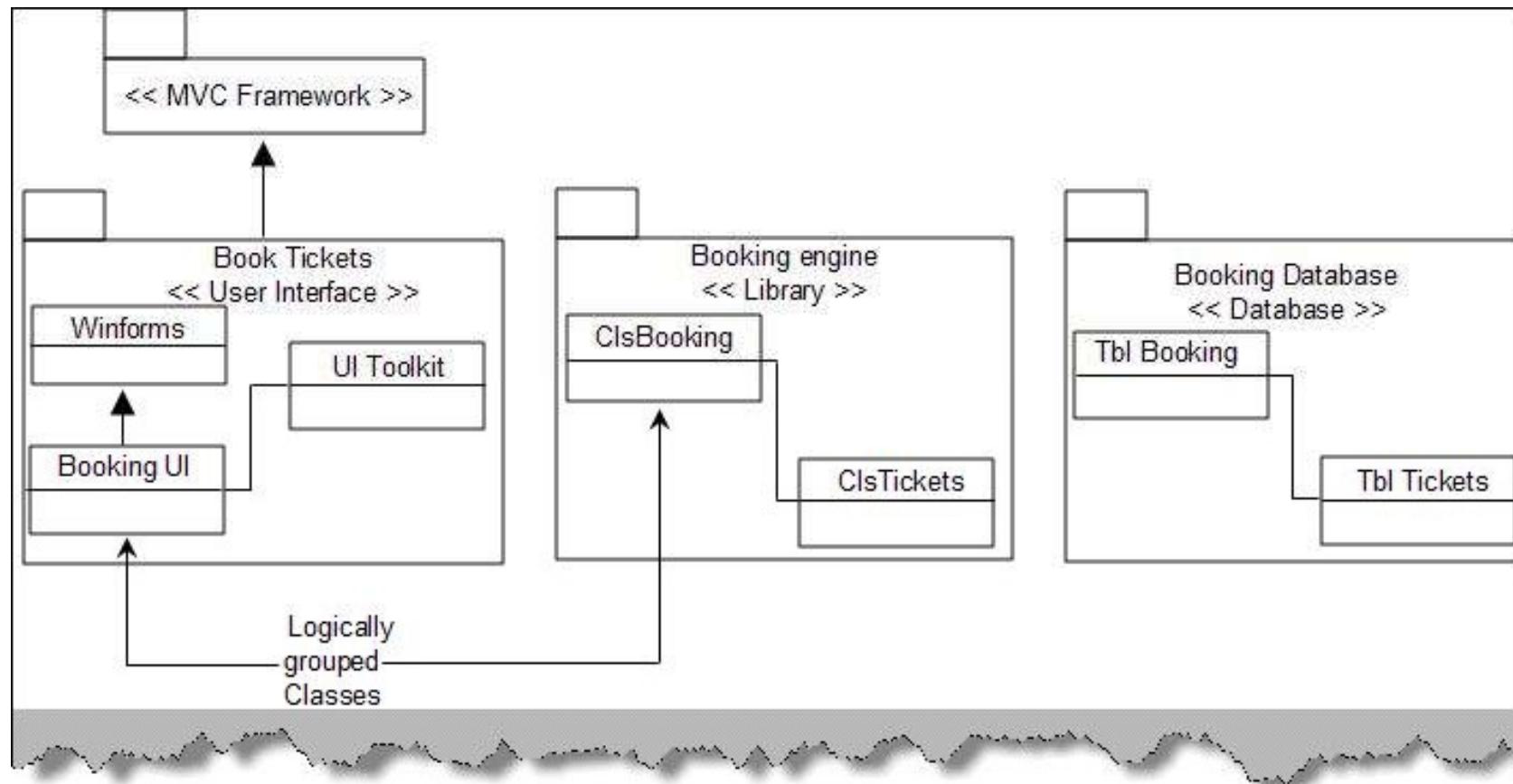
Paquetage

Permet de :

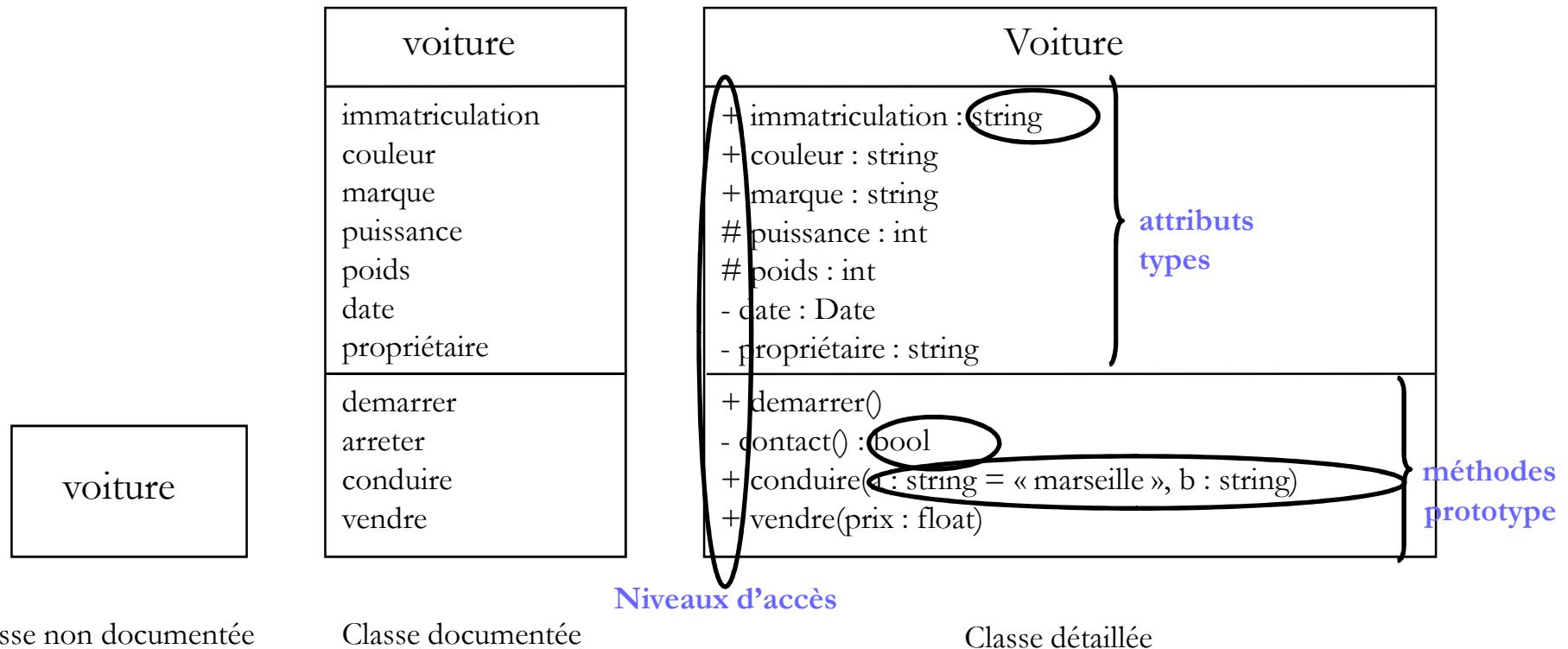
- organiser logiquement (sémantiquement) des diagrammes de classes complexes, ← très utilisé dans les diagrammes de classes d'analyse
- organiser du code, ← seulement dans un diagramme classes de conception évidemment
- faire apparaître sur le diagramme un plus haut niveau d'abstraction lié à l'architecture,
- faire apparaître sur le diagramme un plus haut niveau d'abstraction lié aux *use cases*.

Dépend du sens de l'abstraction que vous désirez faire apparaître

Paquetage : regroupement par use case



Classe : différents niveaux d'abstraction



Choix du niveau d'abstraction

Le niveau de détail du diagramme de classes est à choisir en fonction de la phase de la conception à laquelle il est utilisé (voir *unified process* plus loin dans le cours) → dépend de son objectif.

Deux objectifs très différents par exemple :

- en phase d'analyse : modèle de classes pour validation entre MOA et métier (modèle d'analyse),
- en phase de conception : modèle de classes utilisé par la MOE pour représenter la structure d'un code et le générer (modèle de conception).

Classe : différents niveaux d'abstraction

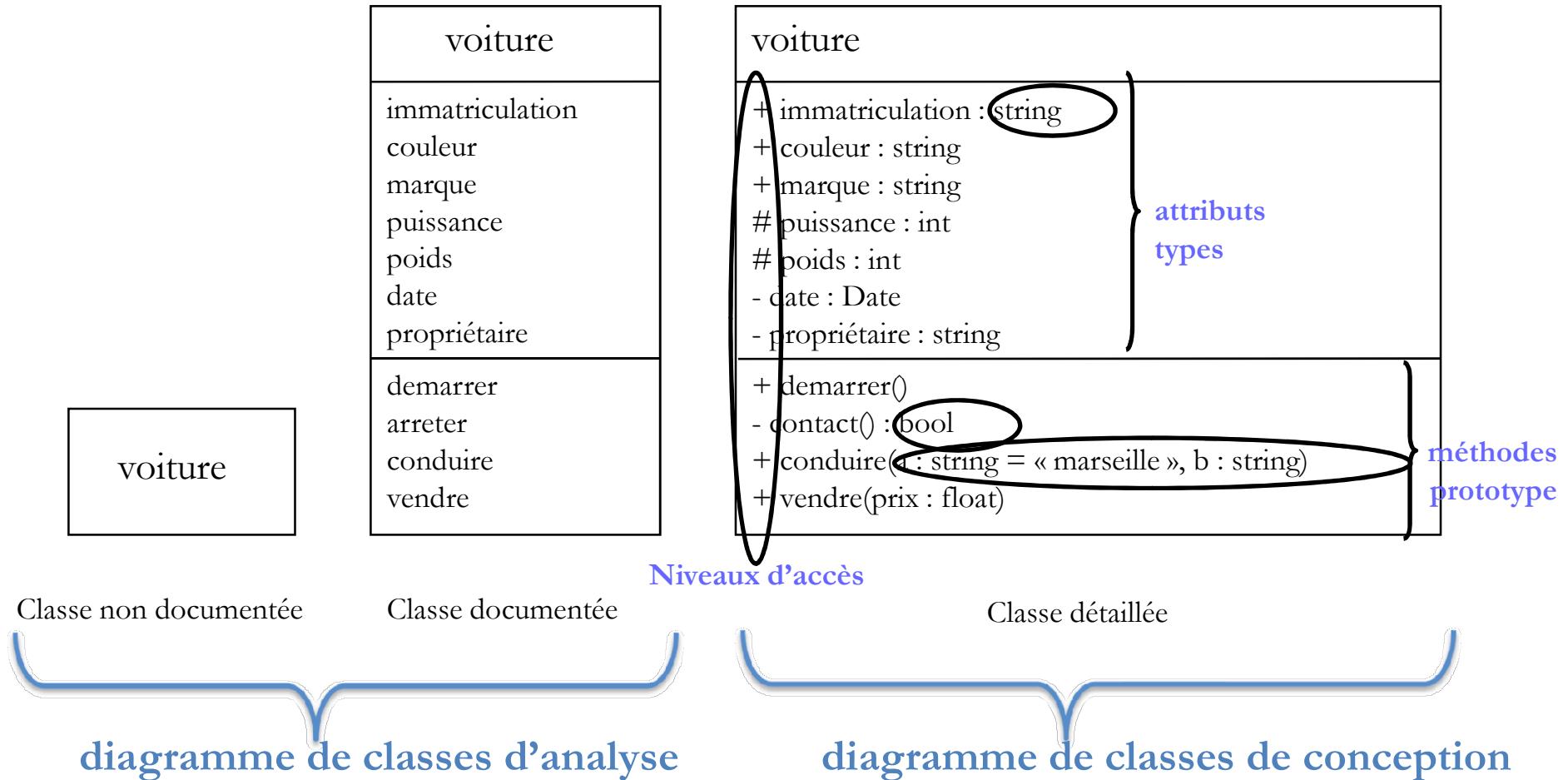


Diagramme de classes : Conseils pratiques

- Bien comprendre le problème à résoudre, la qualité du modèle en dépendra directement
- Avoir une liste précise des informations manipulées avec leur signification sémantique exacte afin de déterminer les classes, les associations et les multiplicités
- Éviter les associations n-aires avec $n > 2$
- Ne pas surcharger le modèle global qui doit rester clair et lisible, si telle ou telle partie doit être plus détaillée (spécialisation,...), faire un paquetage que l'on détaillera

diagramme d'objets

OBJECT DIAGRAM

Diagramme d'objets

- Le diagramme d'objets montre des instances de classes avec leurs liens, en cohérence avec la structure du diagramme de classe (exemple ou contre-exemple). Vue statique des objets.
- Il est utile pour
 - vérifier la bonne structuration du diagramme de classes avec des exemples d'instances,
 - dégager les multiplicités pour les structures complexes (associations n-aires),
 - faciliter la compréhension de la structure du diagramme de classes par des personnes non familiarisées avec le formalisme.

Diagramme d'objets

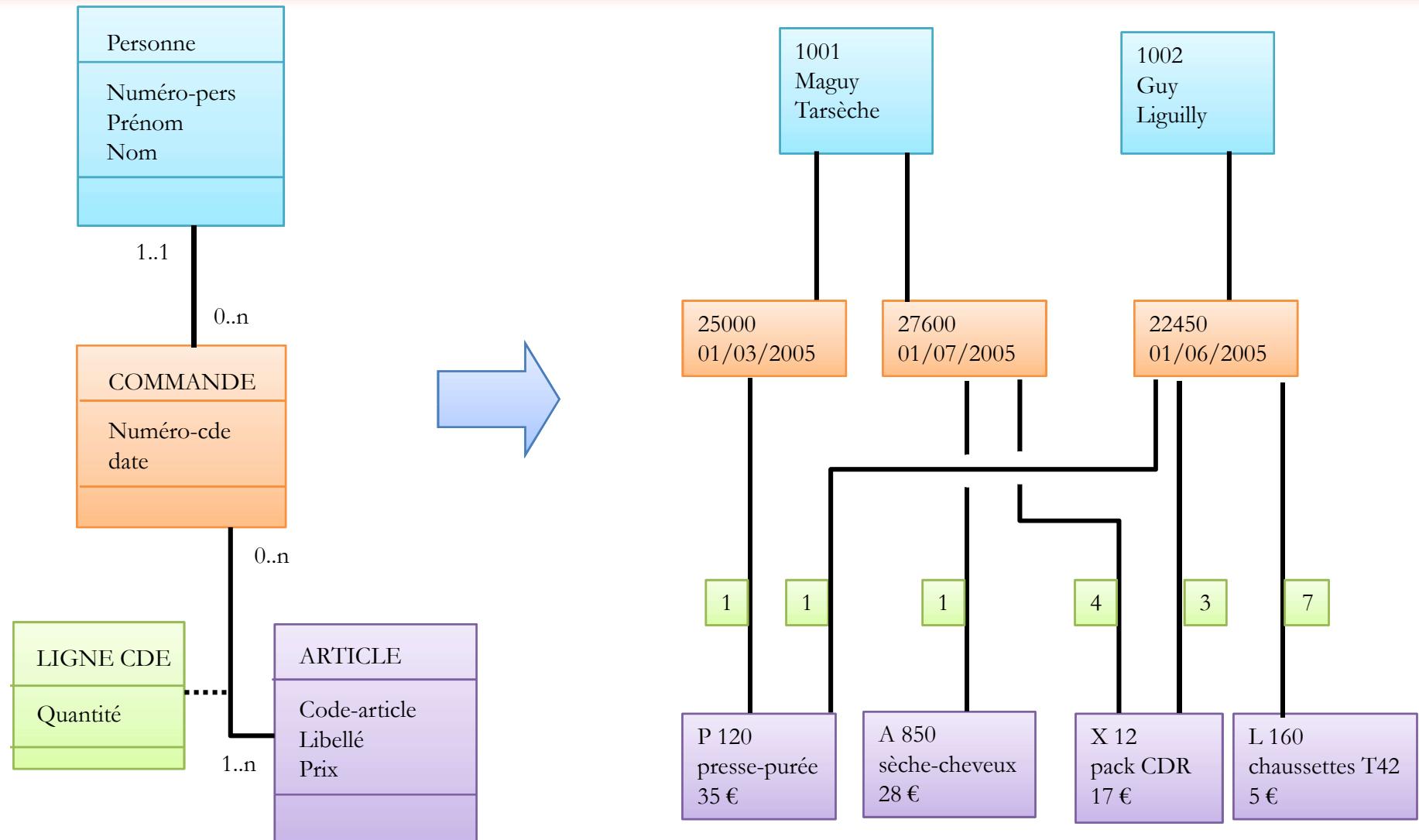


Diagramme d'objets

- Autre exemple : le diagramme d'objets permet de vérifier la bonne structure du diagramme de classe

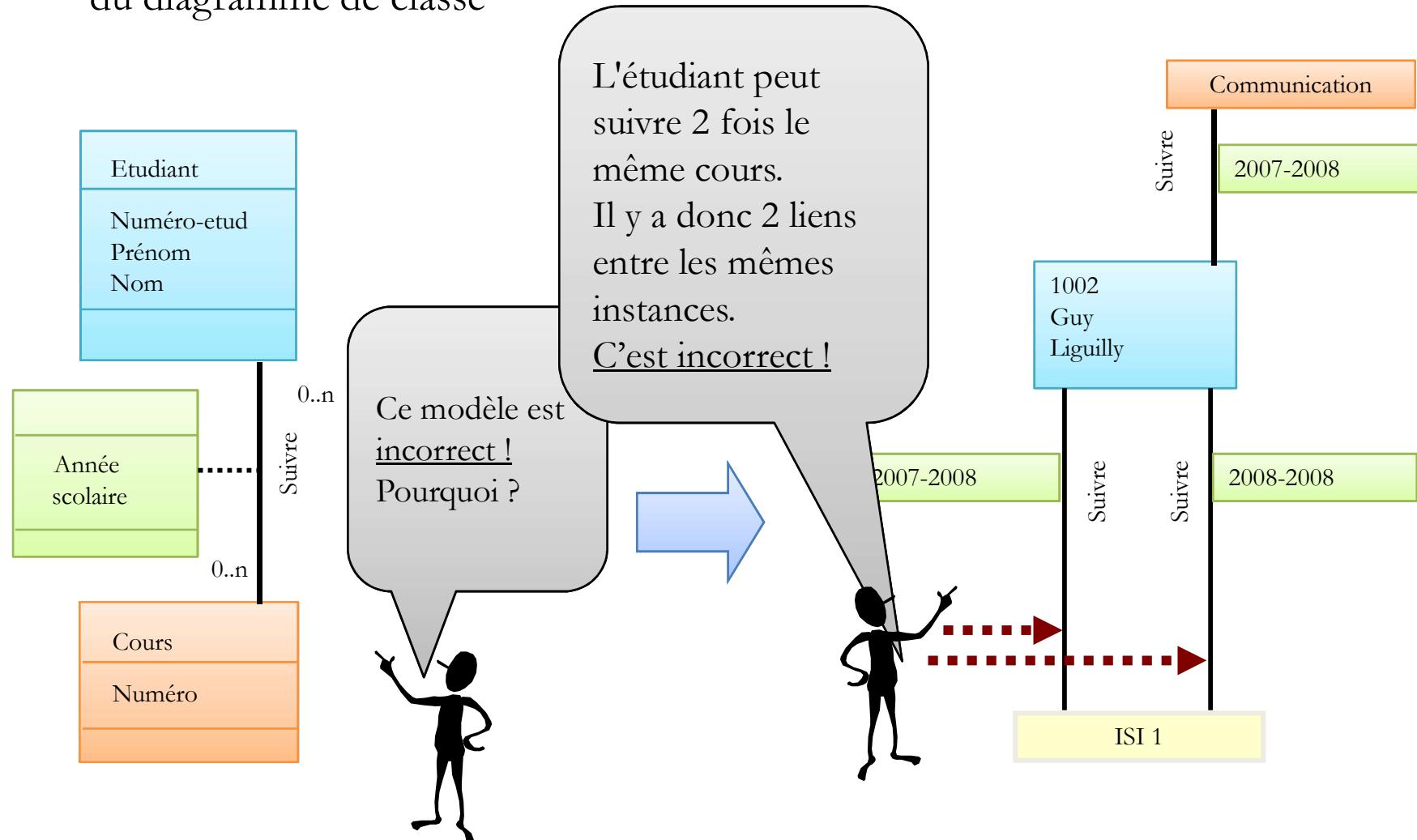


Diagramme d'objets

- Une bonne structuration serait

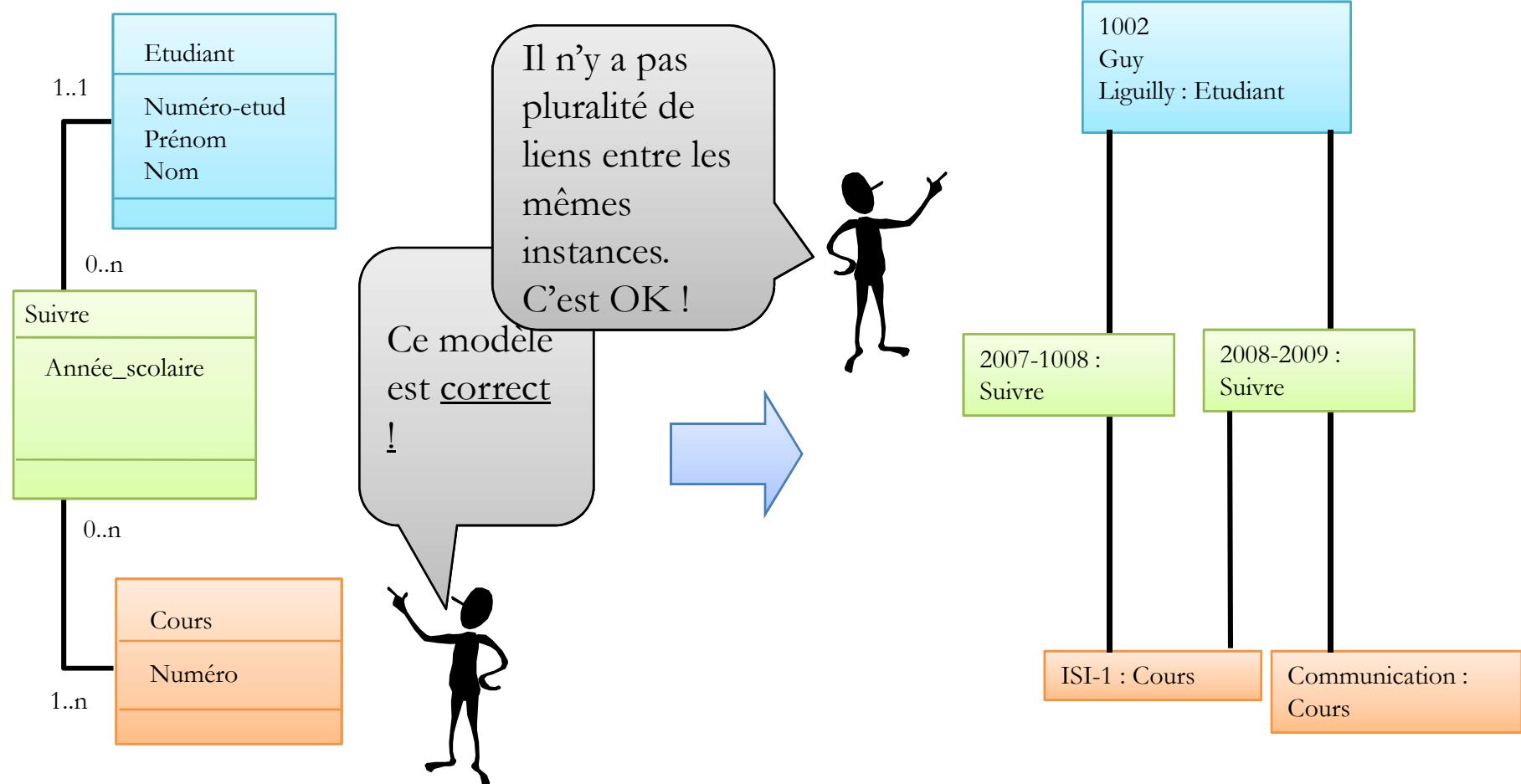


Diagramme d'objets

- Il peut aider à déterminer des multiplicités...

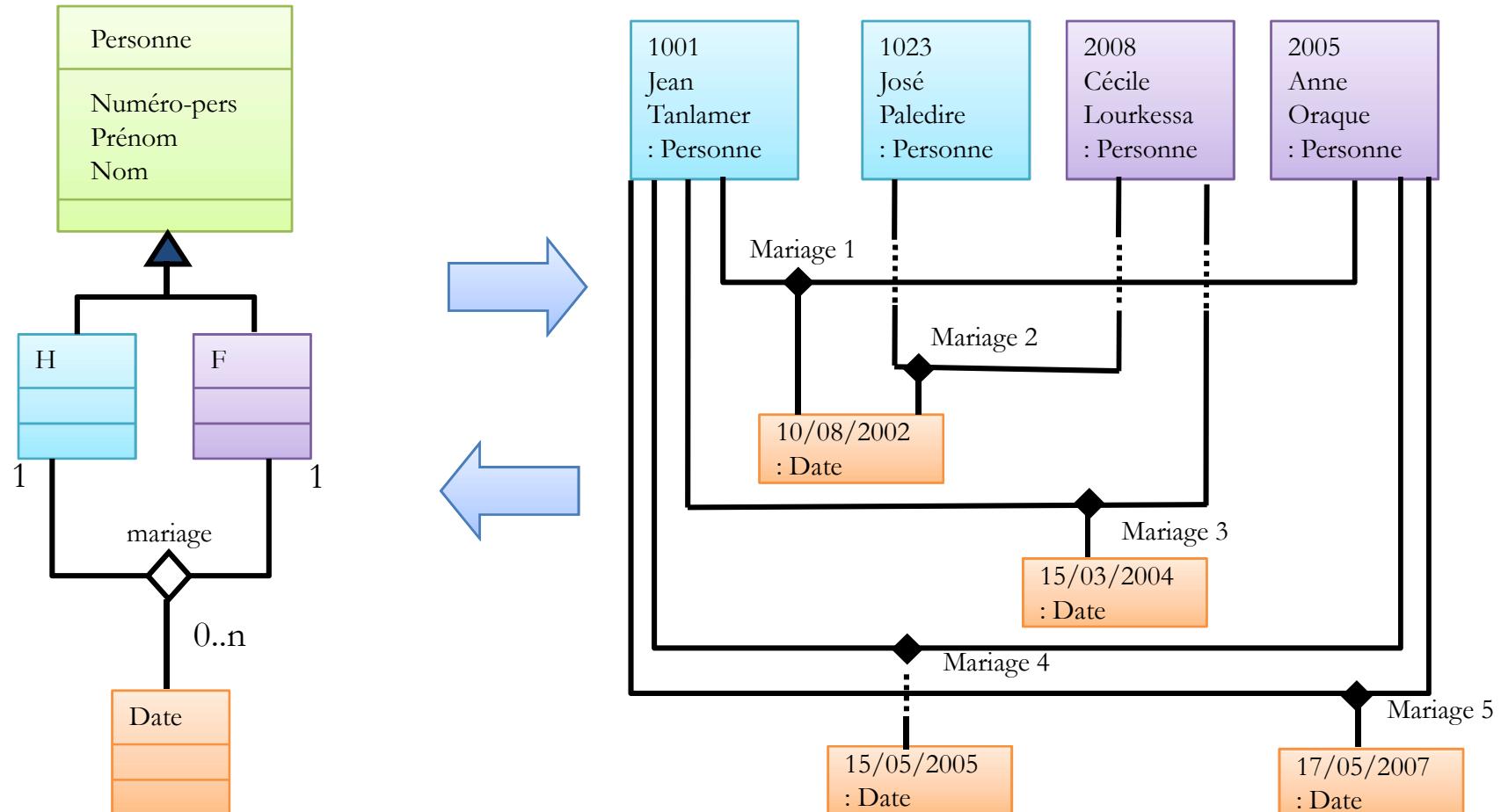


Diagramme d'objets

La syntaxe du diagramme d'objet peut être plus compliquée (par exemple spécifiant le type et la valeur des attributs d'un objet).

→ *Ne complexifier la syntaxe que si cela apporte une réelle plus-value.*

Diagramme d'objets



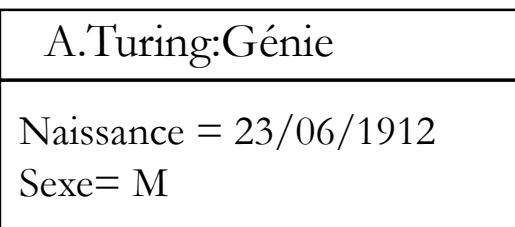
Instance anonyme de la classe Génie



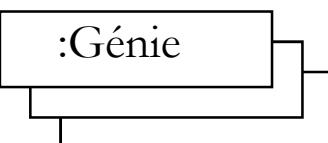
Instance nommée de la classe Génie



Instance nommée d'une classe anonyme



Spécification des attributs et de leur valeur



Collection d'instances

diagramme de séquence

SEQUENCE DIAGRAM

Diagramme de séquence

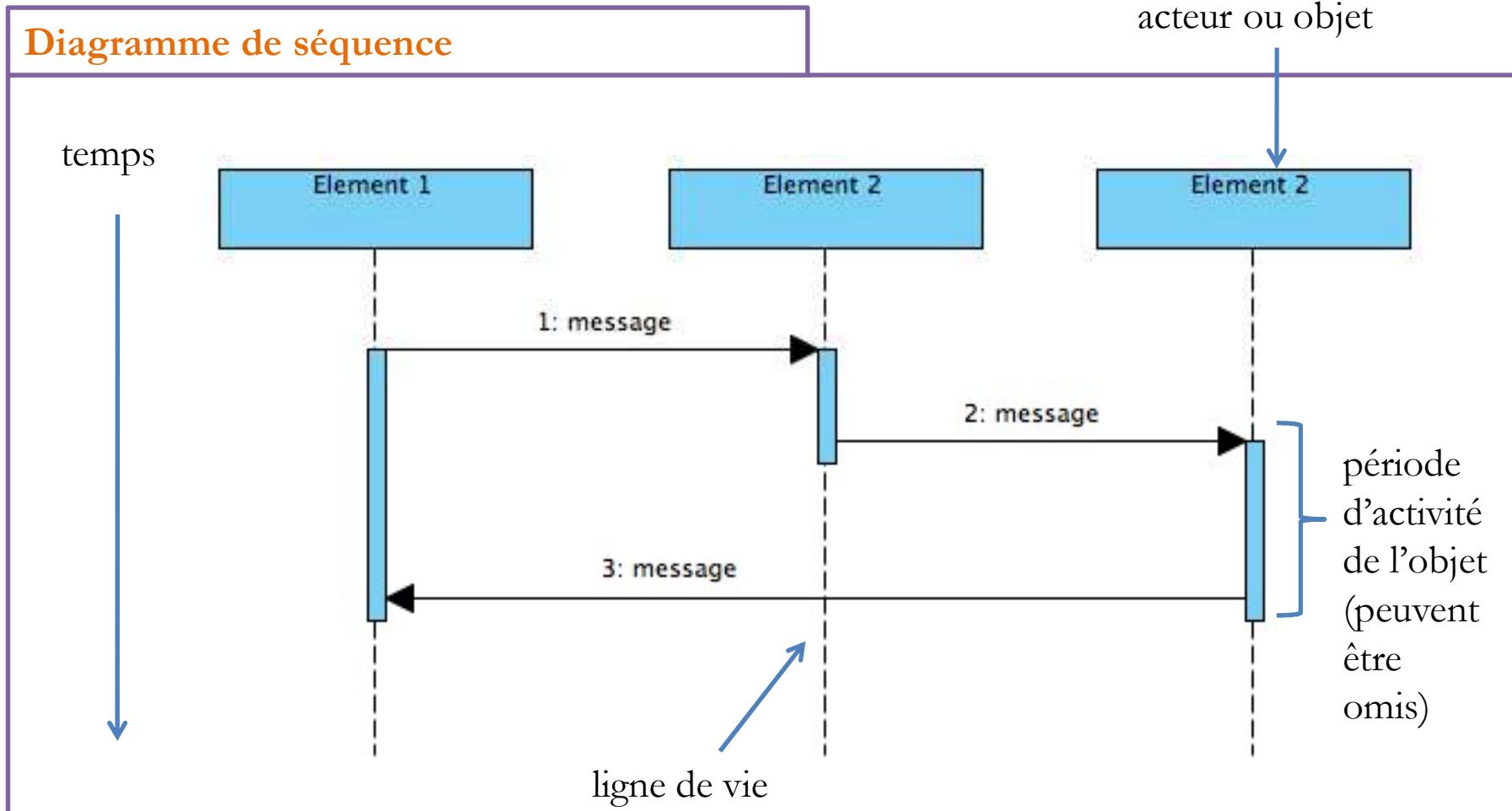
- Représenter des interactions entre objets
- Chronologie (ordre des interactions)

dynamique

Permet par exemple de :

- illustrer un scenario de cas d'utilisation (ex: retirer de l'argent)
- modéliser les traces d'exécution d'un test

Diagramme de séquence : forme générale

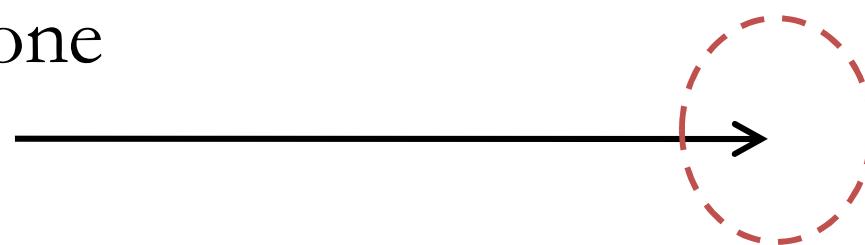


sd Pascal et Manu

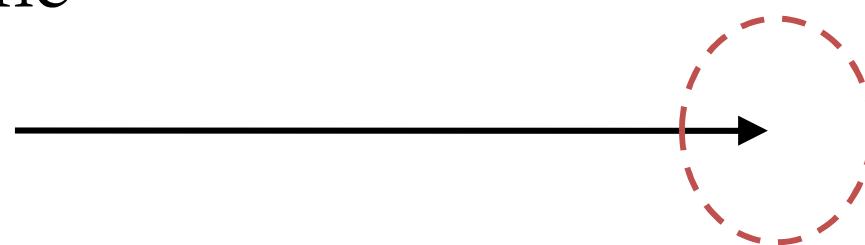


Deux principaux types

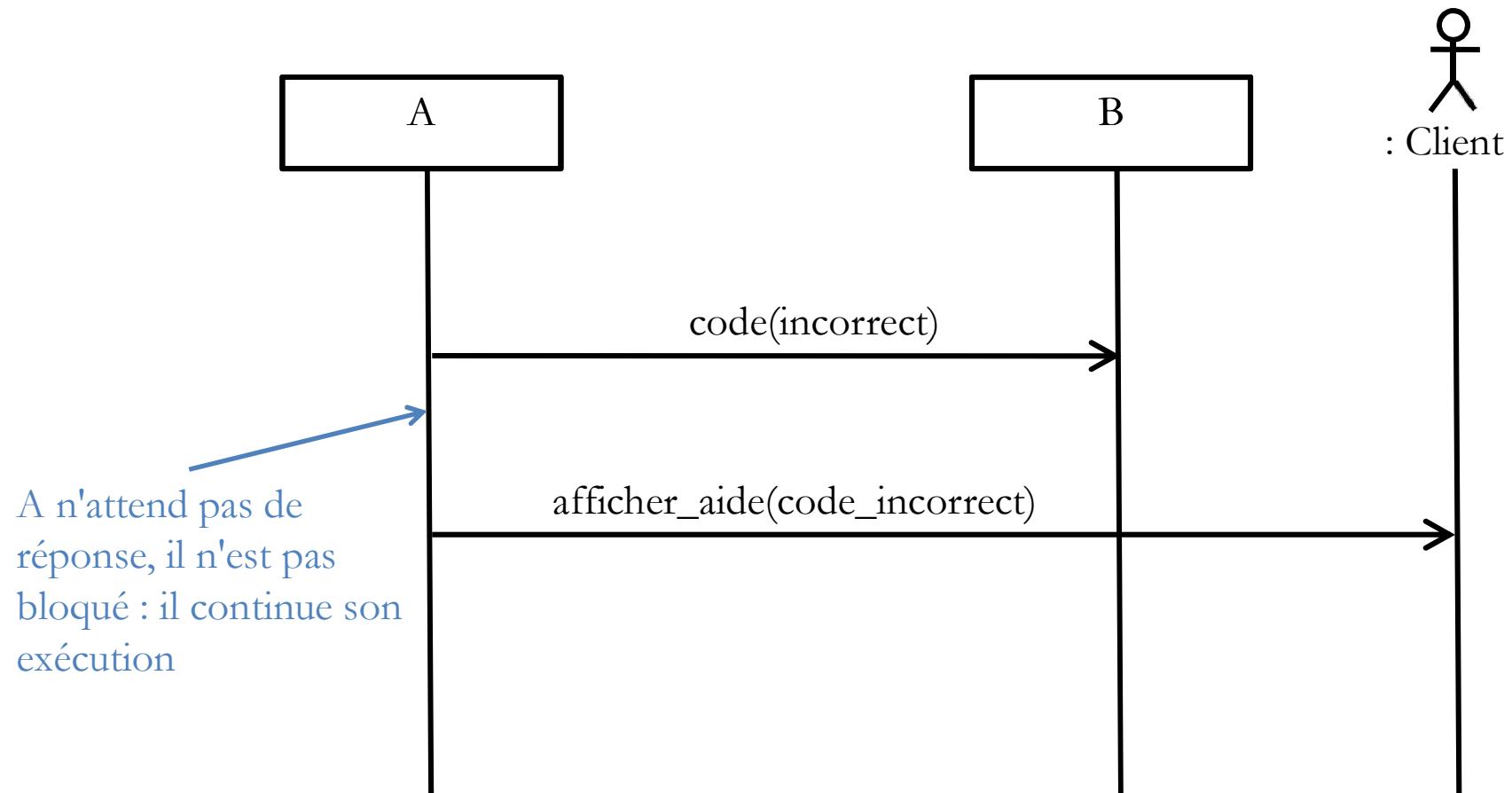
- Message asynchrone



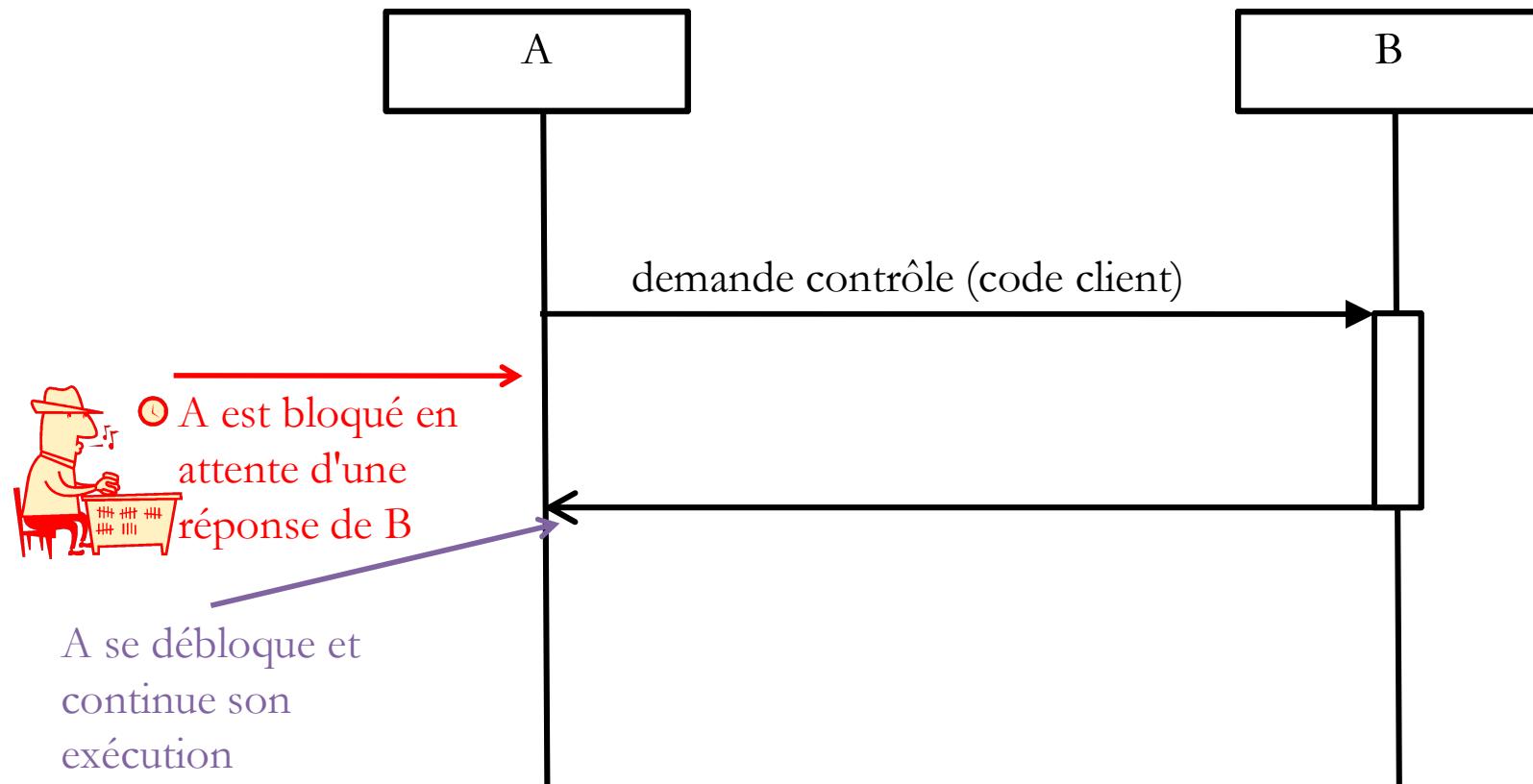
- Message synchrone



Message asynchrone

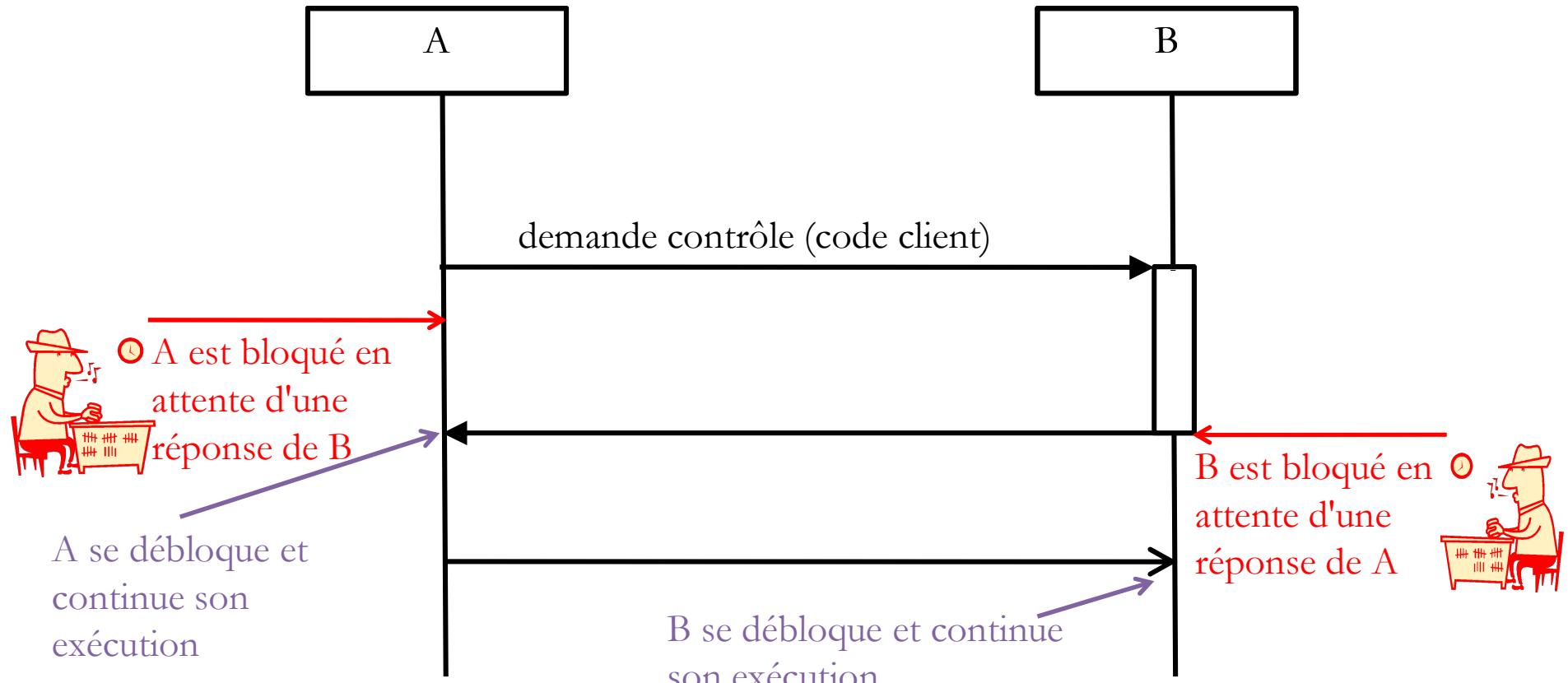


Message synchrone (1^{er} cas)

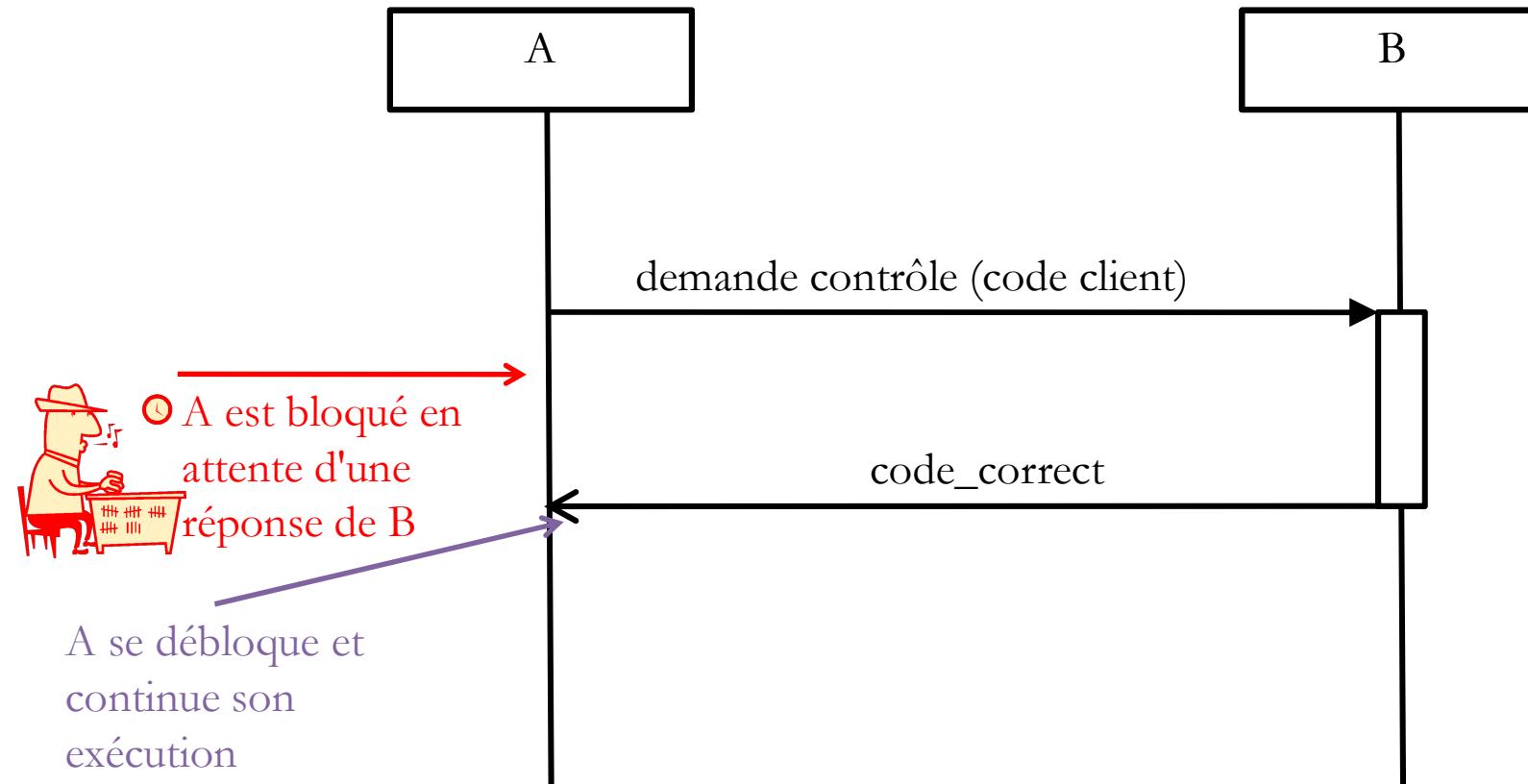


Les messages

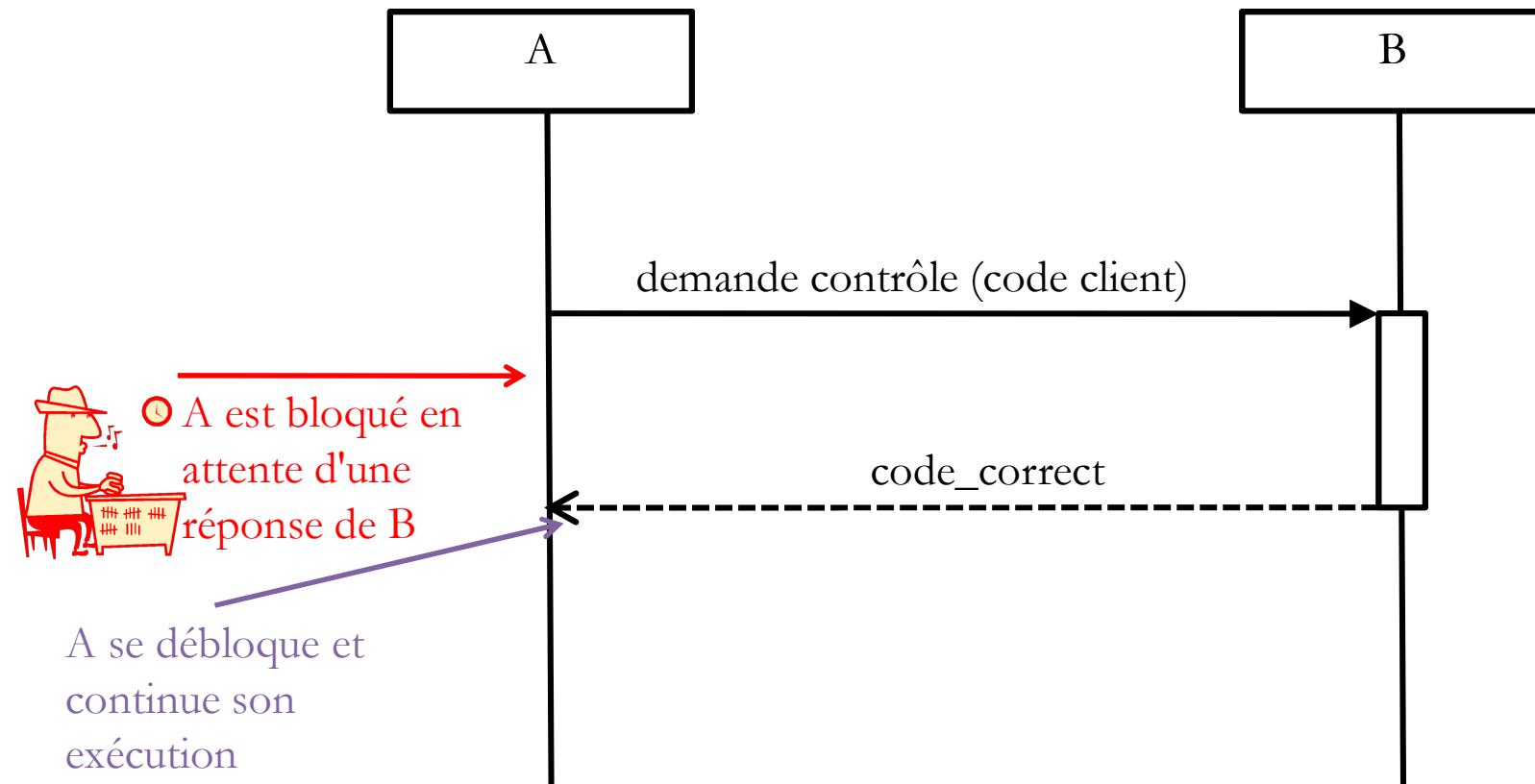
Message synchrone (2^{ème} cas)



Message synchrone (représentation 1)



Message synchrone (représentation 2)



Message synchrone (représentation 3)

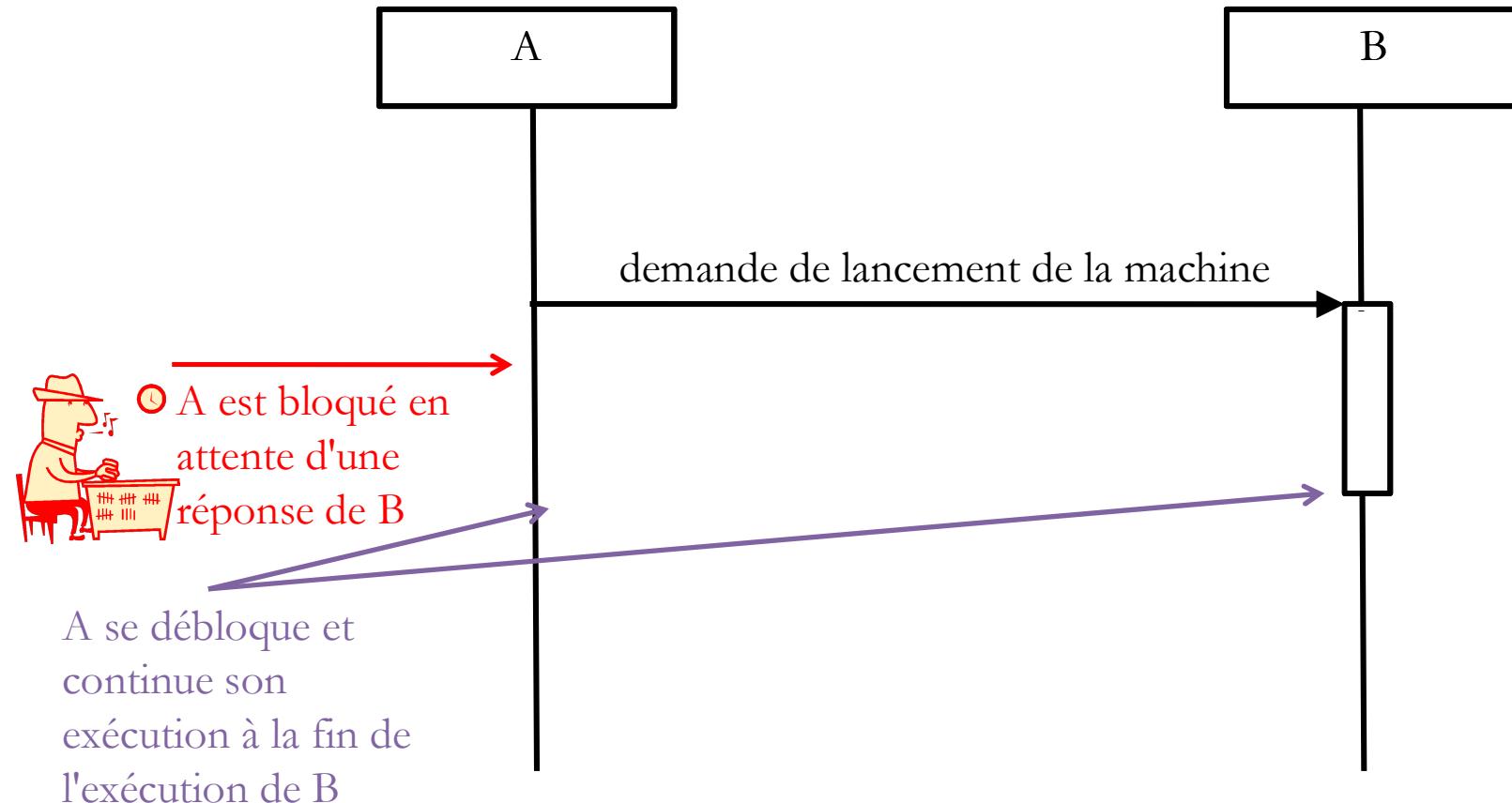
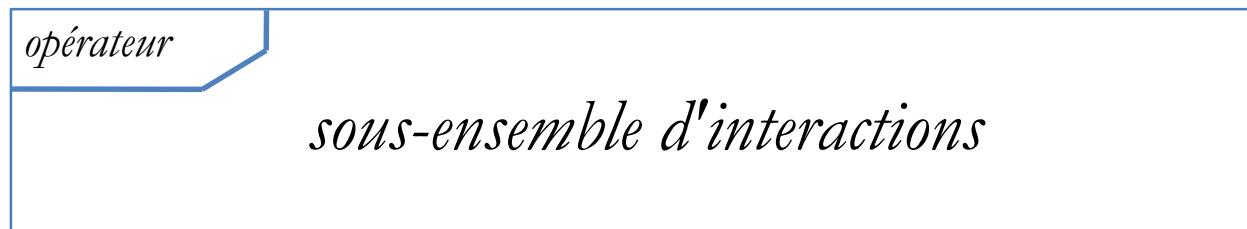


Diagramme de séquence

Cadres (ou fragments) d'interaction



Opérateurs les plus utilisés (parmi les treize définis dans UML) :

- loop : répète le fragment tant que condition
- ref : référence un autre diagramme de séquence
- alt : différentes alternatives selon condition
- opt : fragment optionnel
- par : fragments effectués en parallèle

Diagramme de séquence : opérateur *alt*

alt :

Instruction de test
si-alors-sinon ou avec
une ou plusieurs
alternatives

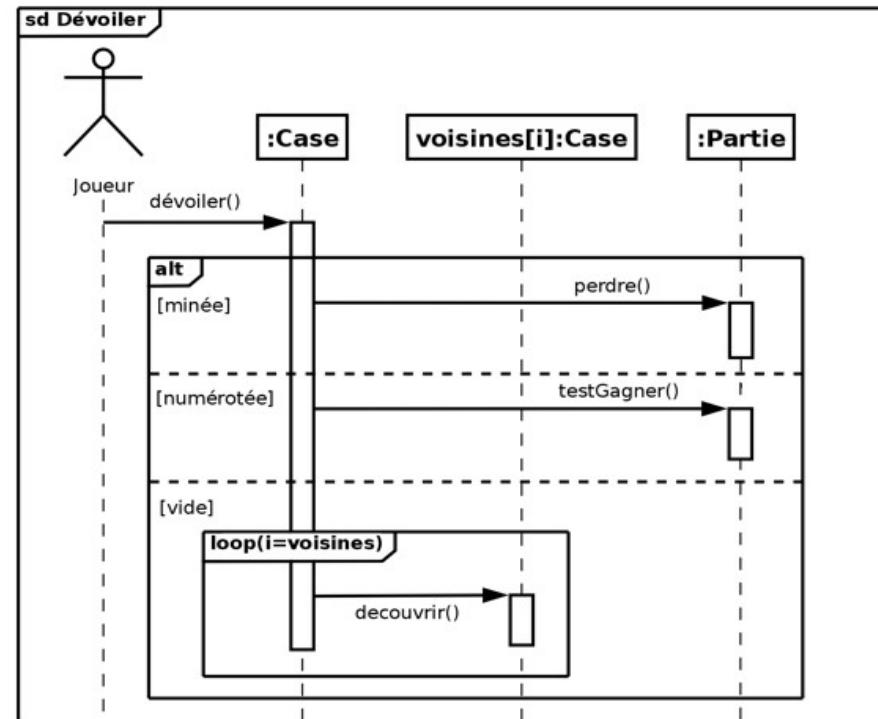


Figure 7.15: Représentation d'un choix dans un diagramme de séquence illustrant le découvertement d'une case au jeu du démineur.

Diagramme de séquence : opérateur *loop*

loop :

boucle sur la sous-séquence tant qu'une condition est satisfaite

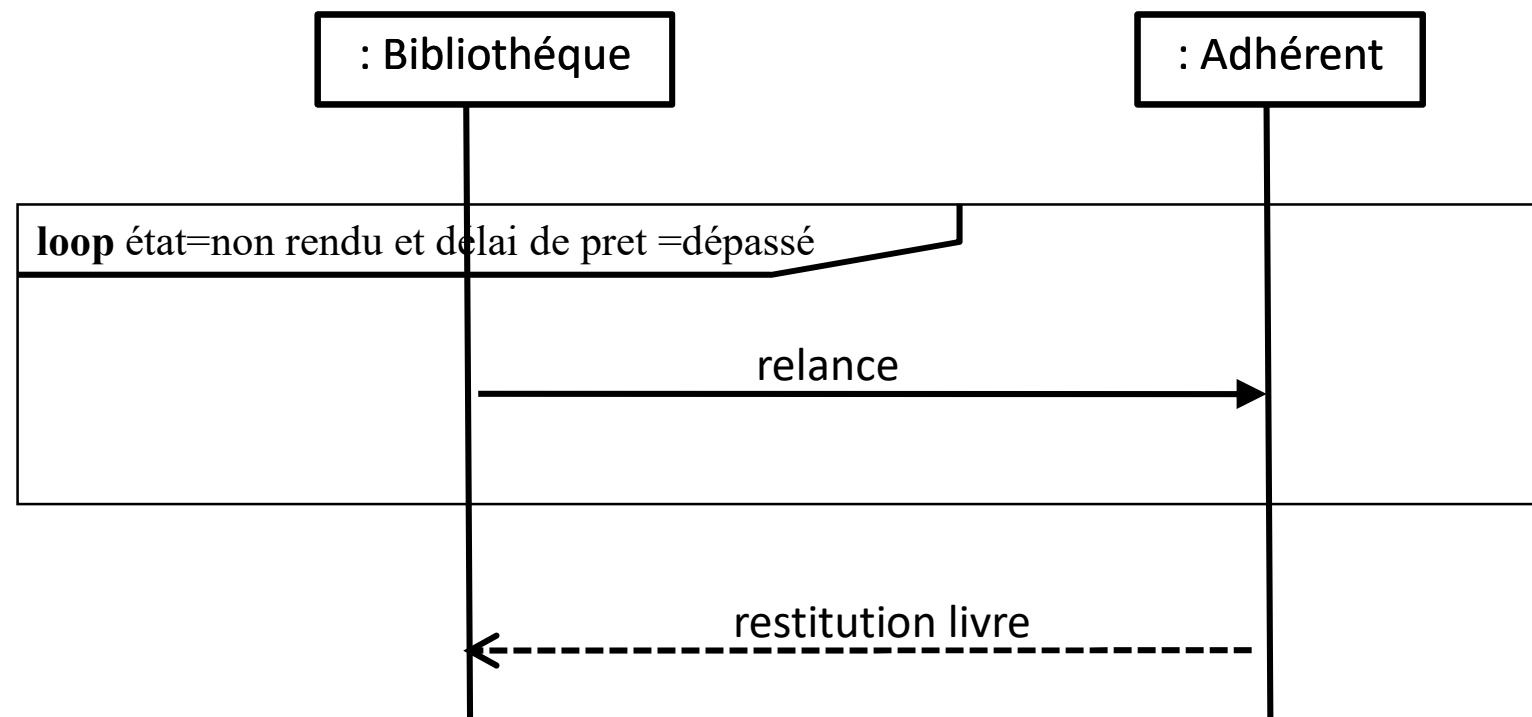


Diagramme de séquence : opérateur *loop*

loop :

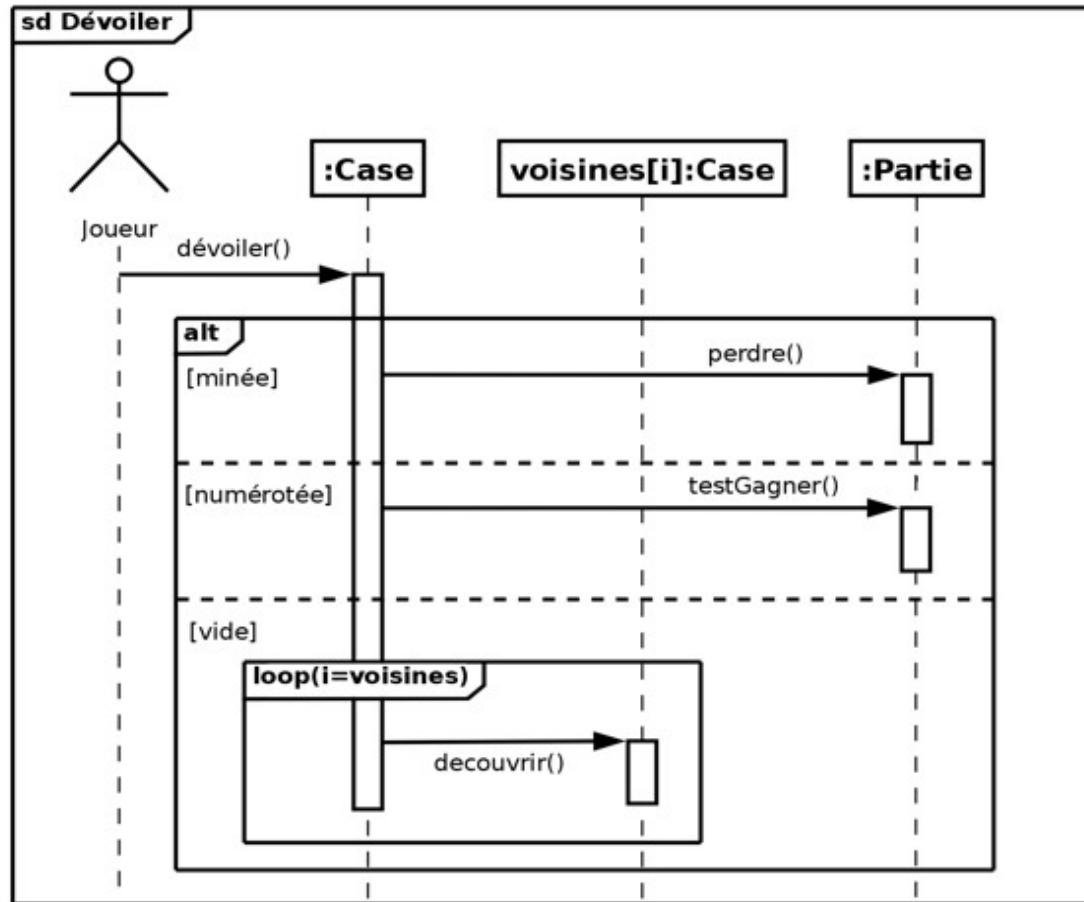


Figure 7.15: Représentation d'un choix dans un diagramme de séquence illustrant le découverrement d'une case au jeu du démineur.

Diagramme de séquence : opérateur *loop*

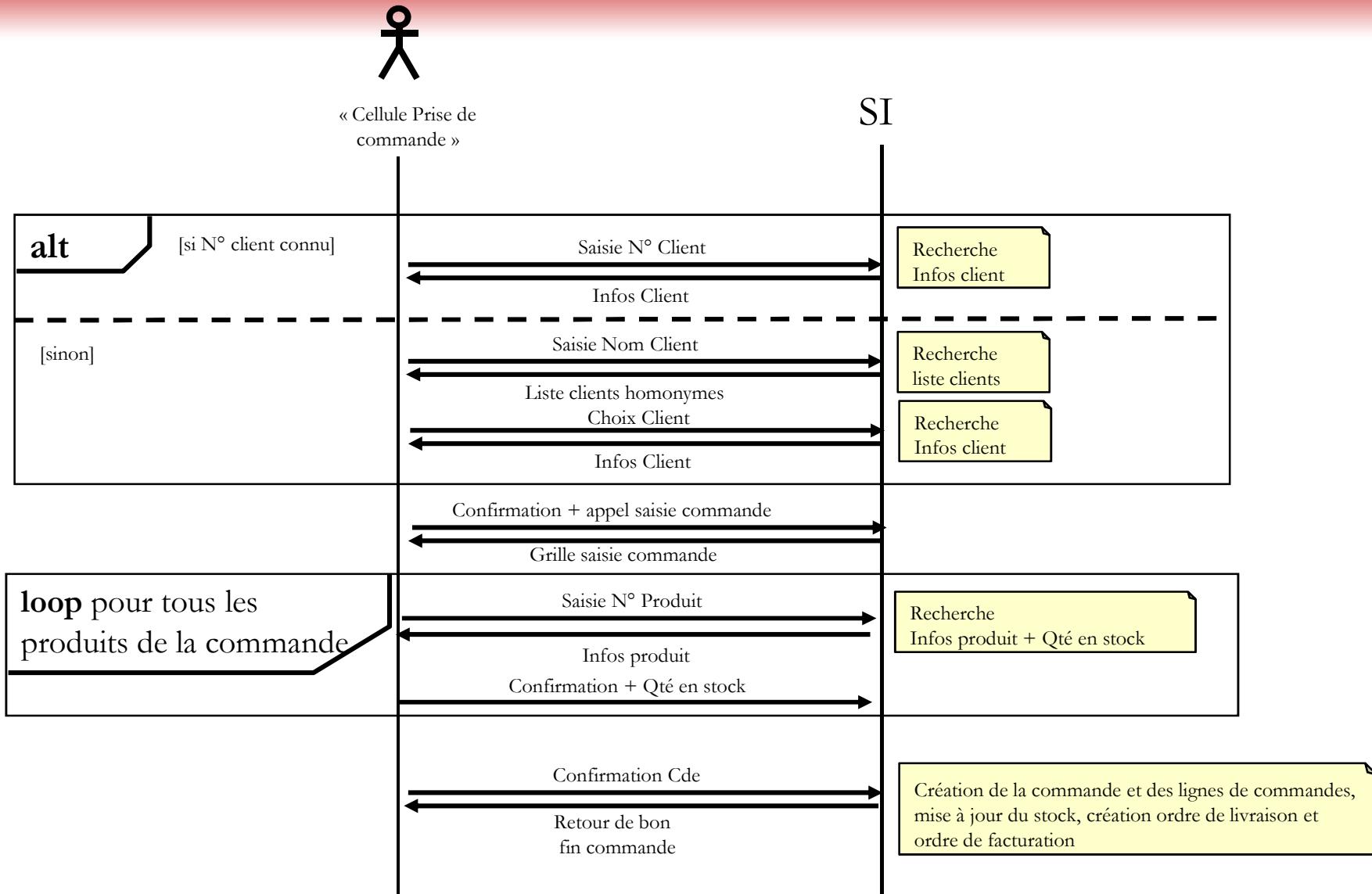


Diagramme de séquence : opérateur *opt*

opt :

La sous-séquence ne s'exécute que si la condition associée est satisfaite

Diagramme de séquence : opérateur *opt*

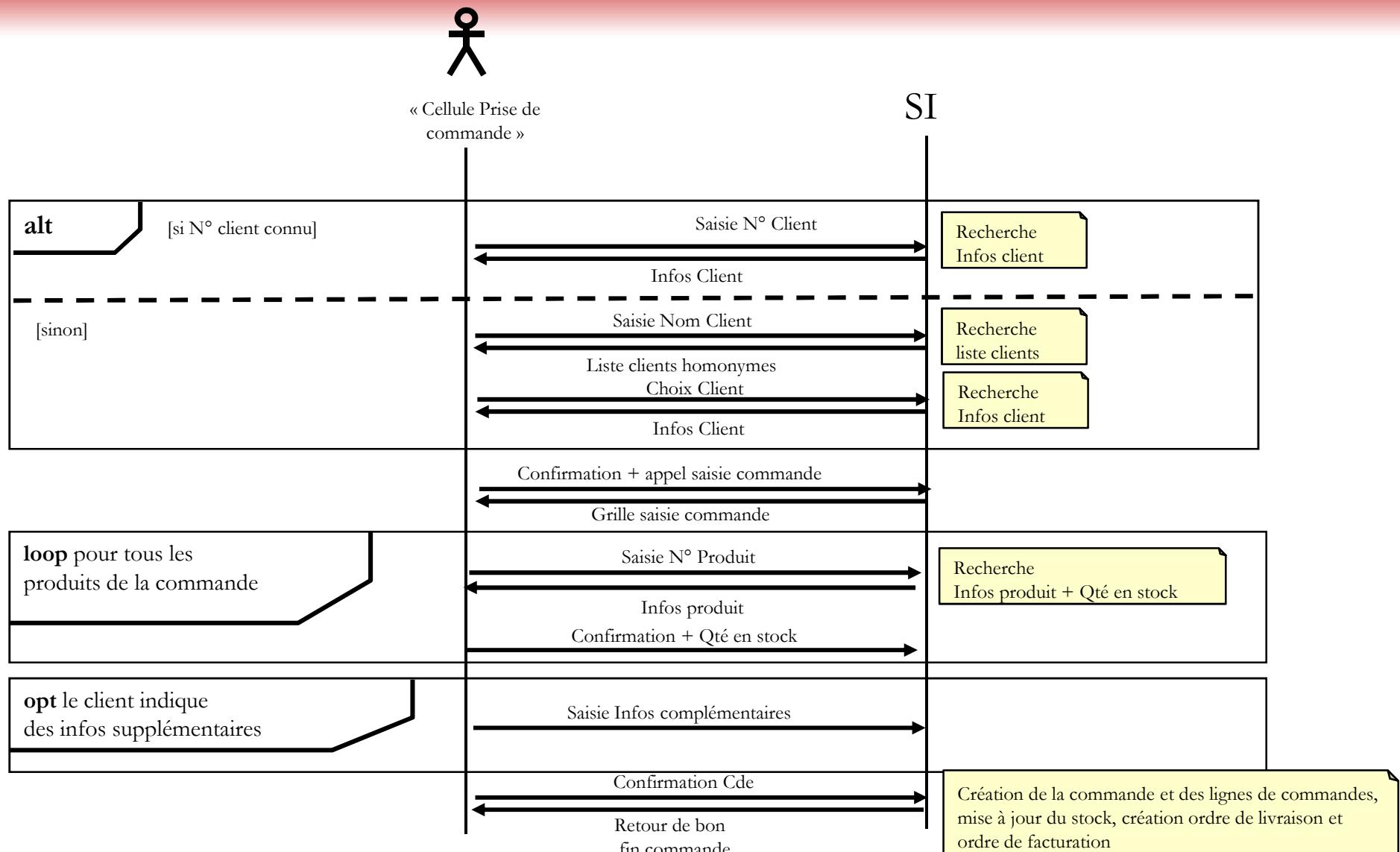


Diagramme de séquence : opérateur *par*

par :

permet de représenter
des sous-séquences se
déroulant en parallèle

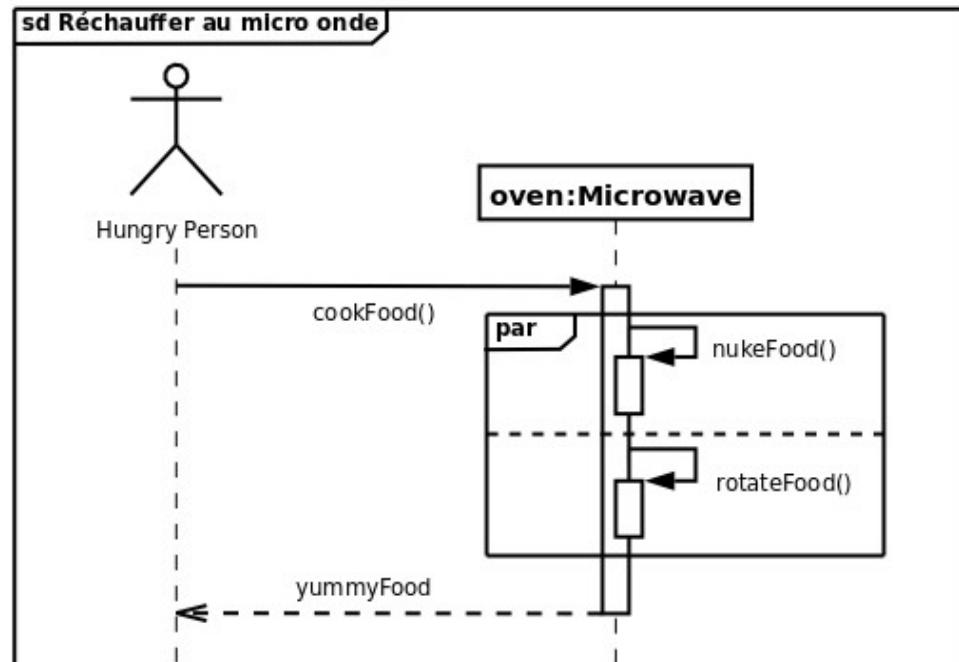


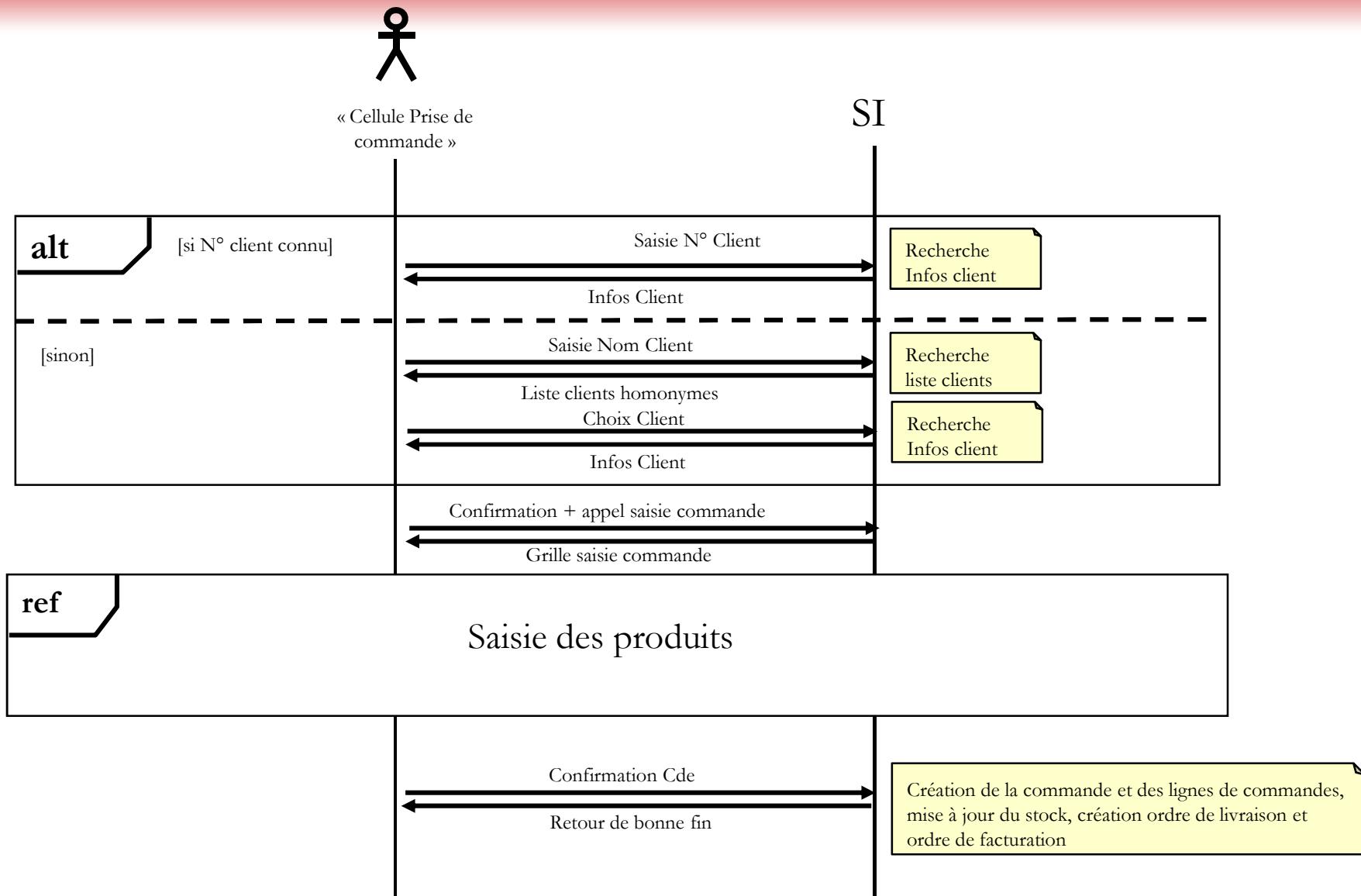
Figure 7.16: *Microwave* est un exemple d'objet effectuant deux tâches en parallèle.

Diagramme de séquence : opérateur *ref*

ref :

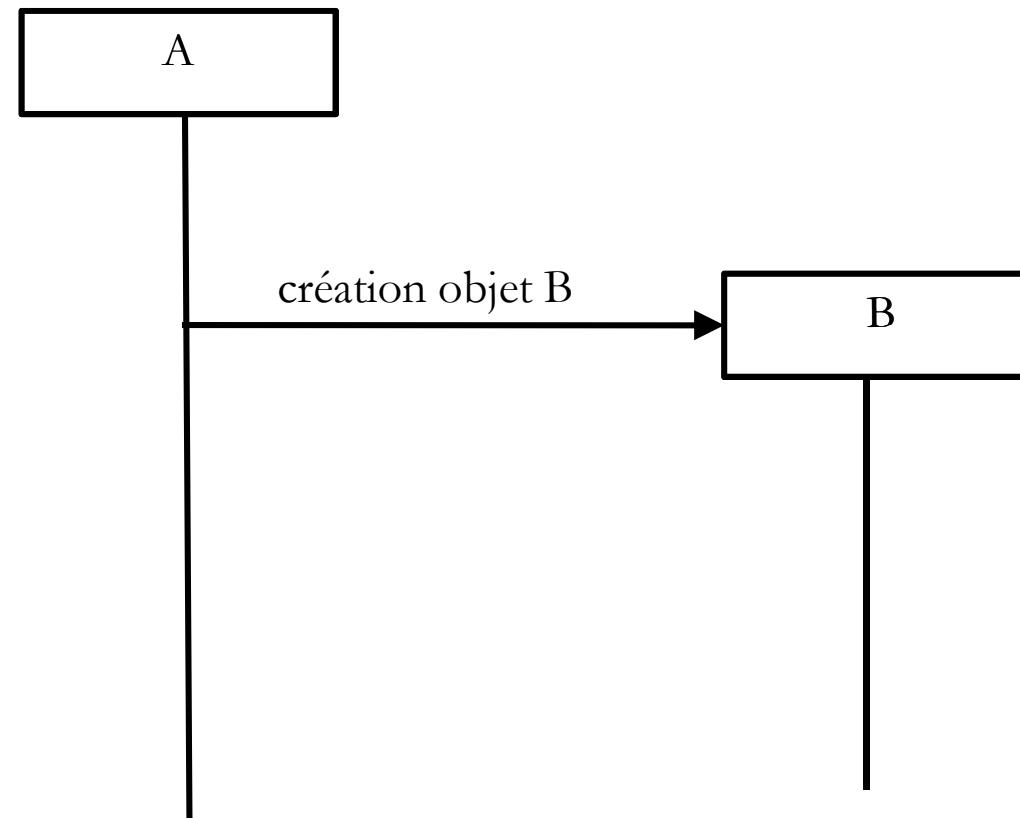
référence à une séquence définie par ailleurs

Diagramme de séquence : opérateur *ref*

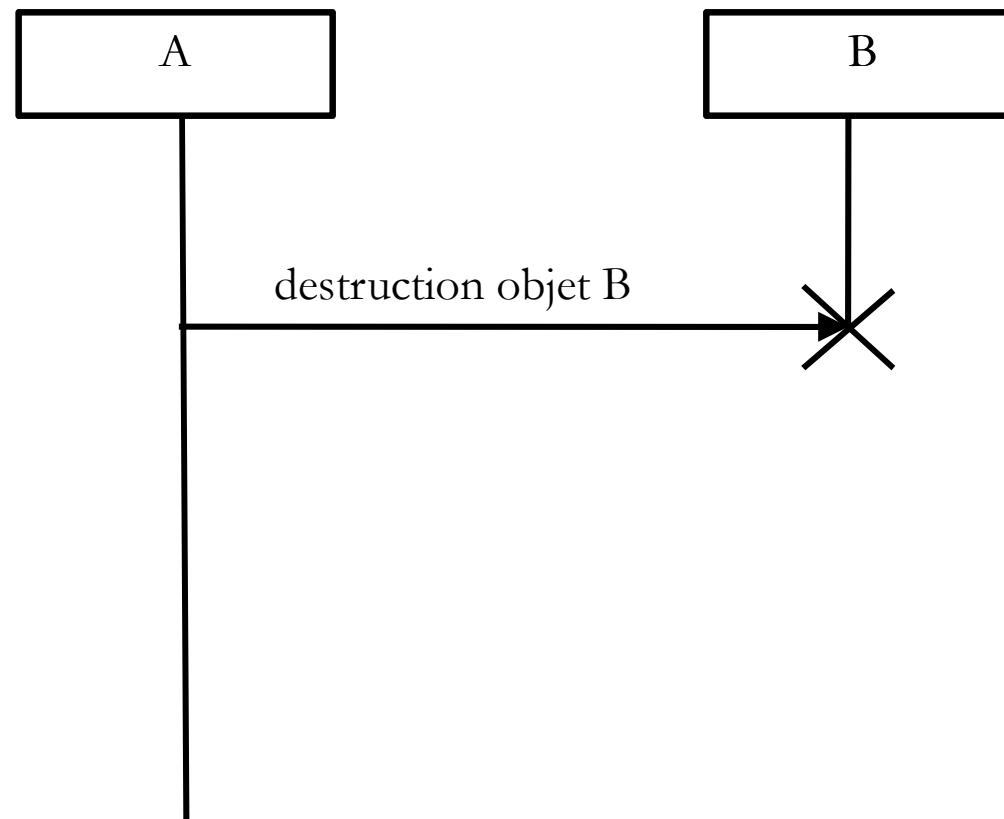


Opérations particulières

Création d'objet



Destruction d'objet



Opérations particulières

Ajout de contrainte temporelle

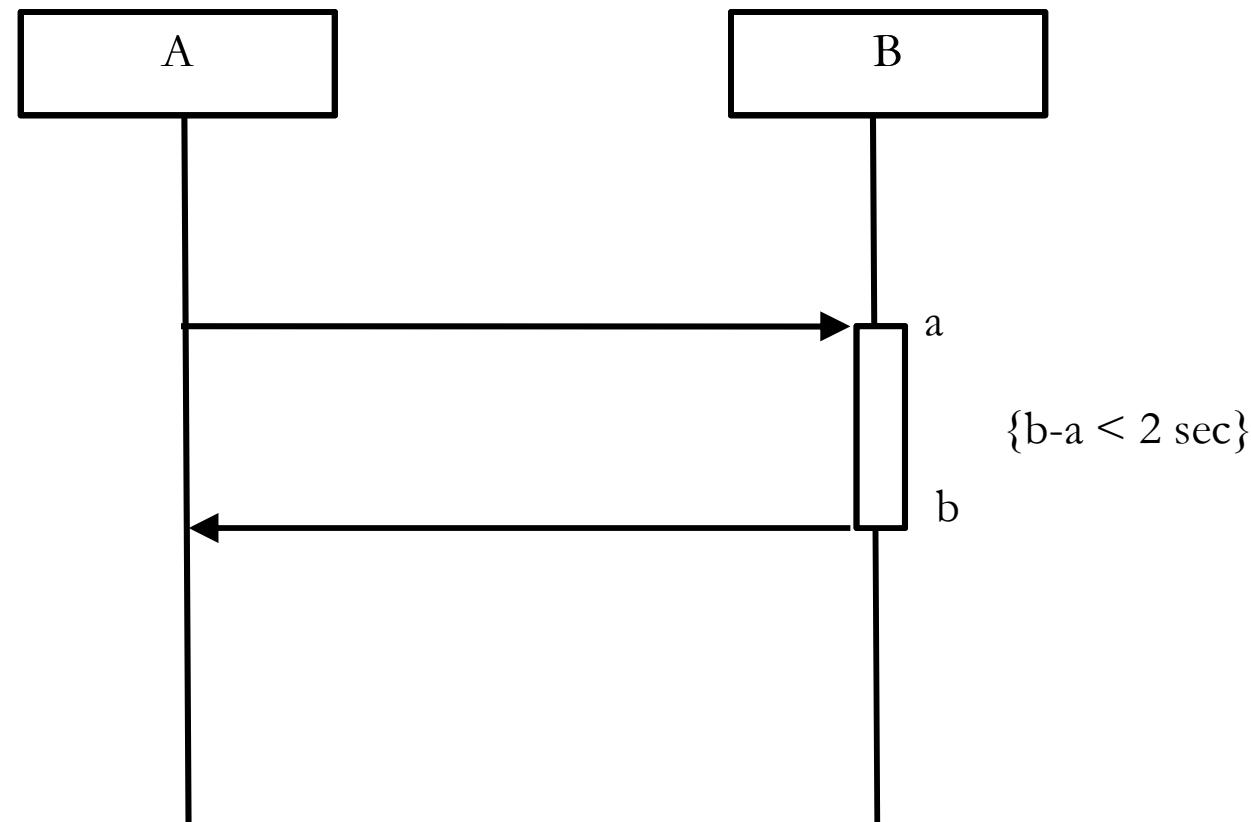
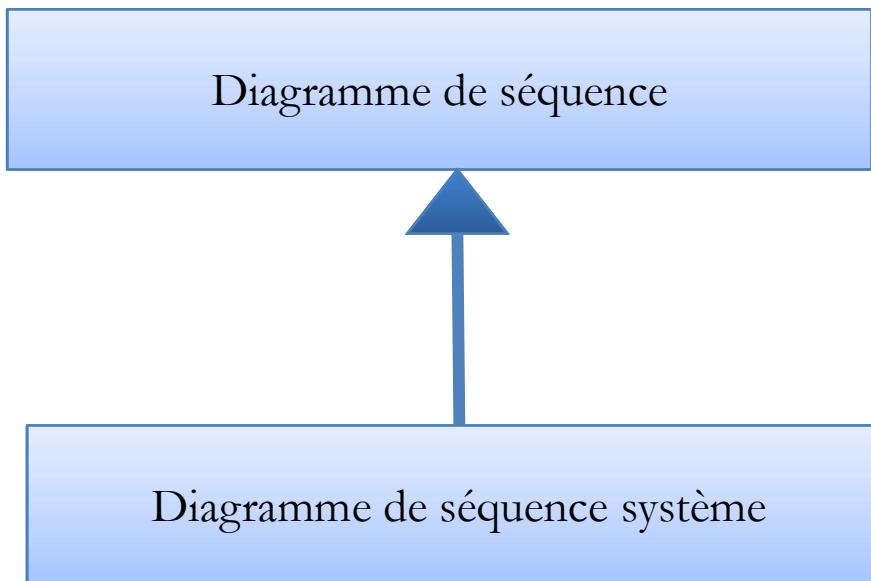


Diagramme de séquence système



Permet de décrire un CU en reprenant un scenario textuel sous forme graphique.

Diagramme de séquence système pour décrire un CU

Diagramme de séquence système

Description du CU
retirer de l'argent

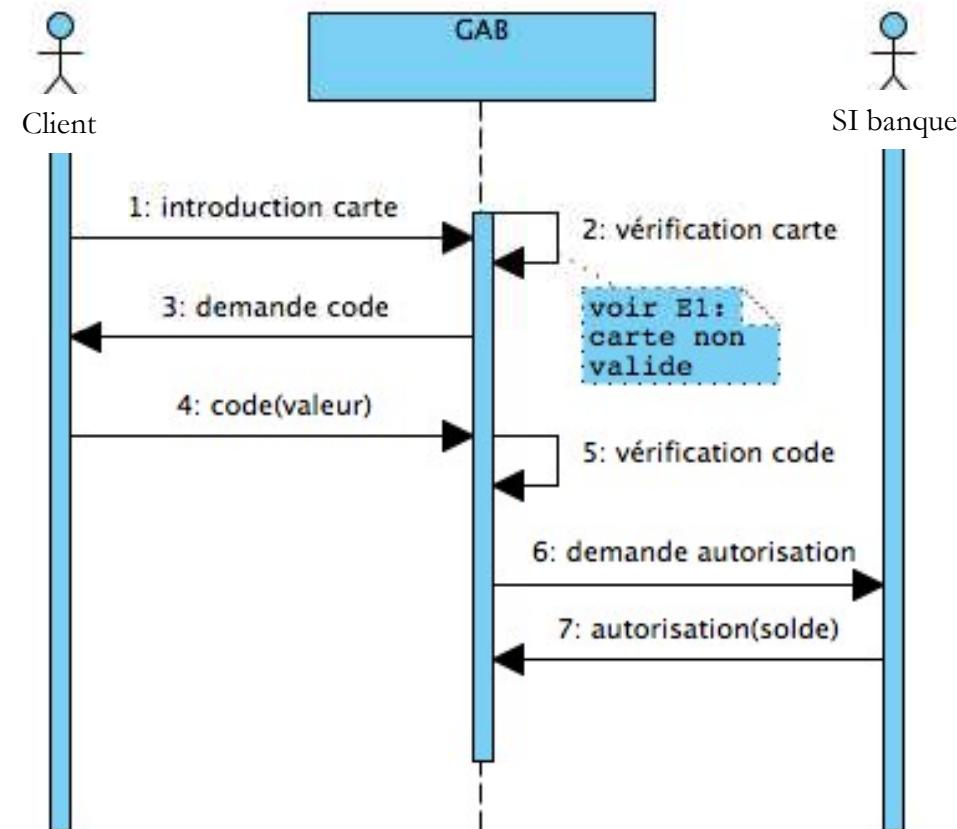


Diagramme de séquence système pour décrire un CU

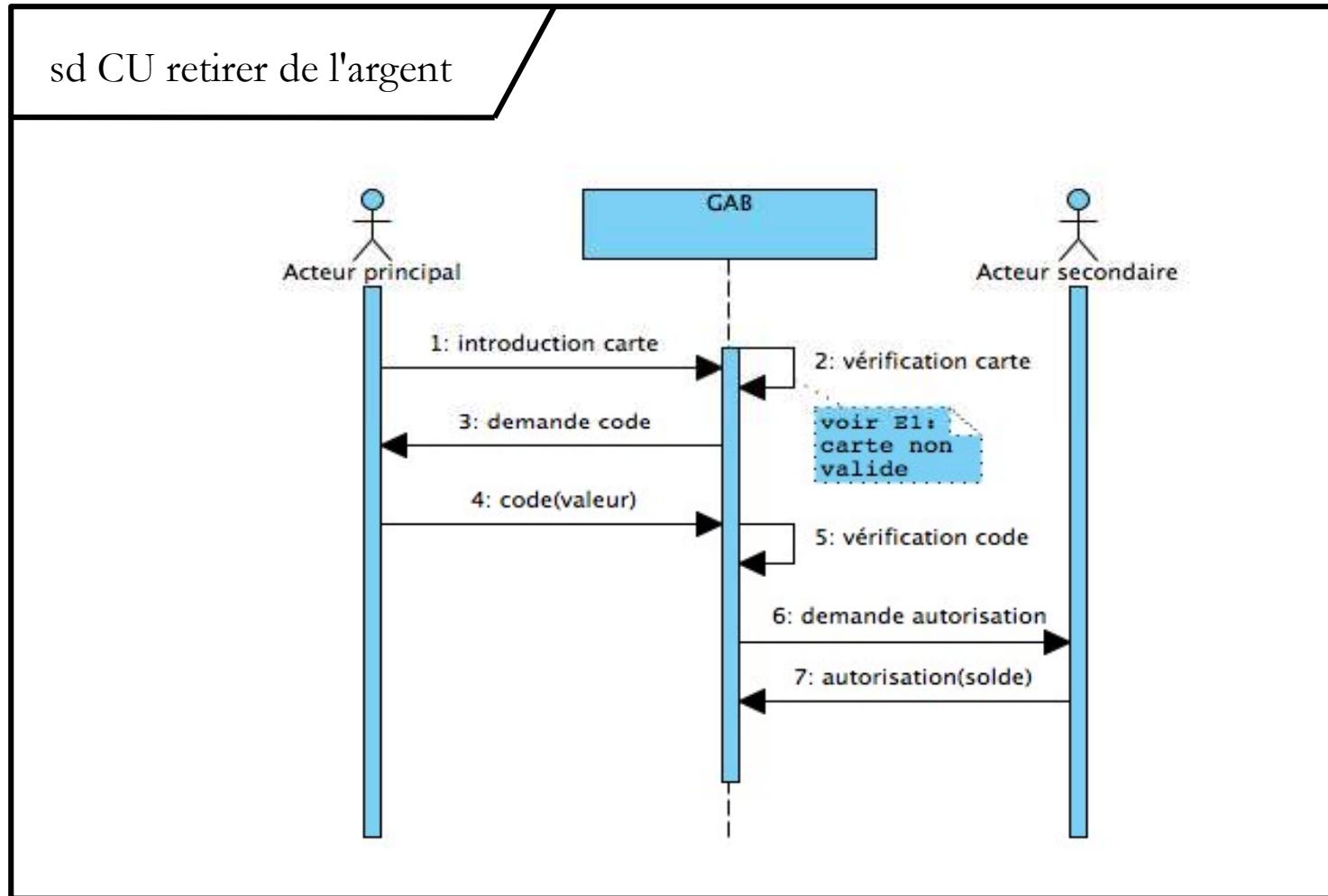


diagramme de communication
(Anciennement appelé diagramme de collaboration)

COMMUNICATION DIAGRAM

Diagramme de communication

En analyse, permet de :

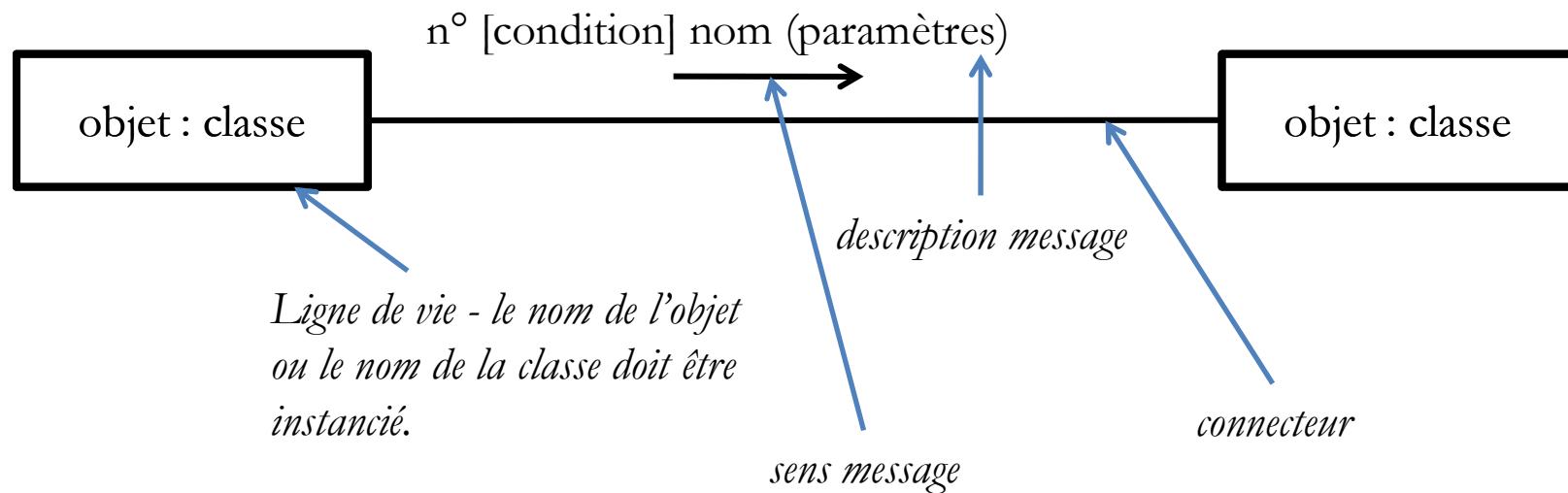
- décrire un contexte dynamique, le système étant représenté comme un objet au centre du diagramme, sans numérotation des messages,
- décrire une organisation (dynamique),
- formaliser un scenario (dynamique),
- parfois décrire un contexte statique (assez rare).

En conception, permet de :

- formaliser les enchaînements d'écrans d'une IHM (dynamique),
- décrire des mécanismes de synchronisation entre objets, l'enchaînement des appels entre les méthodes de classes, Les objets du diagramme sont alors des objets java et les messages sont des appels de méthodes (dynamique),
- ...

Diagramme de communication

- Montre les interactions (messages échangés) entre objets



ou

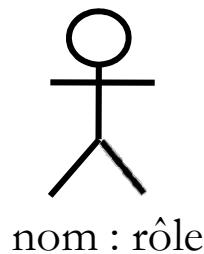


Diagramme de communication

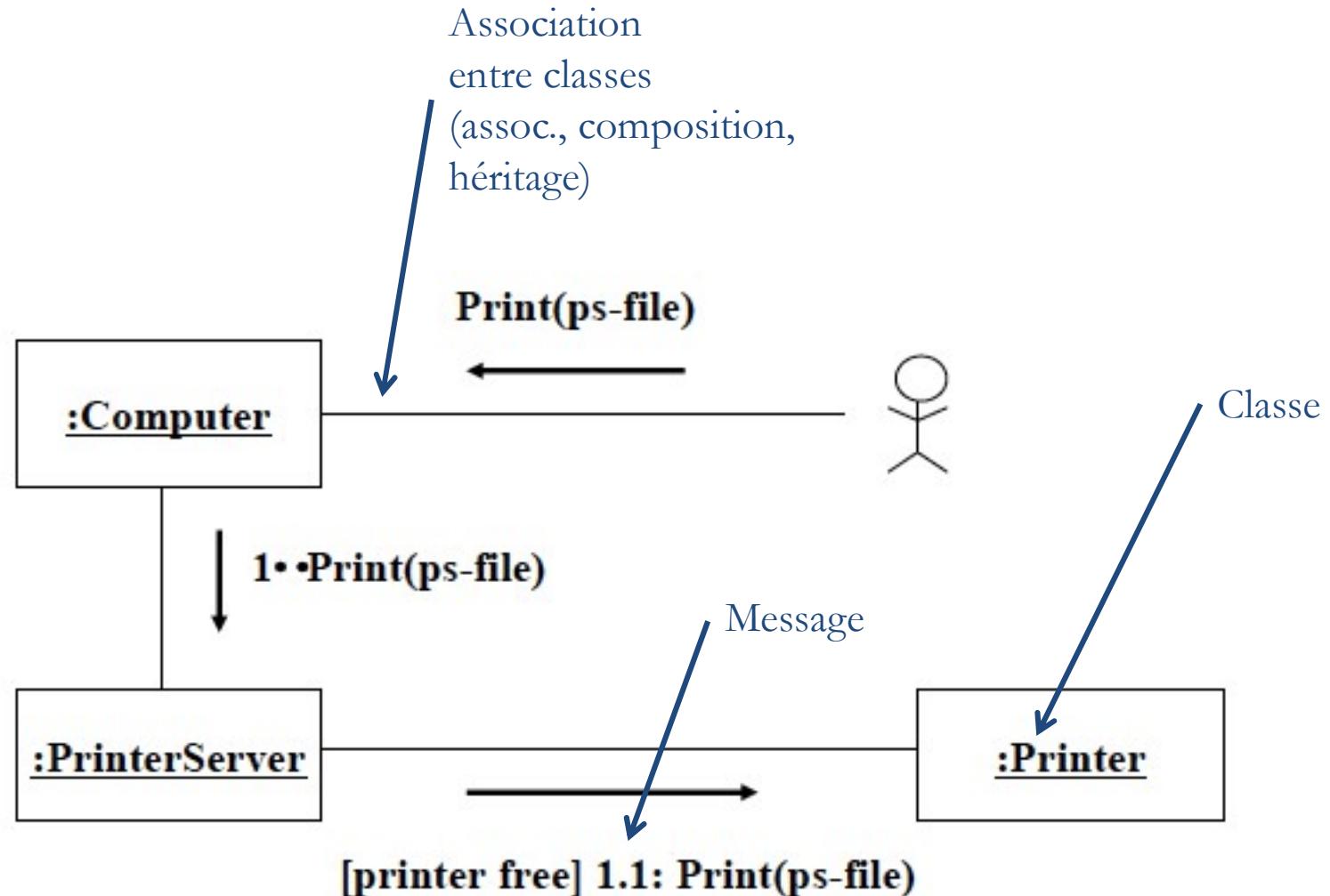


Diagramme de communication

Description d'un contexte (dynamique)

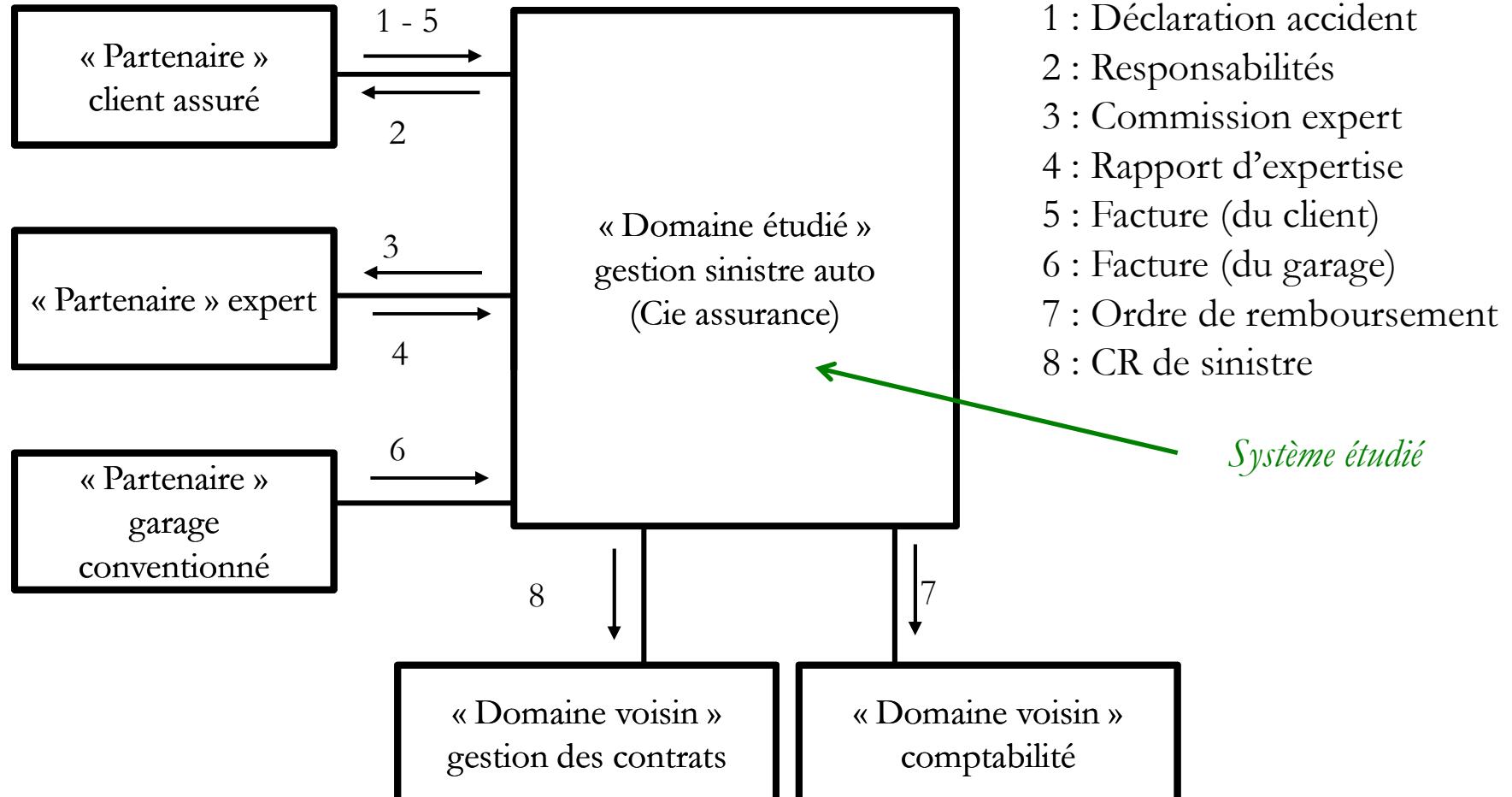
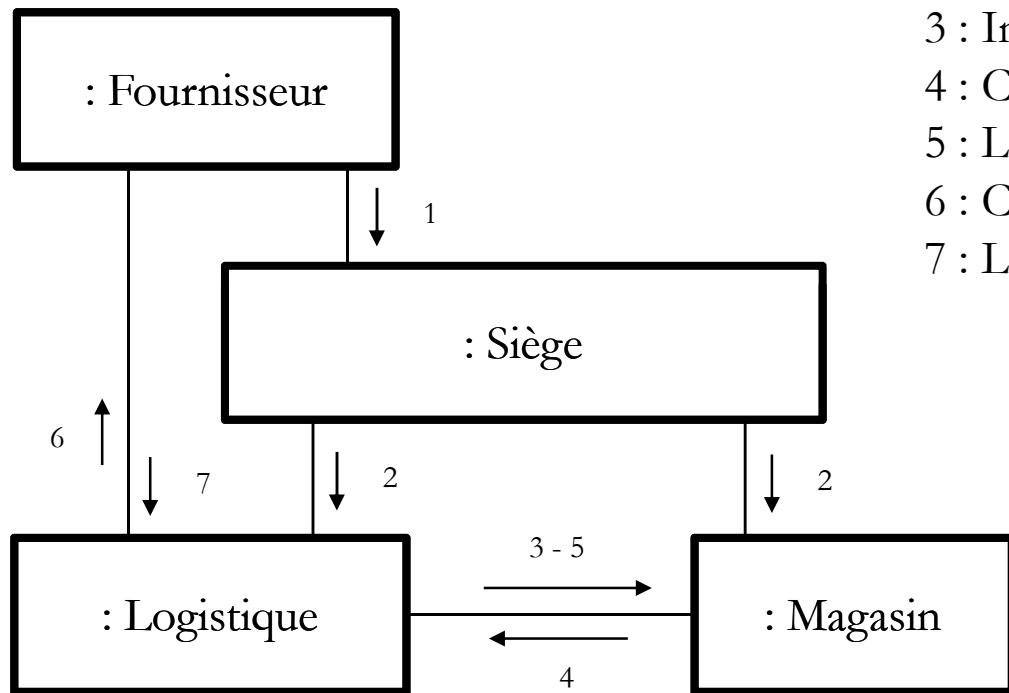


Diagramme de communication

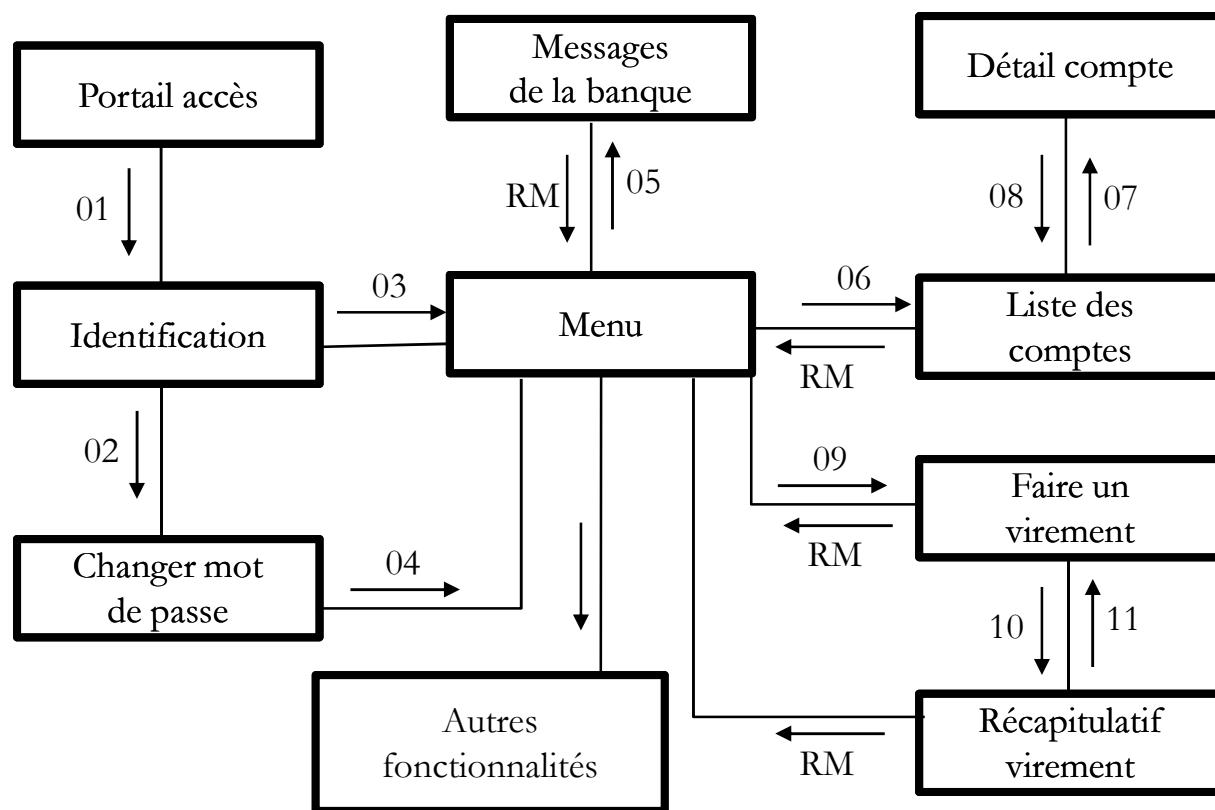
Description d'une organisation (dynamique)



- 1 : Infos nouveau produit
- 2 : Nouveau produit ou produit modifié
- 3 : Infos modalités approvisionnement
- 4 : Commande du magasin
- 5 : Livraison au magasin
- 6 : Commande globale
- 7 : Livraison globale

Diagramme de communication

Description des enchaînements d'écrans d'une IHM (dynq.)



- 01 : accès identification
02 : Identification OK + Mot de passe périmé
03 : Identification OK
04 : nouveau mot de passe OK
05 : accès aux messages de la banque
06 : accès liste des comptes
07 : accès détail d'un compte
08 : retour liste des comptes
09 : faire un virement
10 : virement confirmé
11 : autre virement

RM : retour menu

Diagramme de communication

Description d'un contexte (statique)

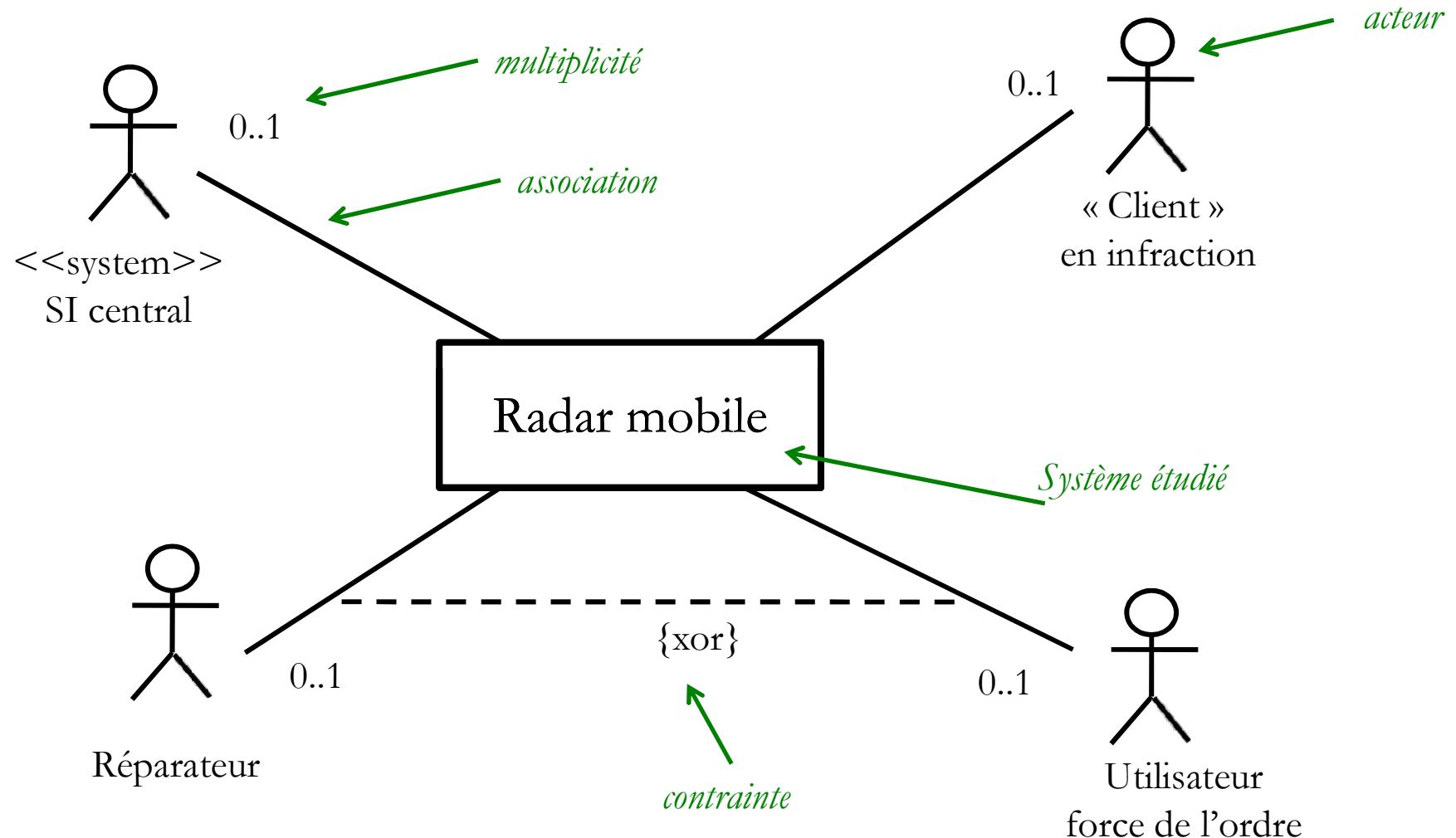


Diagramme de communication

Description d'un scenario (dynamique)

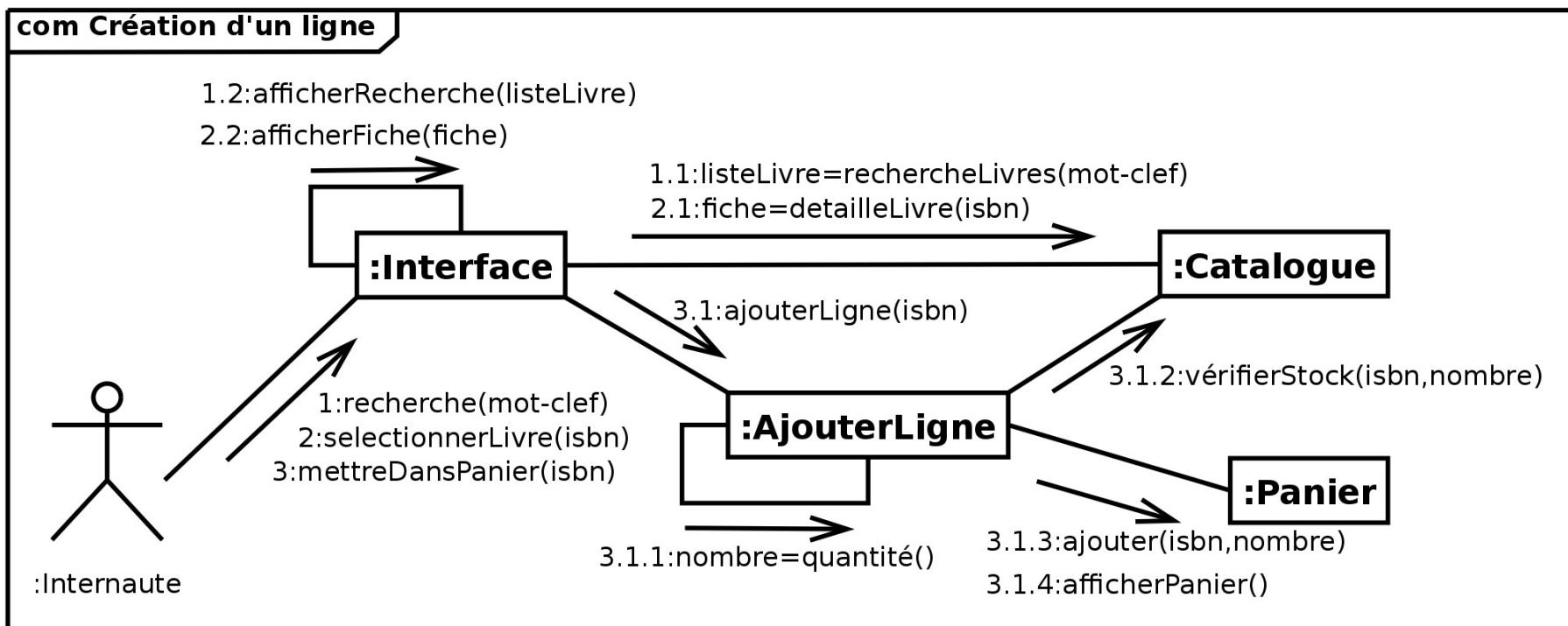


Diagramme de communication

L'information d'un diagramme de communication pour modéliser un scenario peut souvent être représentée à l'aide d'un diagramme de séquence.

De par leurs « formes » :

- Le diagramme de séquence met l'accent sur l'ordre des interactions,
- Le diagramme de communication met l'accent sur le contexte et sur la disposition physique des composants d'un système

diagramme d'états-transitions

(Diagramme d'états)

STATE MACHINE DIAGRAM

Diagramme d'états-transitions

- Automate d'états fini.
- **État** : Situation d'un objet que l'on désire connaître et gérer. L'état d'un objet dépend de la valeur de ses attributs et de la présence (ou l'absence) de liens avec d'autres objets au sein des associations
- **Transition** : Passage d'un objet d'un état à un autre. Elle est déclenchée par un événement
- **Événement** : Stimulus qui provoque une (ou plusieurs) transition(s)
- Point de vue dynamique. Appréhender le cycle de vie (si celui-ci est complexe) d'un objet.



Diagramme d'états-transitions

Etat transition : interrupteur

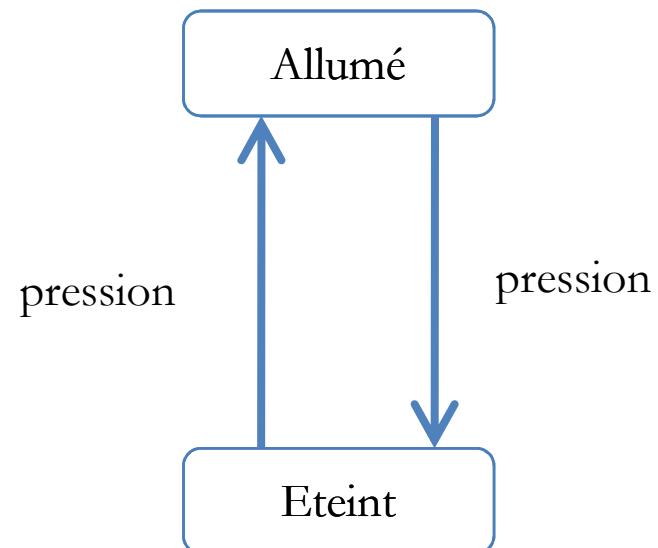


Diagramme d'états-transitions

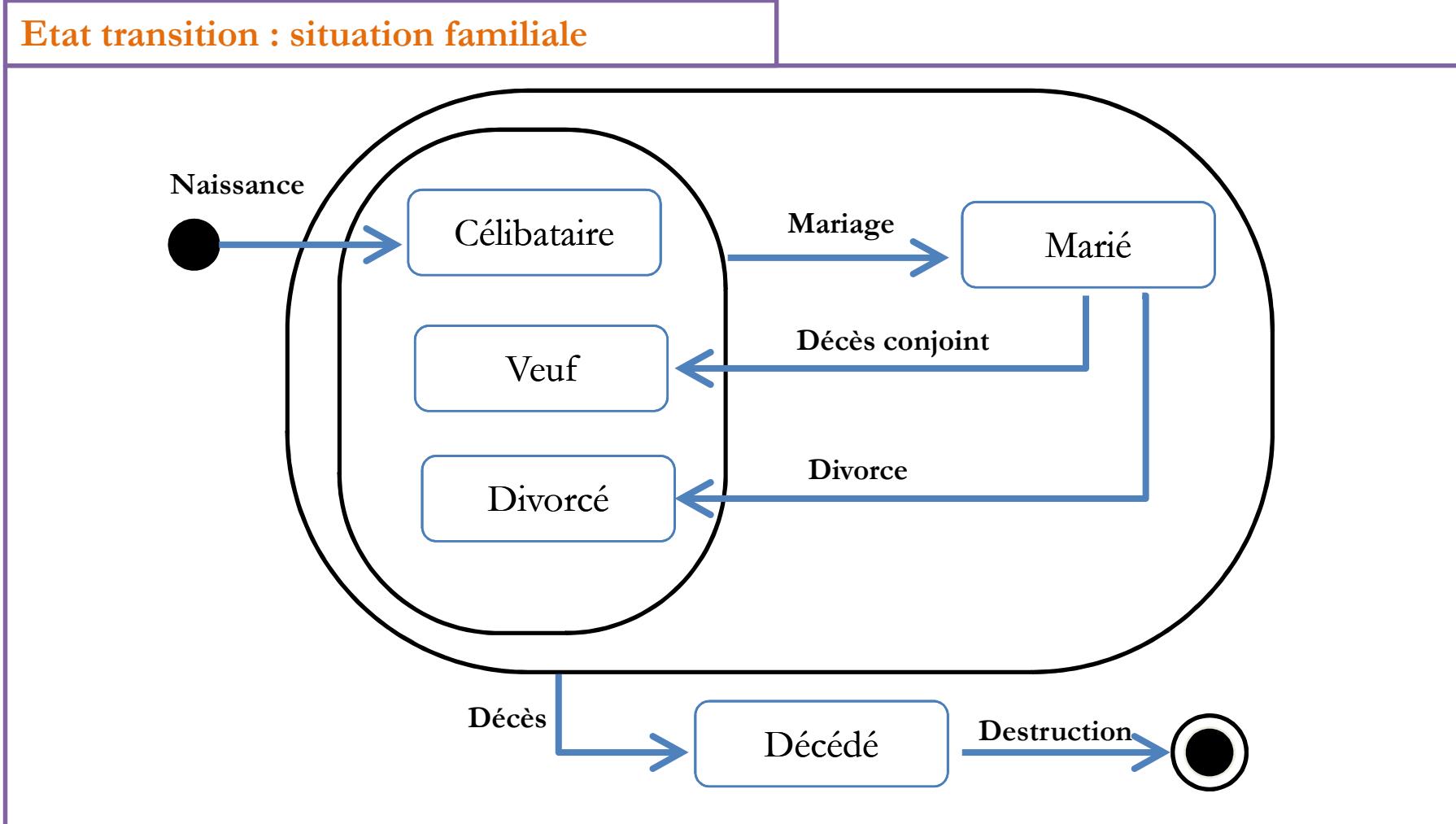


Diagramme d'états-transitions

L'événement :

- Il n'est pas local à une classe.

Un même évènement peut provoquer différentes transitions sur des objets de différentes classes

- Il peut avoir différentes origines :

- Externe (ex : arrivée d'un règlement client)
- Interne (ex : le seuil de 10 commandes est atteint)
- Temporel (ex : le délai de réflexion de 7 jours est terminé)
- Décisionnel (ex : prolongation de la période d'essai)

Le diagramme d'états-transitions

- **Aide à déterminer les opérations**

Une classe doit posséder toutes les opérations qui lui permettront d'assurer toutes les transitions dans tous les cas de figure recensés sur le diagramme.

- **Aide à concevoir les IHM**

L'ensemble des IHM doit permettre à l'utilisateur d'effectuer toutes les transitions pour lesquelles il est habilité.

- **Aide à construire les scénarios de test**

Les scénarios de tests doivent couvrir toutes les transitions dans tous les cas de figure recensés sur le diagramme.

- **Aide à déterminer les états pour gérer les risques de dysfonctionnement**

Il est important que le dysfonctionnement laisse le système dans un état stable et facilement identifiable.

Diagramme d'états-transitions : Pour aller plus loin...

- Certaines actions peuvent être rattachées à un état
 - Action d'entrée : exécutée chaque fois qu'on entre dans l'état
Notation : entry / action
 - Action de sortie : exécutée chaque fois qu'on sort de l'état
Notation : exit / action
 - Action durable : exécutée tout le long de l'activation de l'état
Notation : do / action

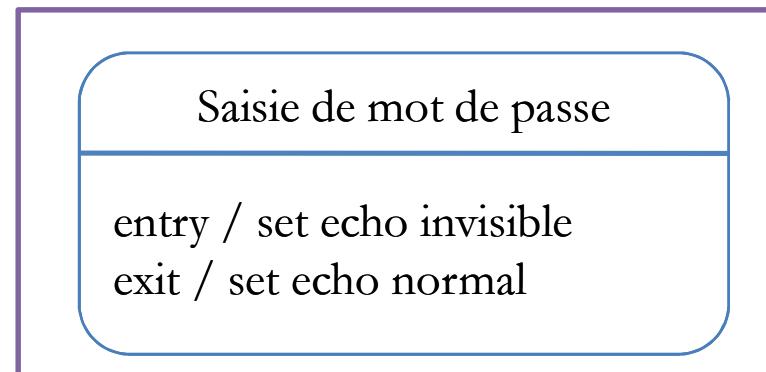
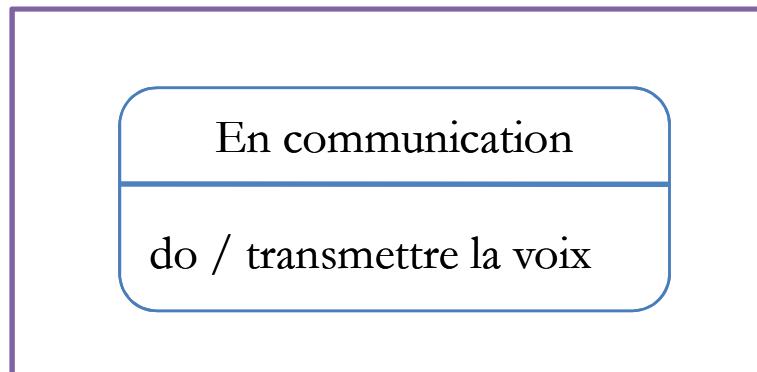


Diagramme d'états-transitions : Pour aller plus loin...

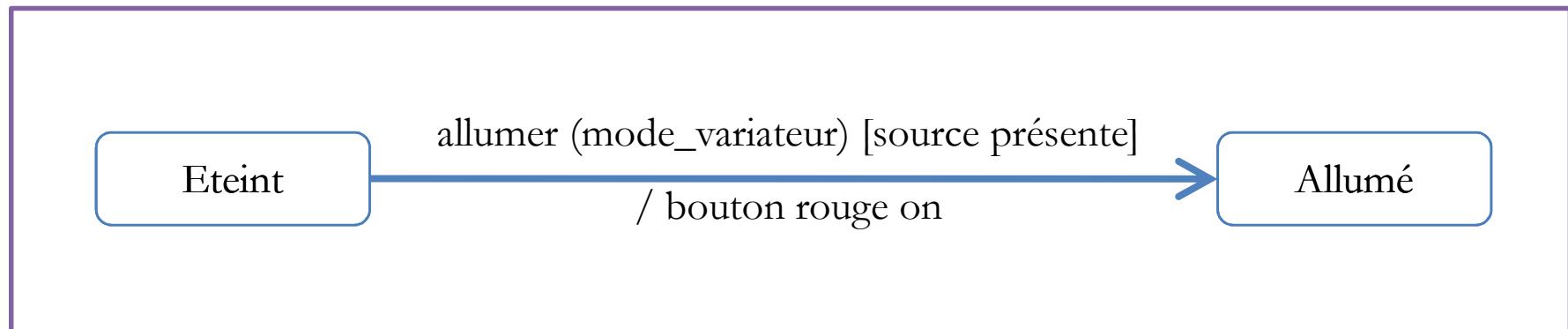
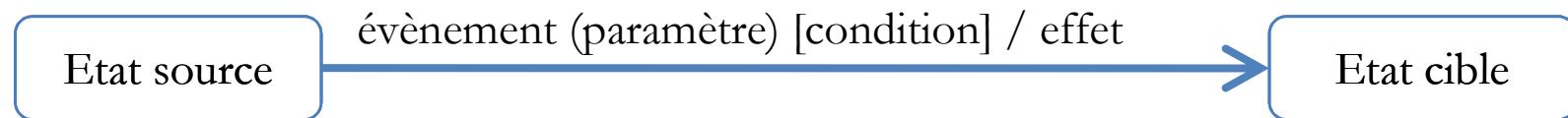


Diagramme d'états-transitions : Pour aller plus loin...

Point de jonction

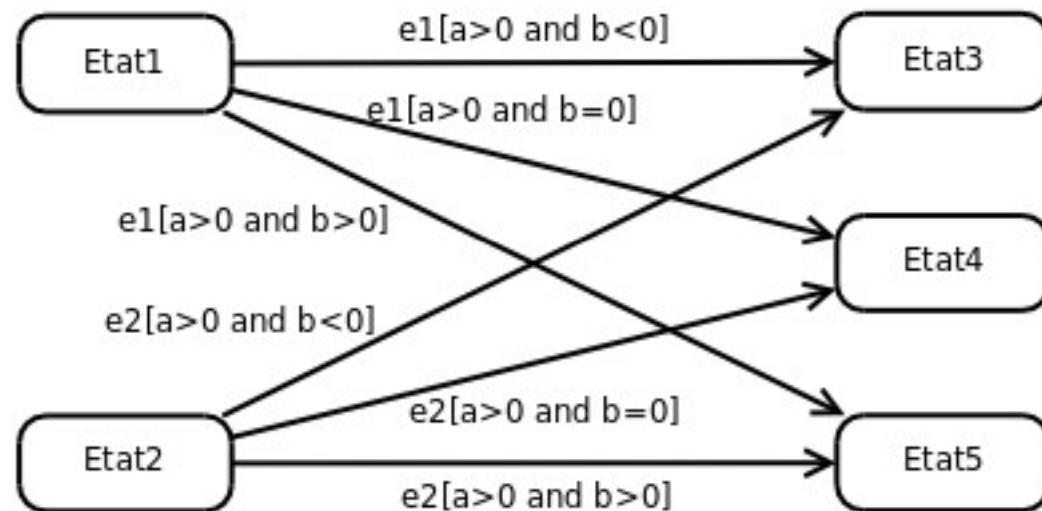
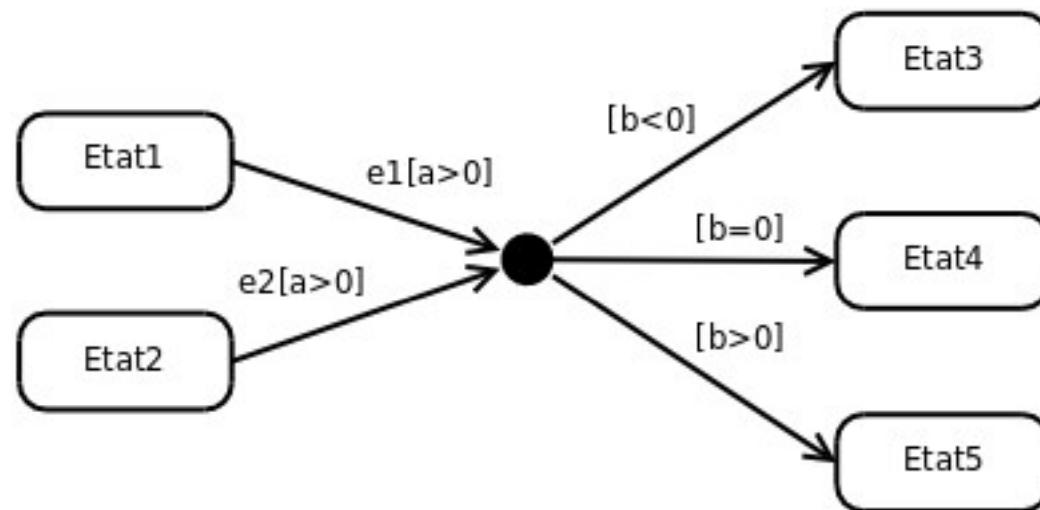


Diagramme d'états-transitions : Pour aller plus loin...

Point de jonction



*(Diagramme strictement équivalent
au précédent)*

Point de jonction = artefact graphique permettant de rendre le diagramme plus lisible

Diagramme d'états-transitions : Les points de choix

Point de jonction

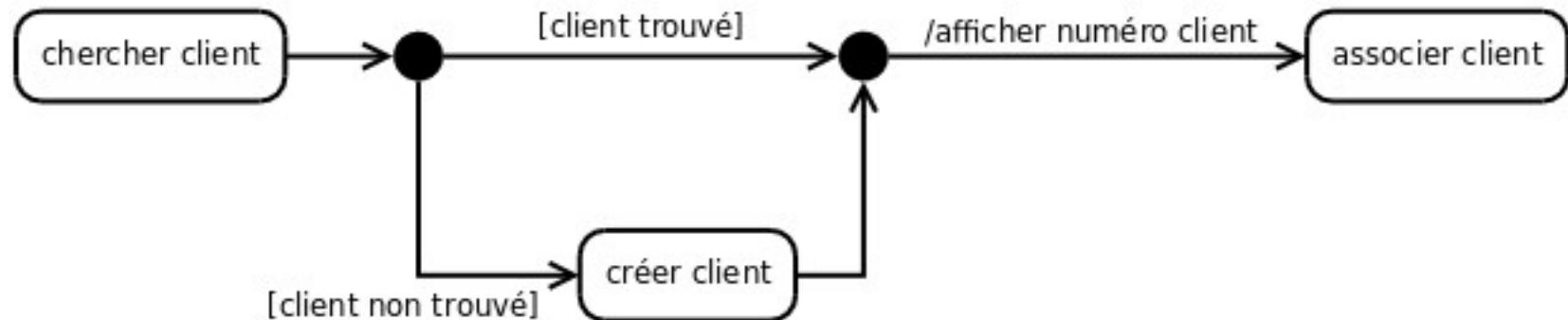
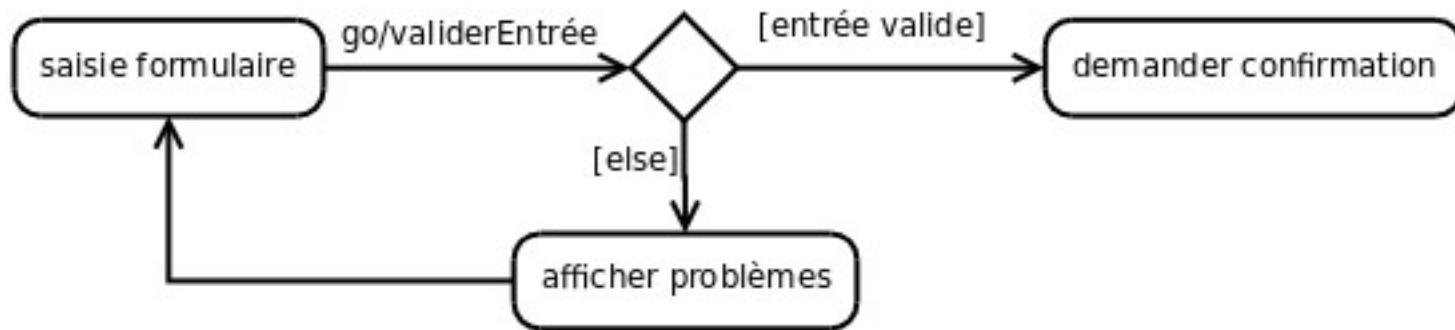


Diagramme d'états-transitions : Les points de choix

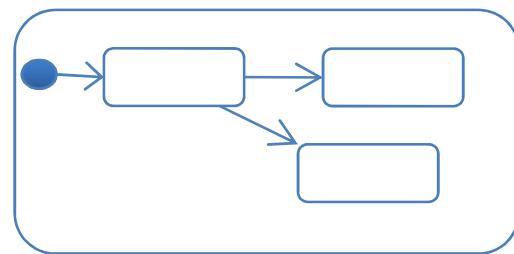
Point de décision



Comme pour les diagrammes d'activités, conditions de sortie exclusives

Diagramme d'états-transition : Etats composites

états séquentiels



états concurrents

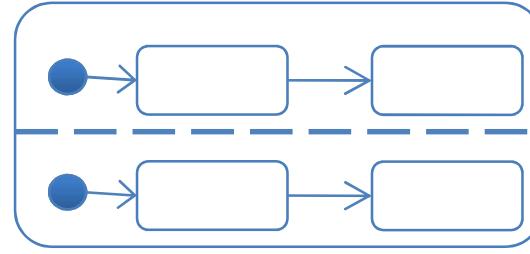


Diagramme d'états-transitions : Etat composite

Etat séquentiel

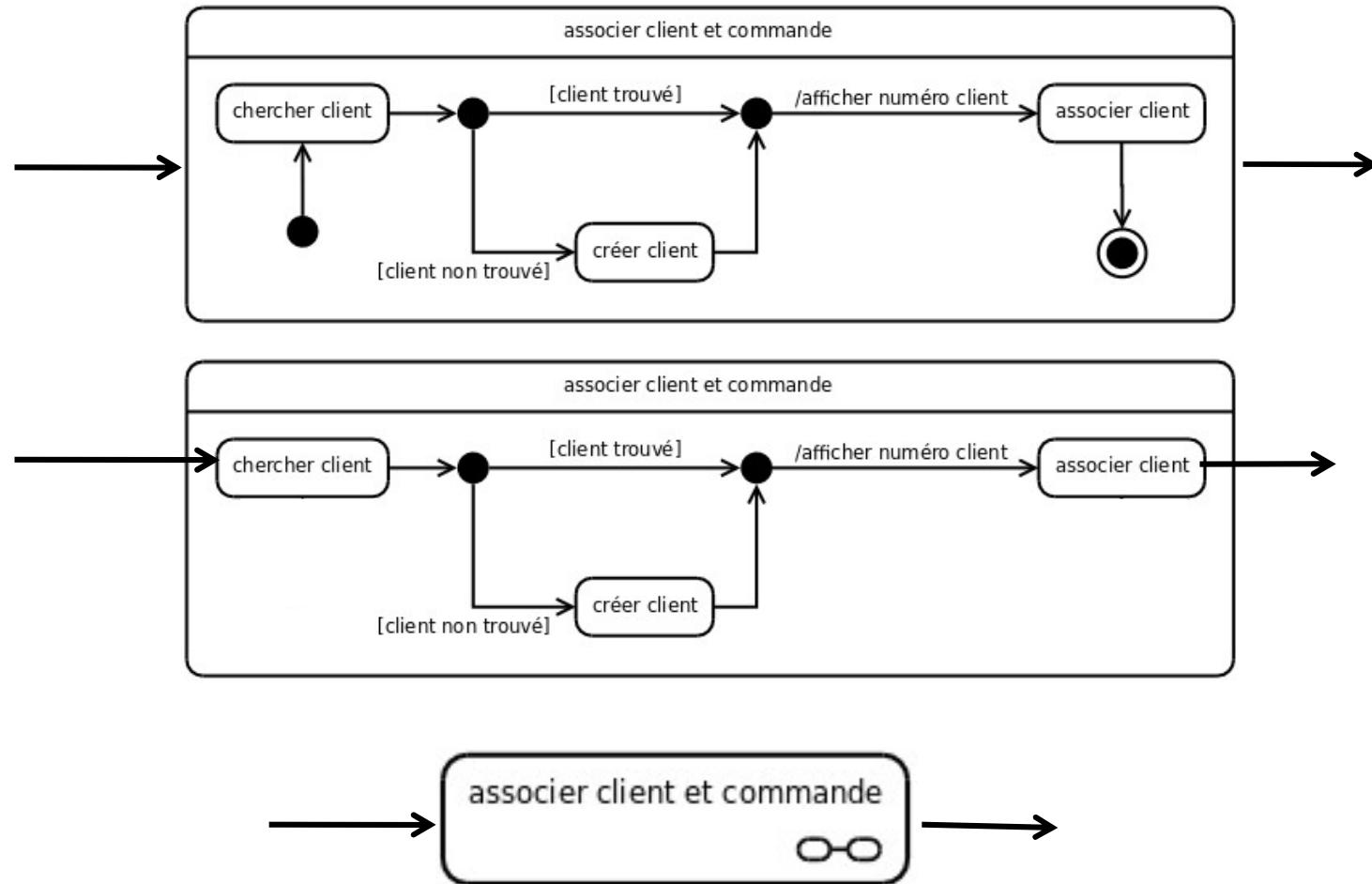
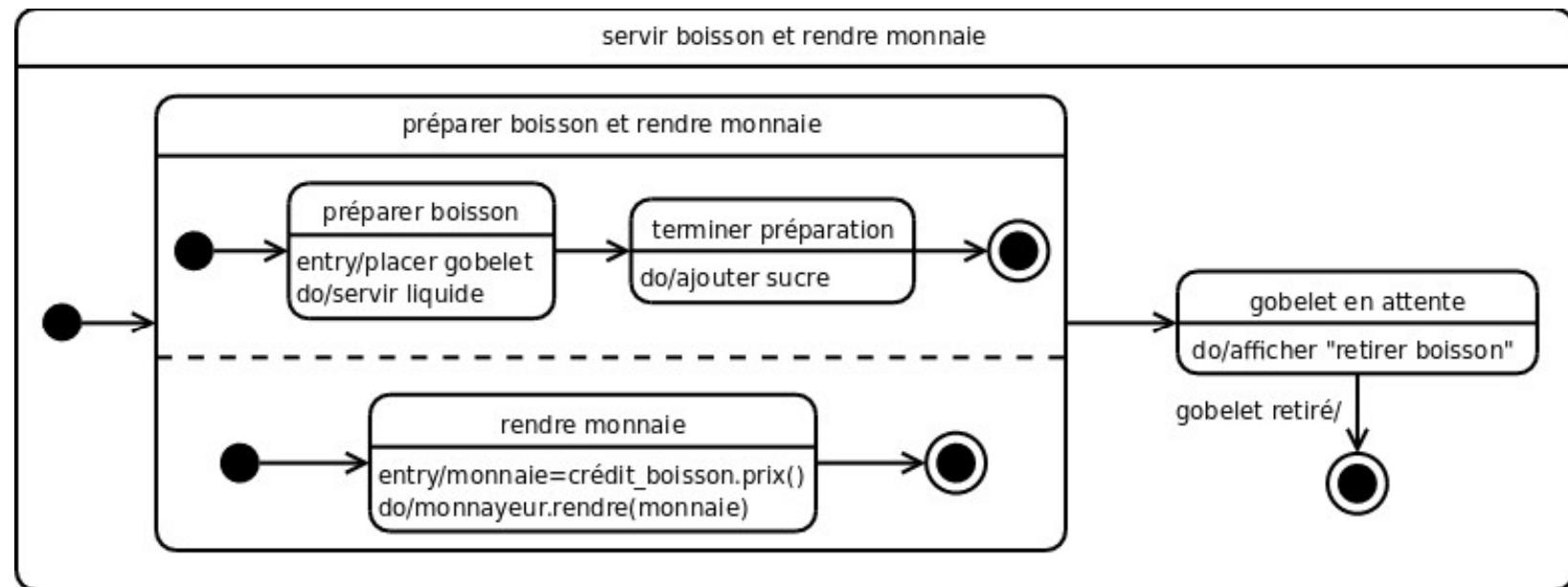


Diagramme d'états-transitions : Etat composite

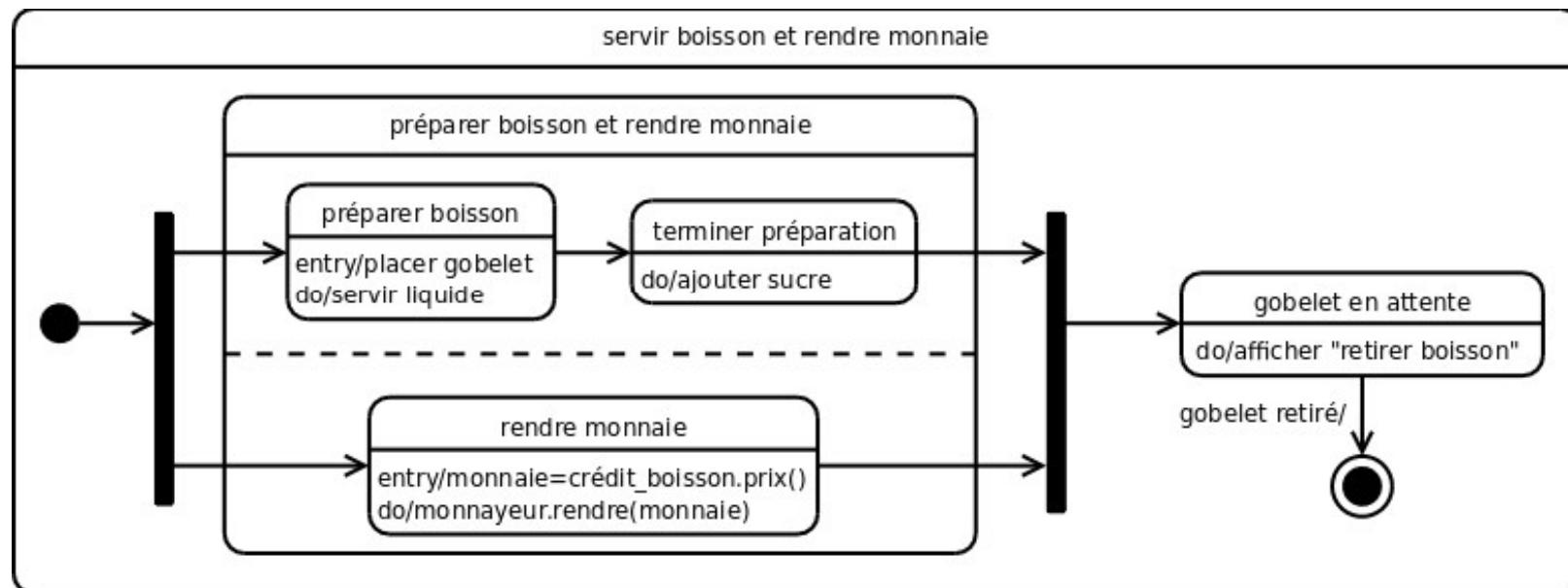
Etat concurrent



Etat composite terminé lorsque tous les états finaux des régions sont atteints
→ parallélisation

Diagramme d'états-transitions : Etat composite

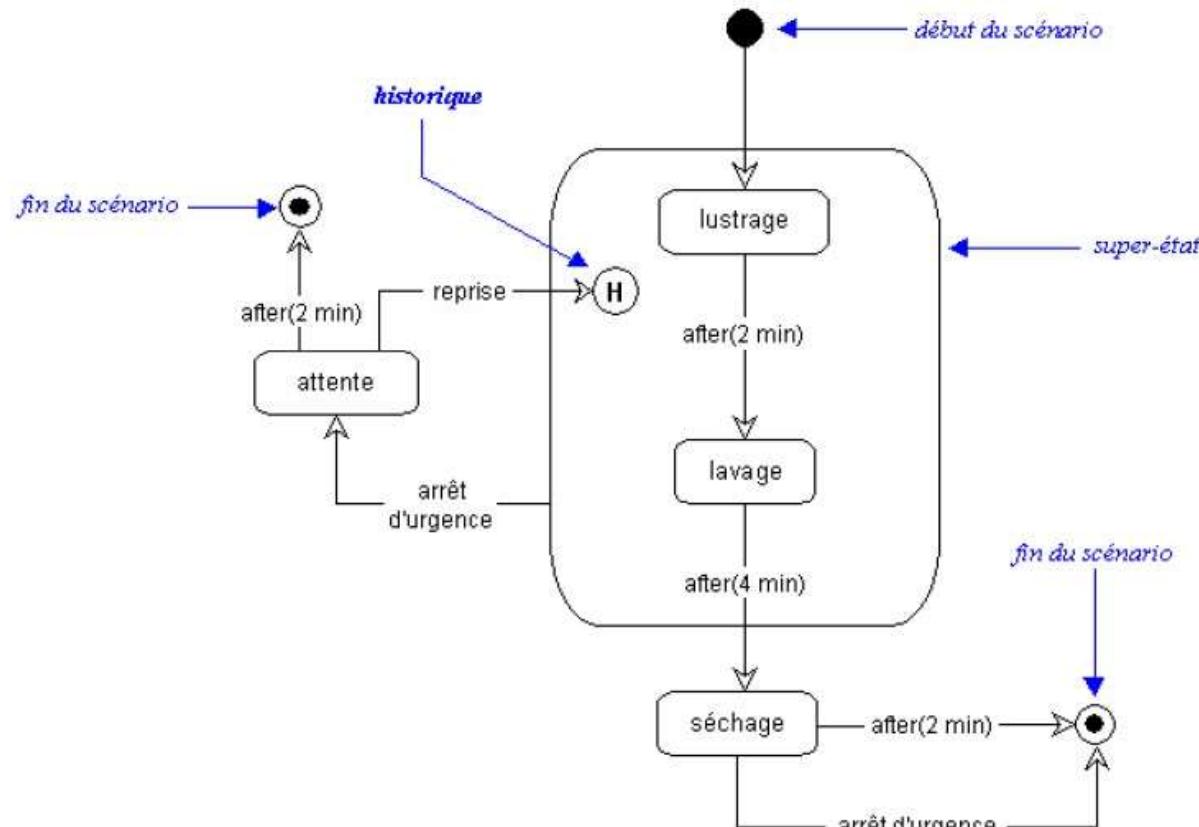
Etat concurrent



Même signification que les diagramme précédent

Diagramme d'états-transitions

Etat transition : machine à laver



Pseudo-état History :

Il permet à un super-état de se souvenir du dernier sous-état séquentiel qui était actif avant une transition sortante.

Une transition vers l'état *History* rend à nouveau actif le dernier état actif au lieu de ramener vers le sous-état initial.

diagramme d'activité

ACTIVITY DIAGRAM

Diagramme d'activité

Permet de modéliser une activité. Particulièrement utilisé pour modéliser

- un processus métier,
- l'enchaînement d'activités liées à un cas d'utilisation,
- une règle métier.

Diagramme d'activité = nœuds (actions) reliés par des transitions.

Diagramme d'activité

Trois types de nœuds : activité, objet, contrôle

Nœuds d'activité :

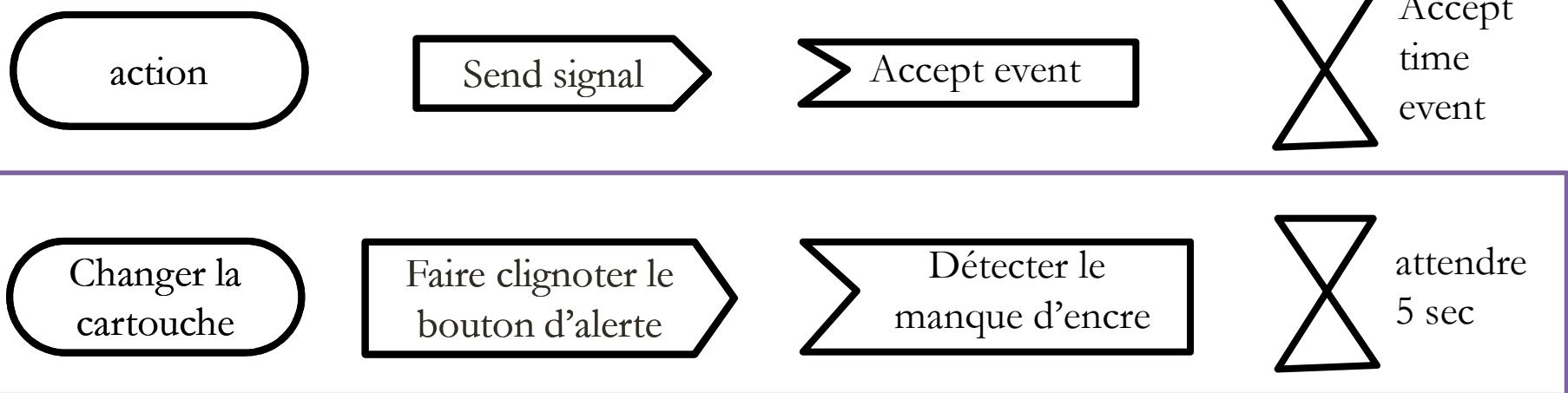


Diagramme d'activité

Nœud d'objet :



Nœuds de contrôle :

initial



final

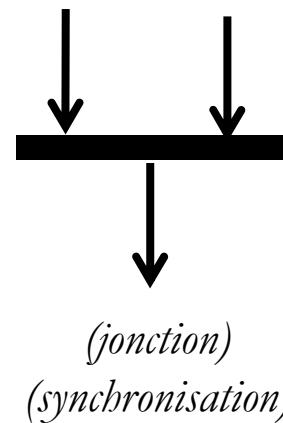
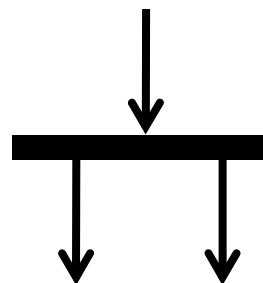
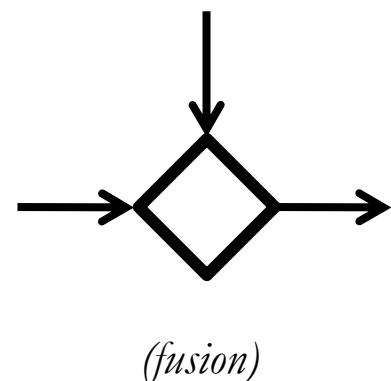
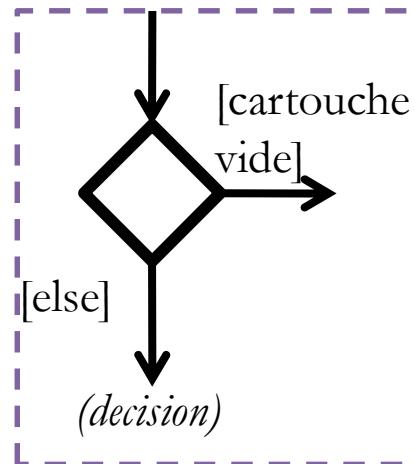
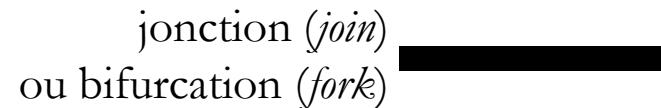
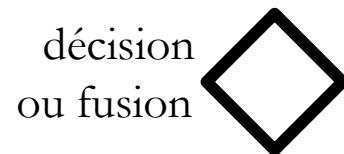


Diagramme d'activité

DA : consulter son solde bancaire

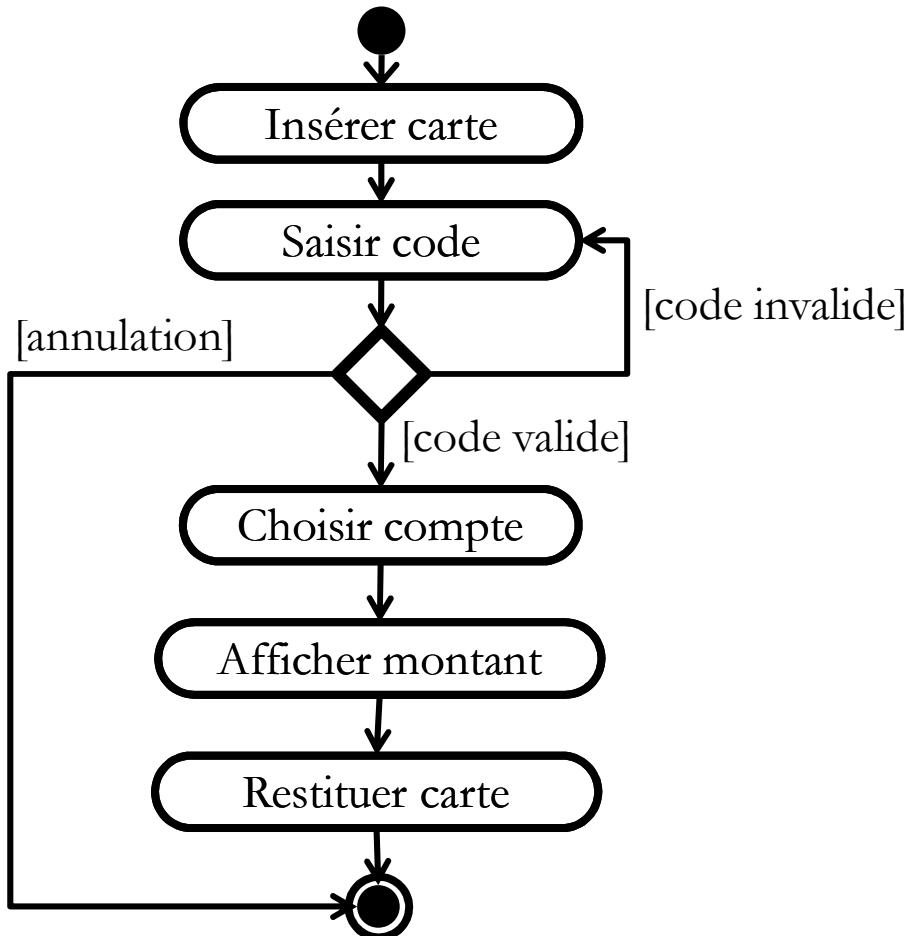


Diagramme d'activité

DA : consulter son solde bancaire

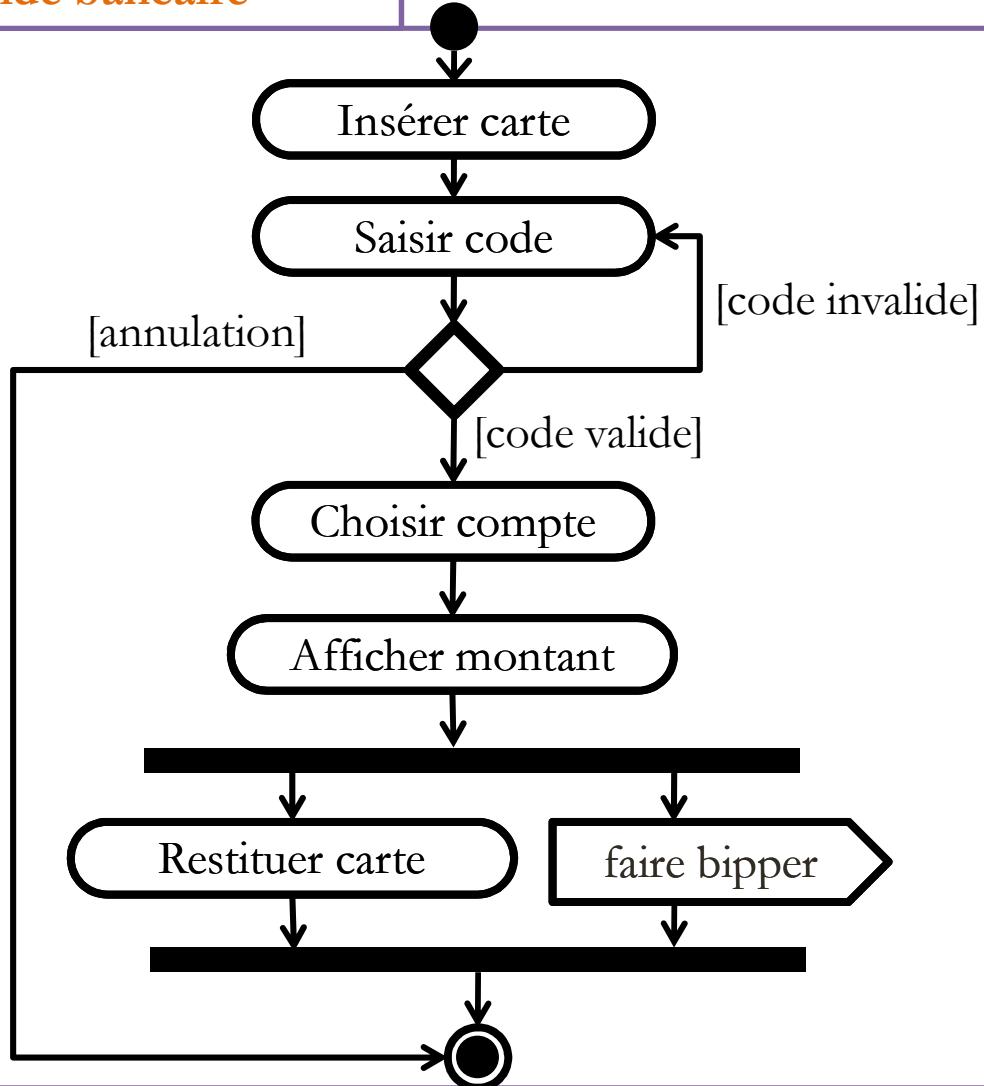


Diagramme d'activité : Condition de garde

Les conditions de garde

Une garde est une condition booléenne posée sur une transition.

Elle est évaluée lorsque l'événement se produit. Si elle est vraie (resp. fausse), la transition (ne) peut avoir lieu.

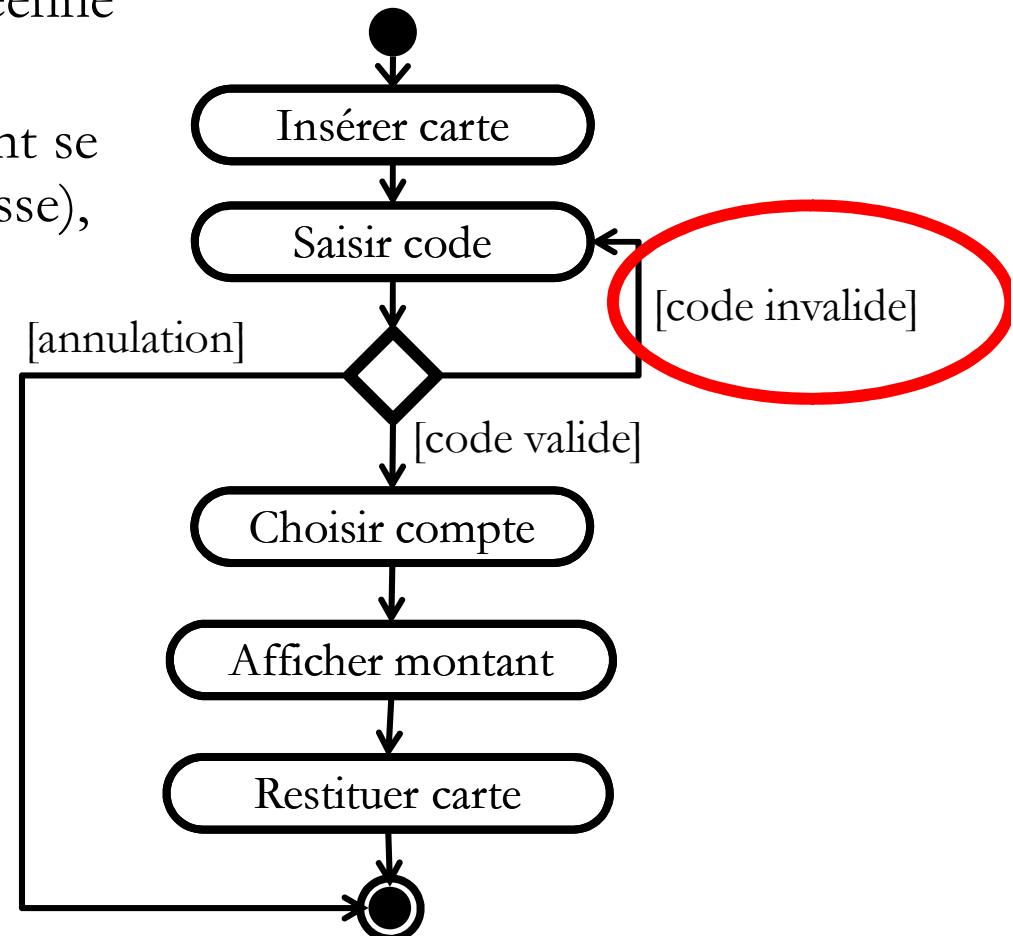


Diagramme d'activité : Partition

Les partitions (couloir, lignes d'eau, *swimlanes*)

Elles permettent de préciser qui effectue une action.

Les partitions n'ont pas de signification bien arrêtée, mais correspondent souvent à des acteurs ou des unités d'organisation du modèle (p.e. « service commande », « service facturation », « client »).

Diagramme d'activité : Partition

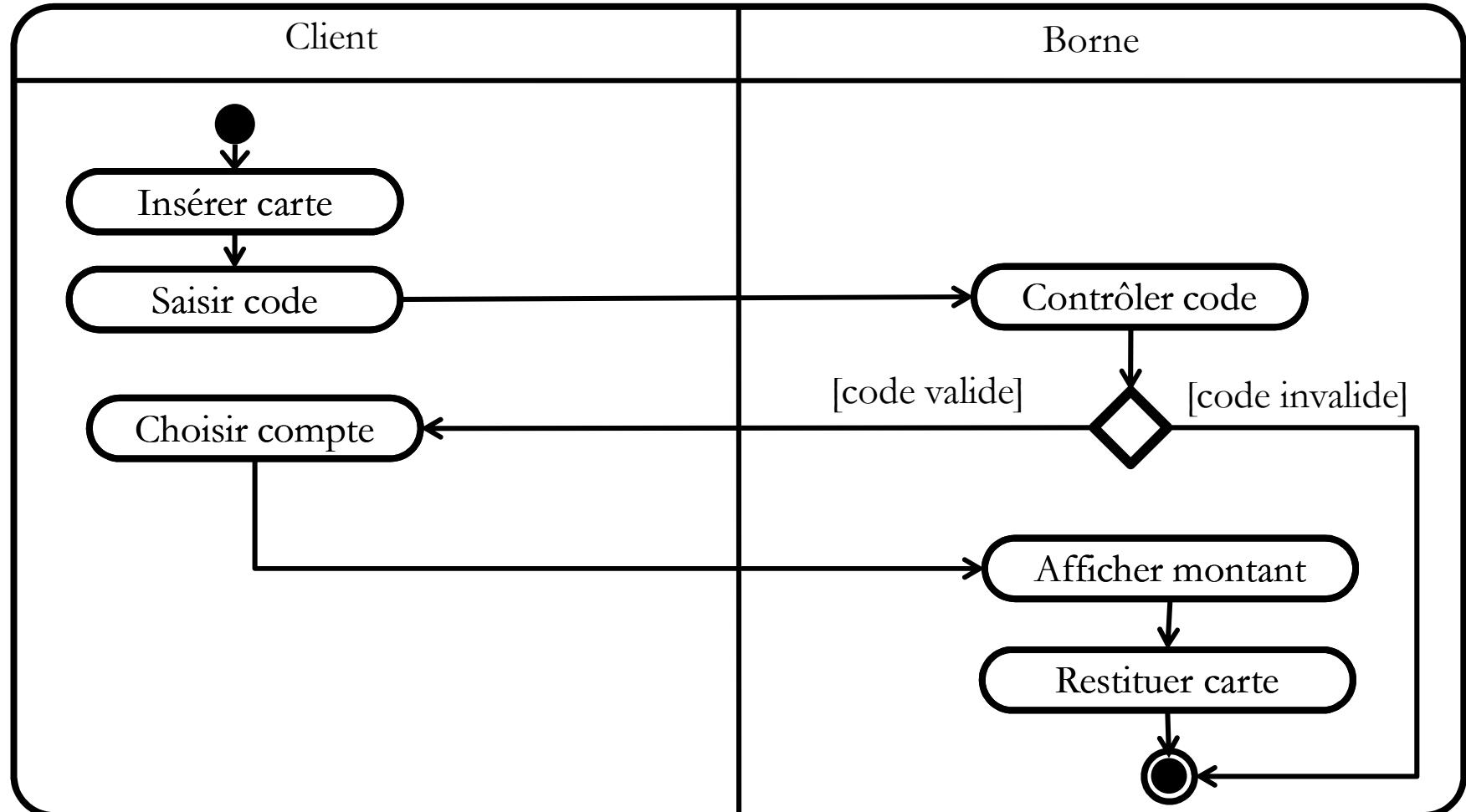


Diagramme d'activité : Région d'expansion

Région d'expansion

C'est un nœud d'activité s'exécutant une fois pour chaque élément d'une collection en entrée.

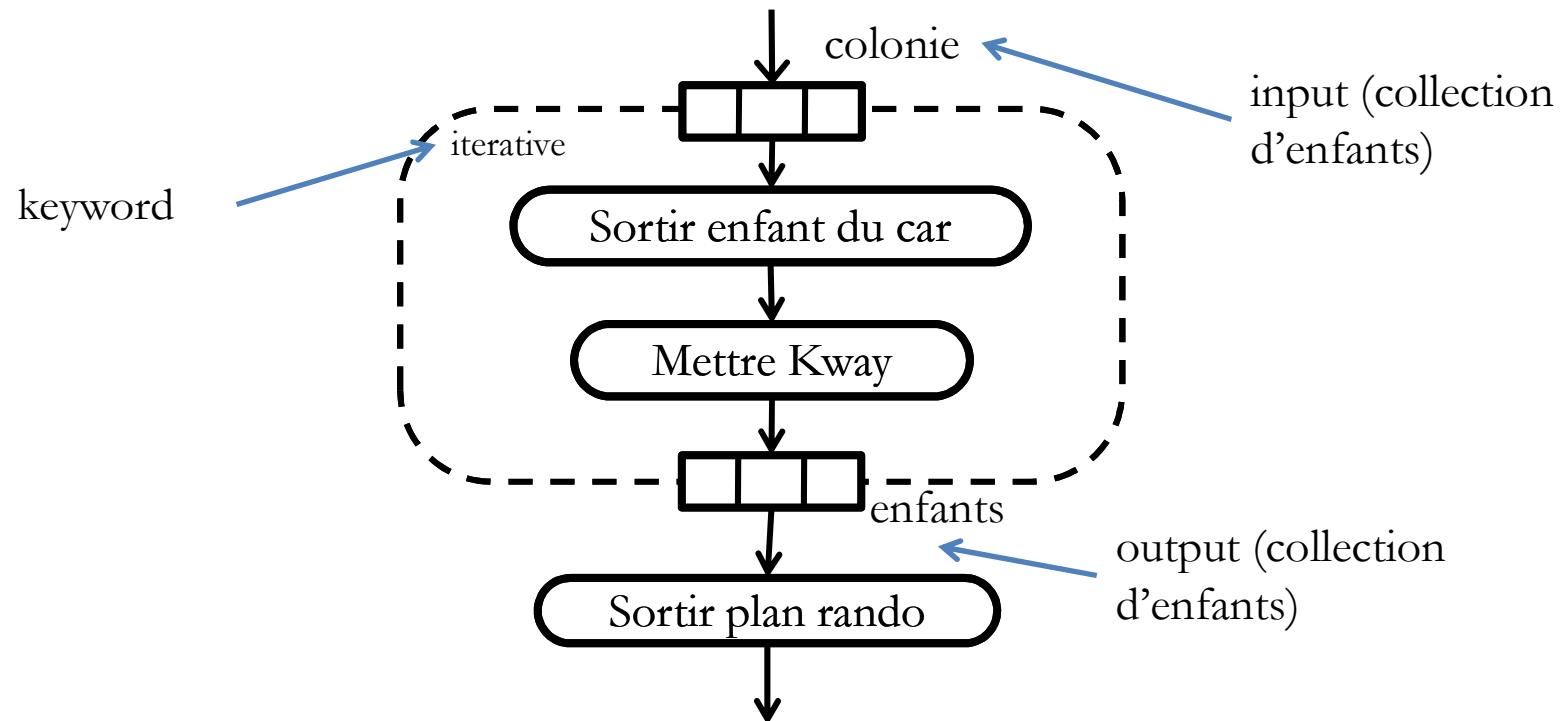
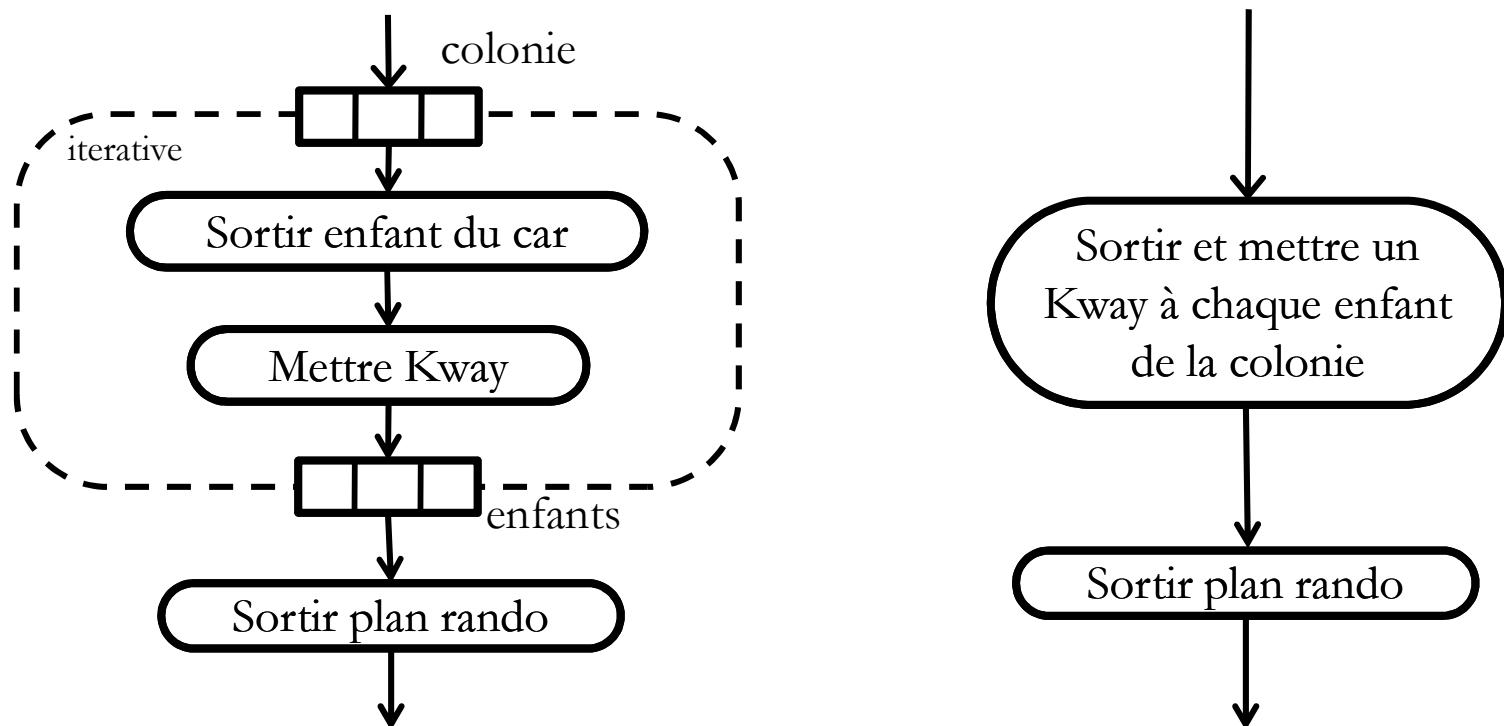


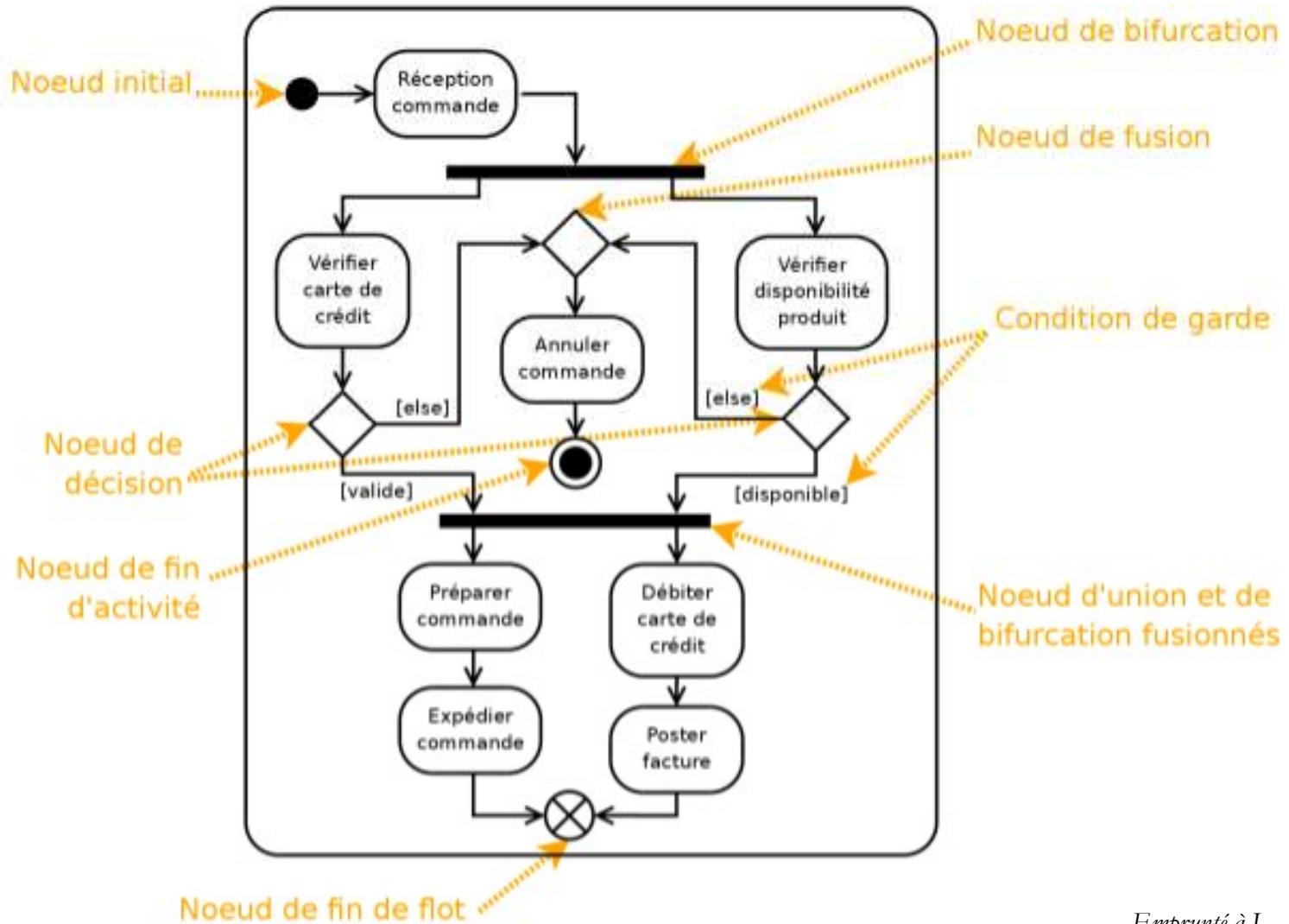
Diagramme d'activité : Région d'expansion

Région d'expansion... discussion

Il s'agit d'un niveau de détail assez fin → si en phase d'analyse, un nœud « simple » peut suffire



Fin de flot ou final



Emprunté à L. Audibert

Diagramme d'activité : Objets

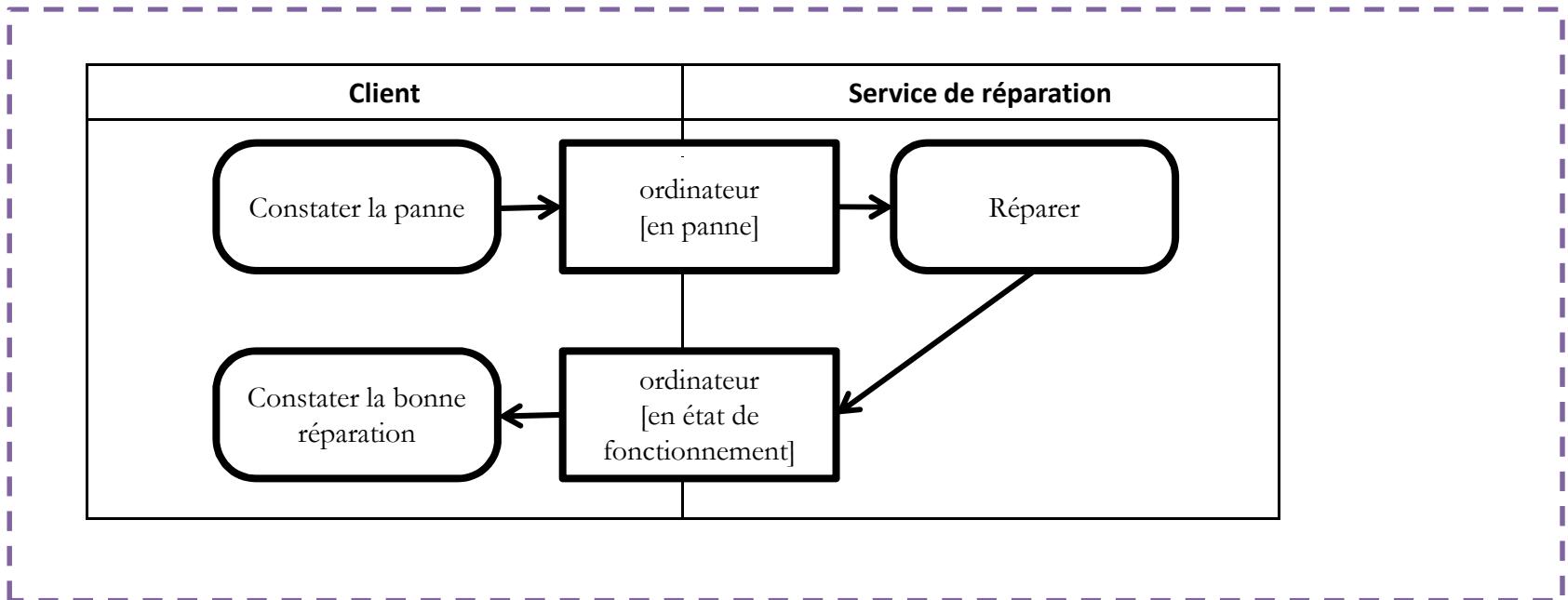
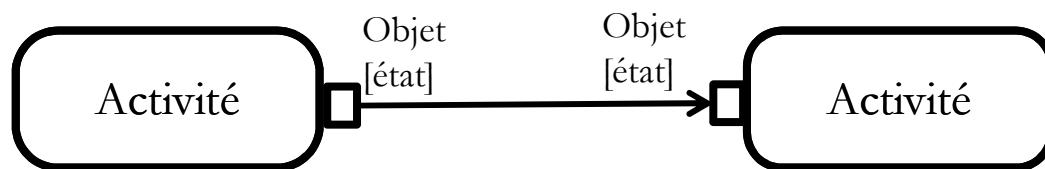


Diagramme d'activité : Objets

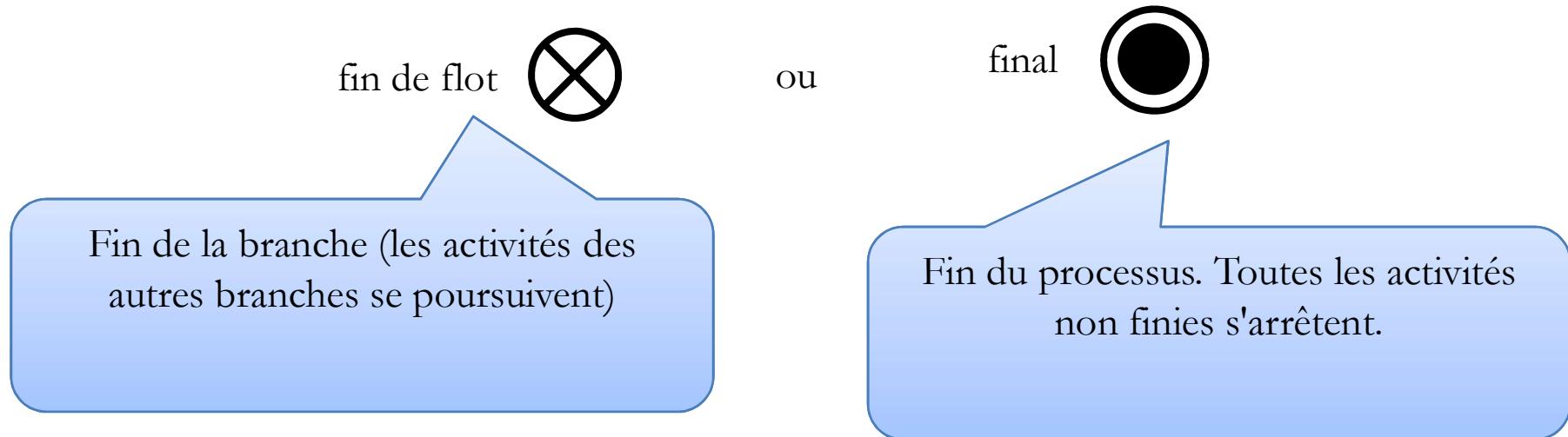
Discussion

Ne rajouter les objets que si cela a un intérêt.

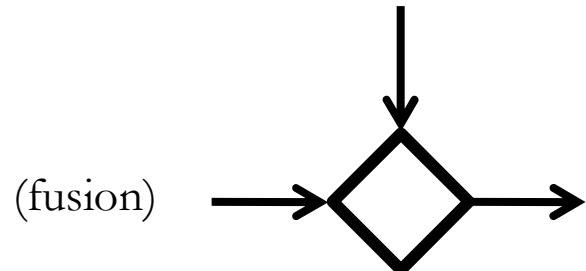
Notation équivalente



Quelques nuances

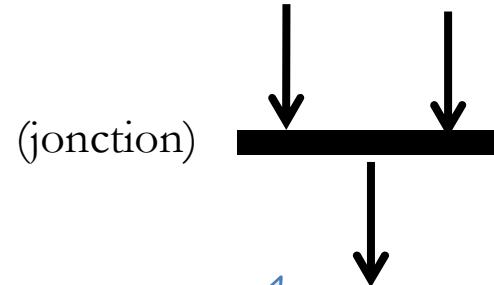


Quelques nuances



(fusion)

ou



(jonction)

Le flot sortant est exécuté dès qu'un flot entrant est activé.

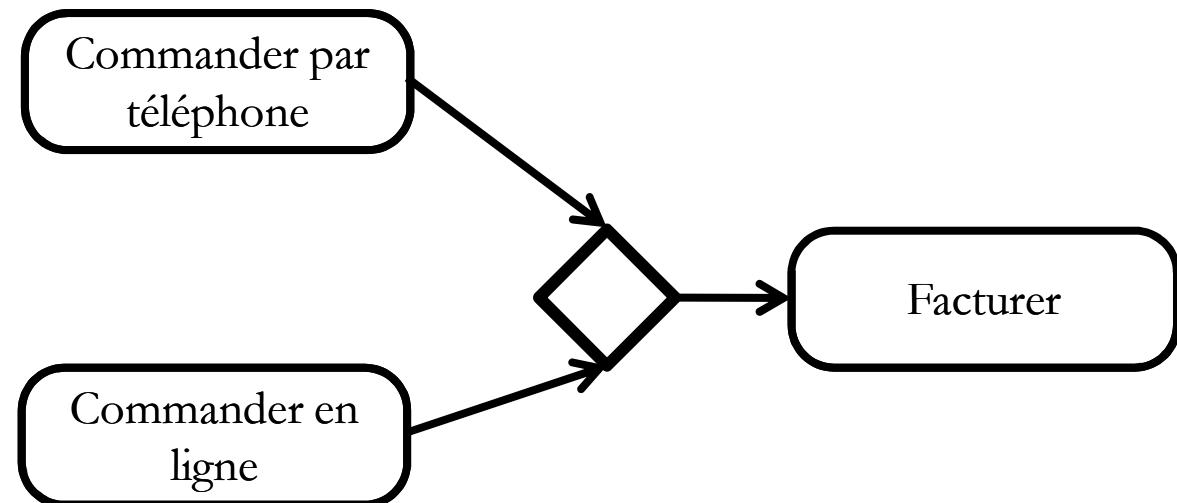
Les activités entrantes ne se synchronisent pas.

Utilisé pour les alternatives

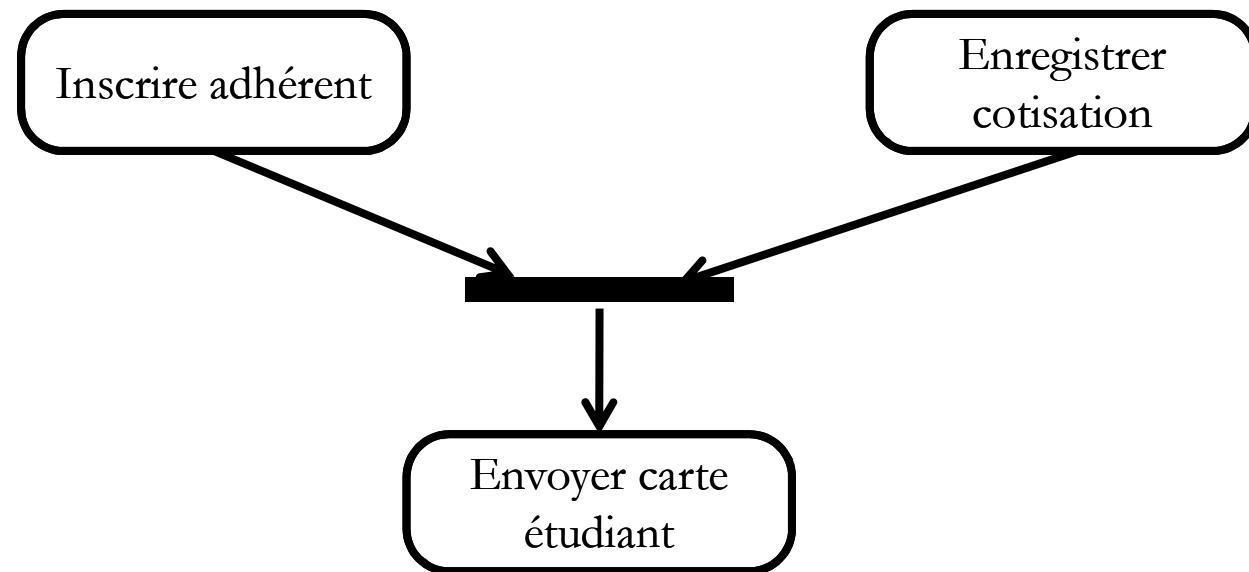
On attend que toutes les activités entrantes soient terminées pour passer à l'activité suivant le join : synchronisation des activités entrantes.

Utilisé pour les synchronisations

Quelques nuances

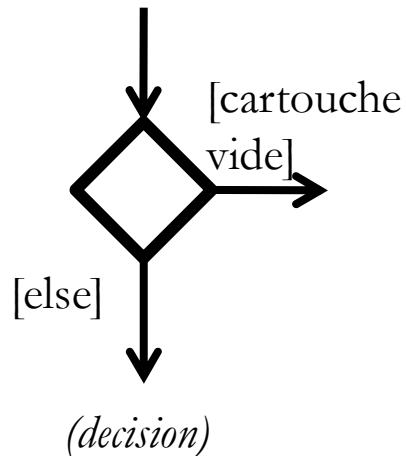


Quelques nuances



Quelques nuances

Nœud de décision



Les conditions de sorties doivent être exclusives : une seule sortie possible par passage dans le nœud de décision

diagramme de packages

PACKAGES DIAGRAM

Paquetage

197

- Dépendance : 

Au moins un élément du paquetage source utilise les services d'au moins un des éléments du paquetage destination (plutôt dans le cas où le diagramme de classe est détaillé)

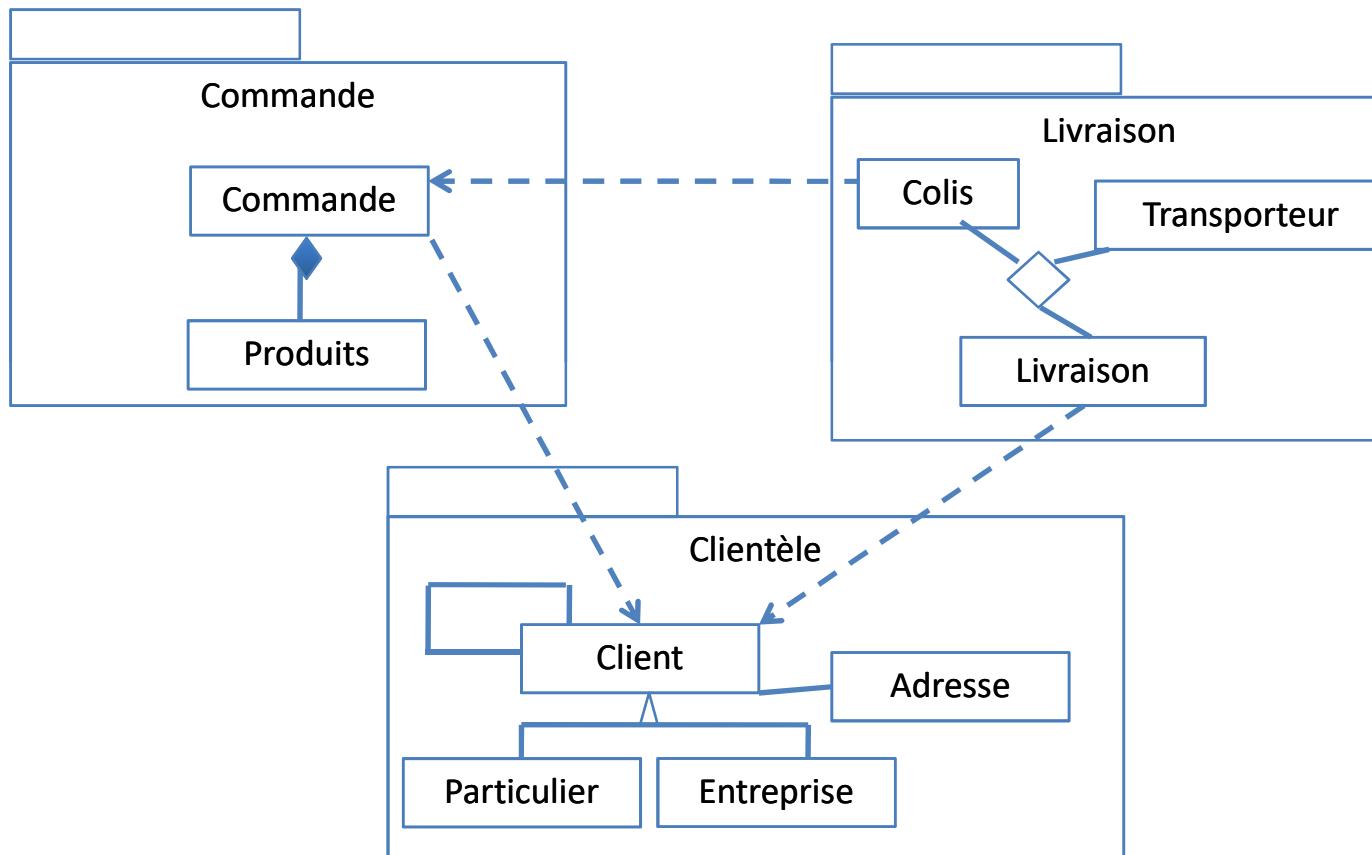
*Attention à ce qu'il n'existe pas de dépendance circulaire
(p.e. $p1 \rightarrow p2 \rightarrow p1$)*

- Héritage : 

Au moins un élément du paquetage source spécialise au moins un des éléments du paquetage destination

Paquetage

198



Packages diagram

199

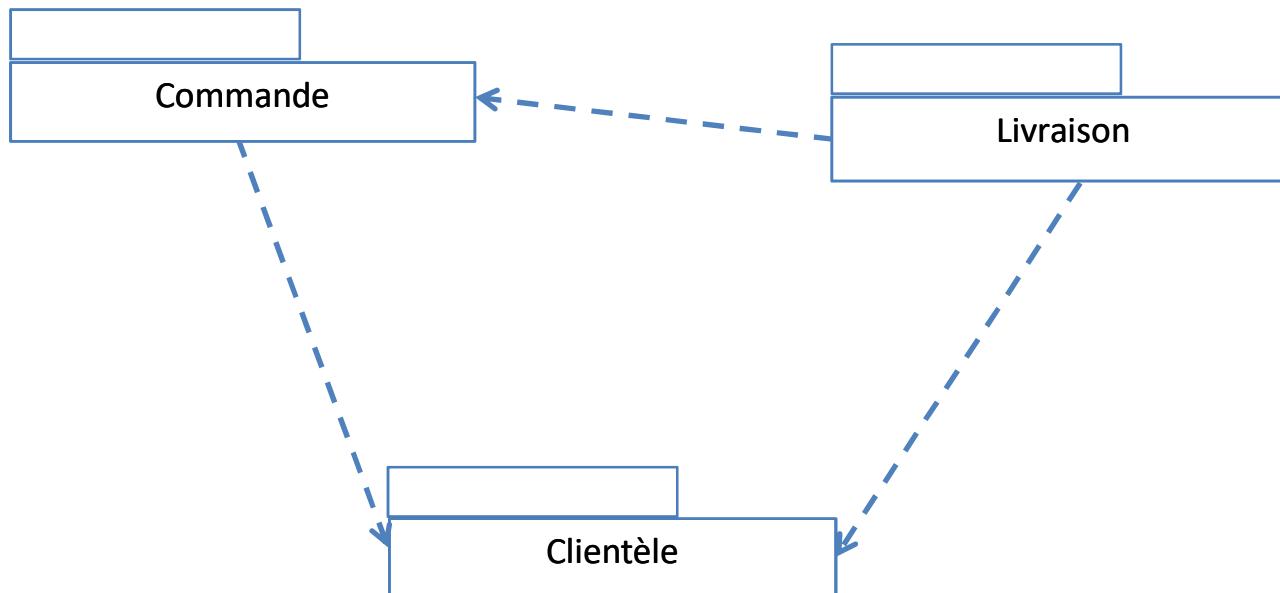


diagramme de composants

COMPONENT DIAGRAM

Diagramme de composants

201

- Pour la phase de mise en œuvre.
- Montre l'architecture physique d'une application en terme de modules : fichiers sources, exécutables, librairies, etc.
- Les dépendances entre composants permettent notamment d'identifier les contraintes de compilation et de mettre en évidence la réutilisation de composants.

Diagramme de composants

202

Un composant est une partie modulaire, déployable et remplaçable de système. Il encapsule l'implantation et expose ses interfaces

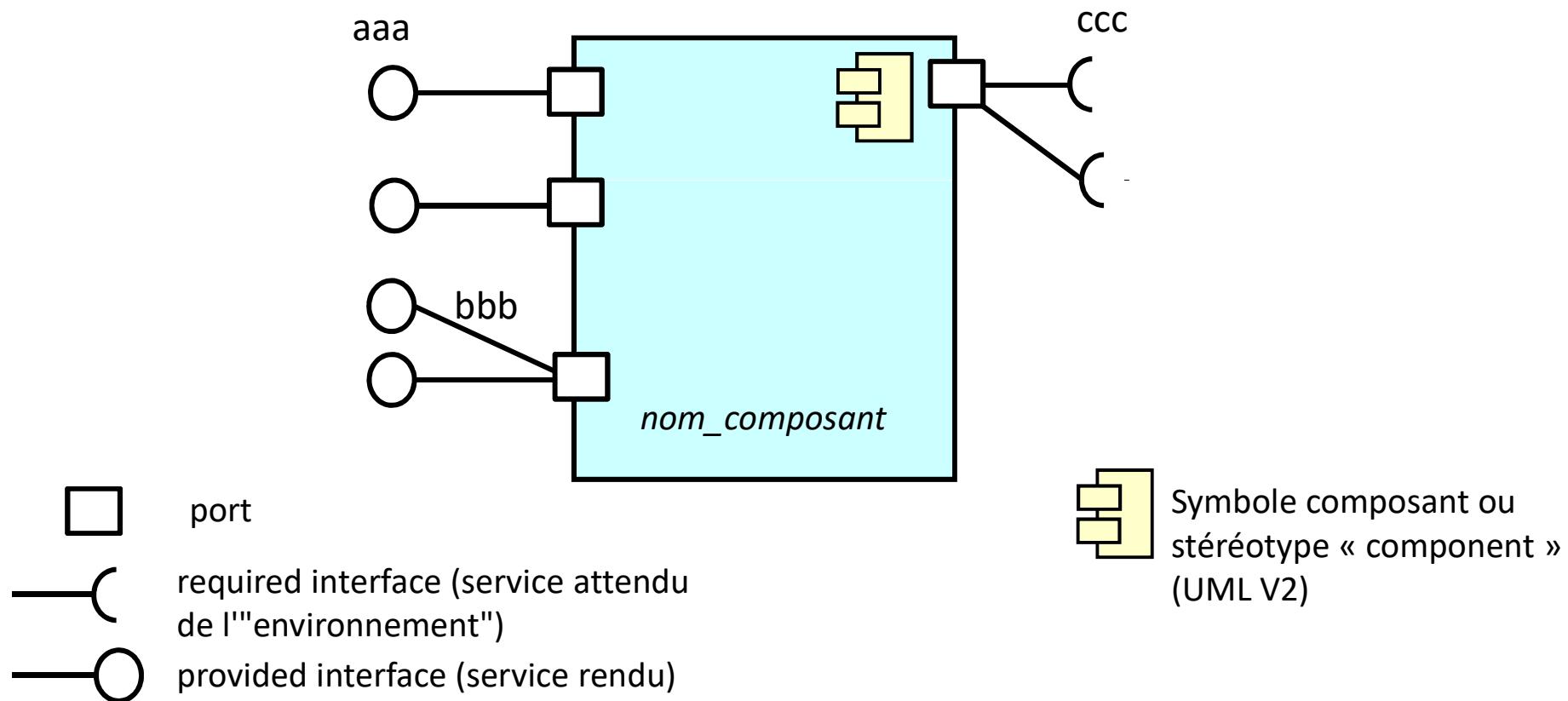


Diagramme de composants

203

- Un diagramme de composants montre l'assemblage des composants par leurs interfaces

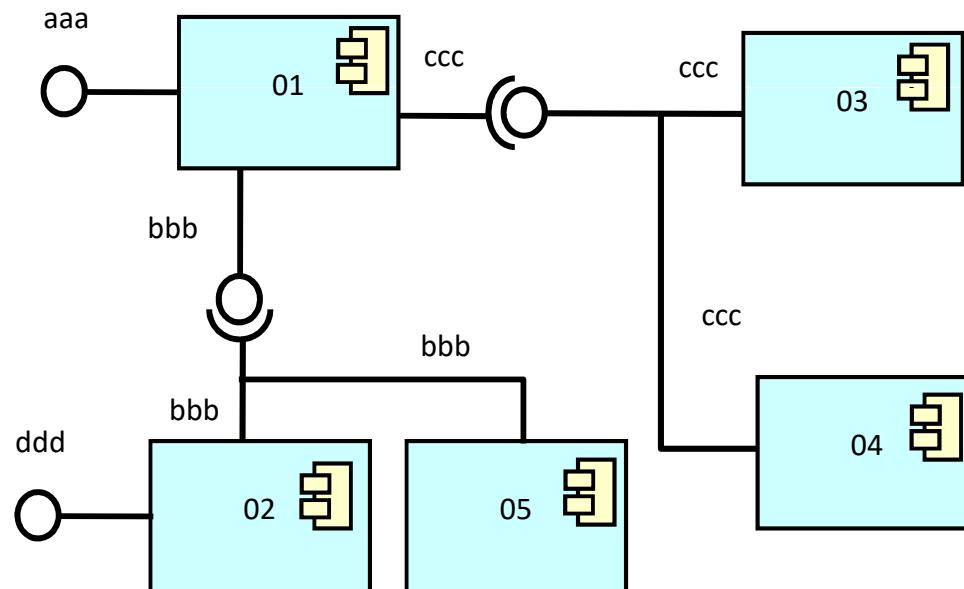
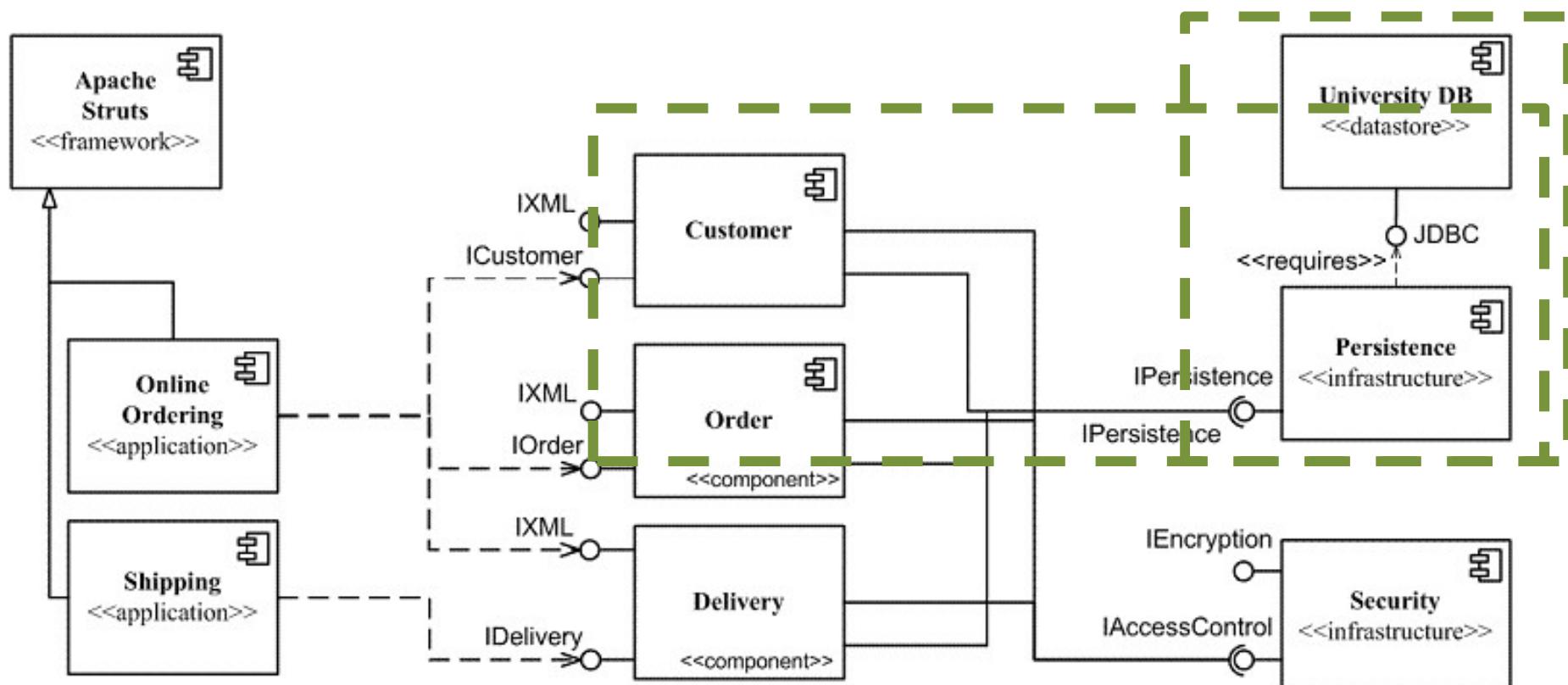


Diagramme de composants

204



Copyright 2005 Scott W. Ambler

Issu du site Agile Modelling de Scott W. Ambler

Introduction à UML

diagramme de déploiement

DEPLOYMENT DIAGRAM

Diagramme de déploiement

206

- Pour la phase de mise en œuvre.
- Montre la configuration physique des différents matériels nécessaires à l'exécution du système, ainsi que des artefacts qu'ils supportent.
- Nœuds (ressources matérielles) connectés avec des liens physiques.

Diagramme de déploiement

207

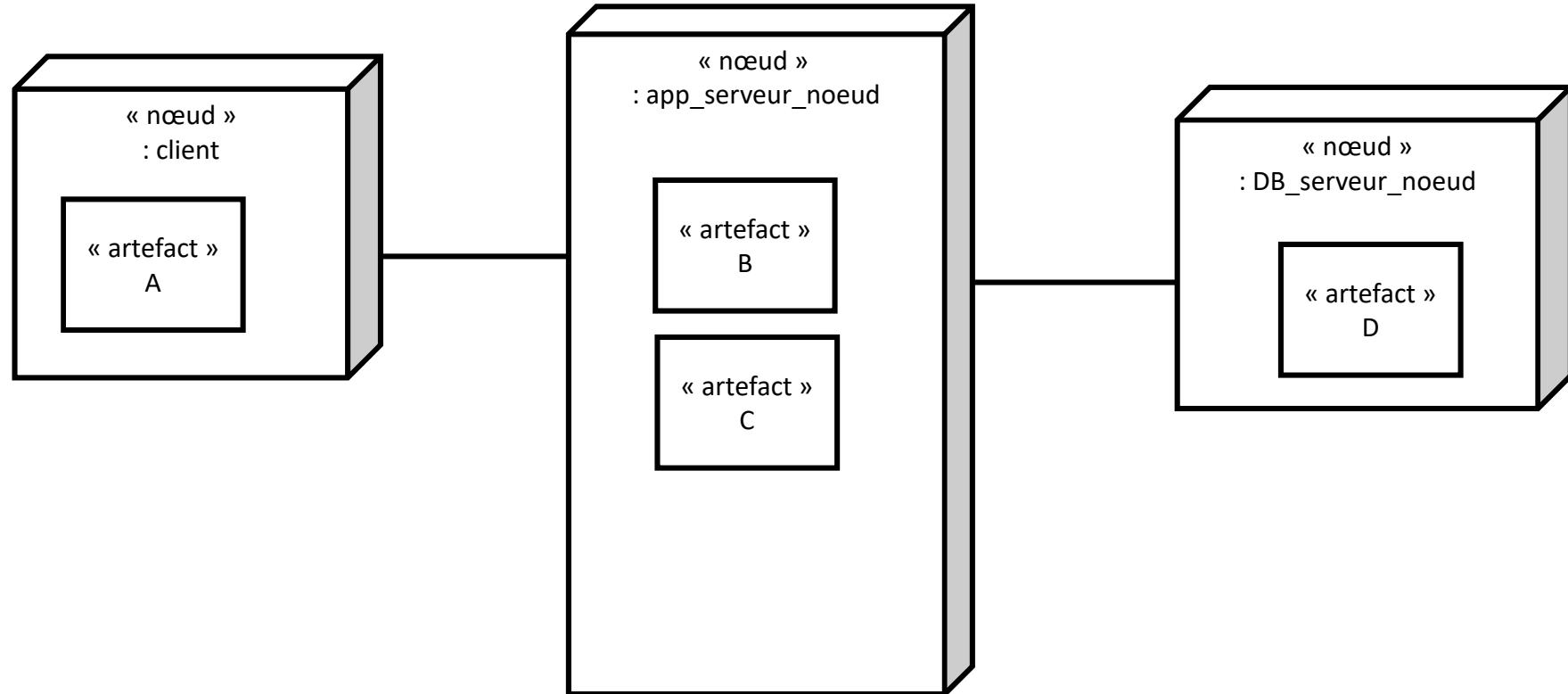
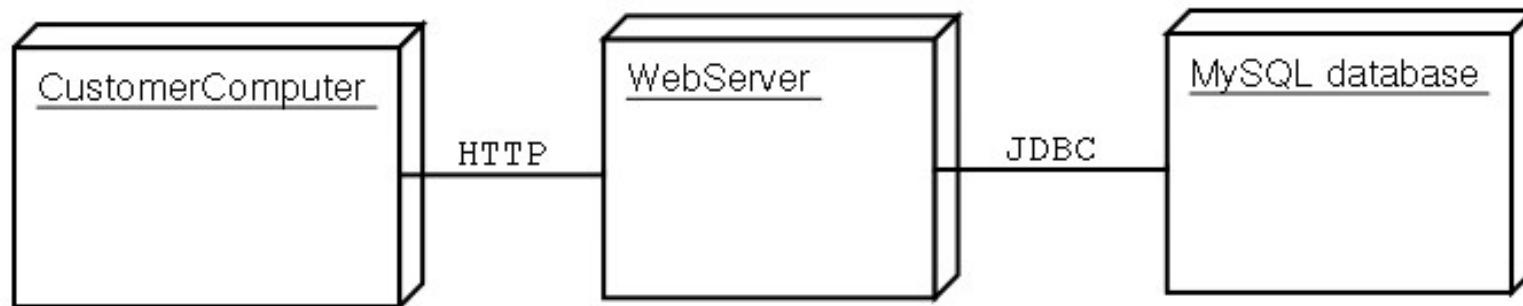


Diagramme de déploiement

208

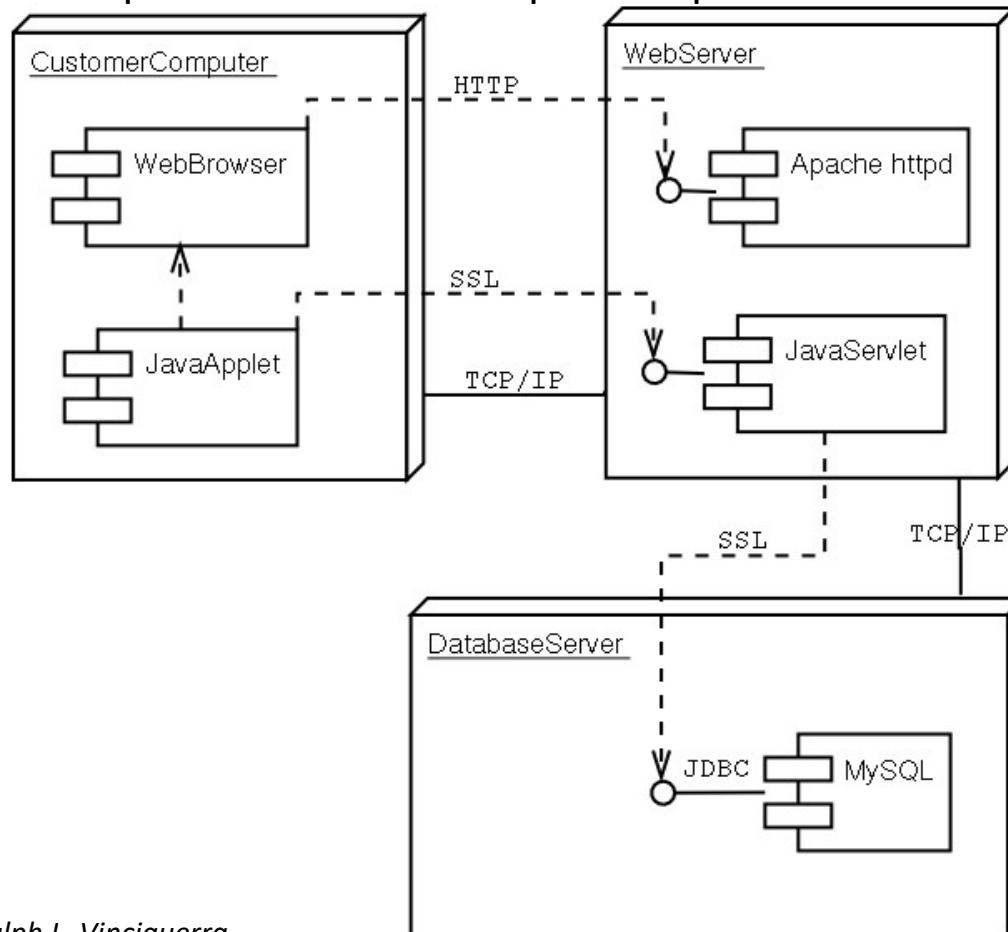


Auteur du diagramme : *Ralph L. Vinciguerra*

Introduction à UML

Diagramme de déploiement + composants

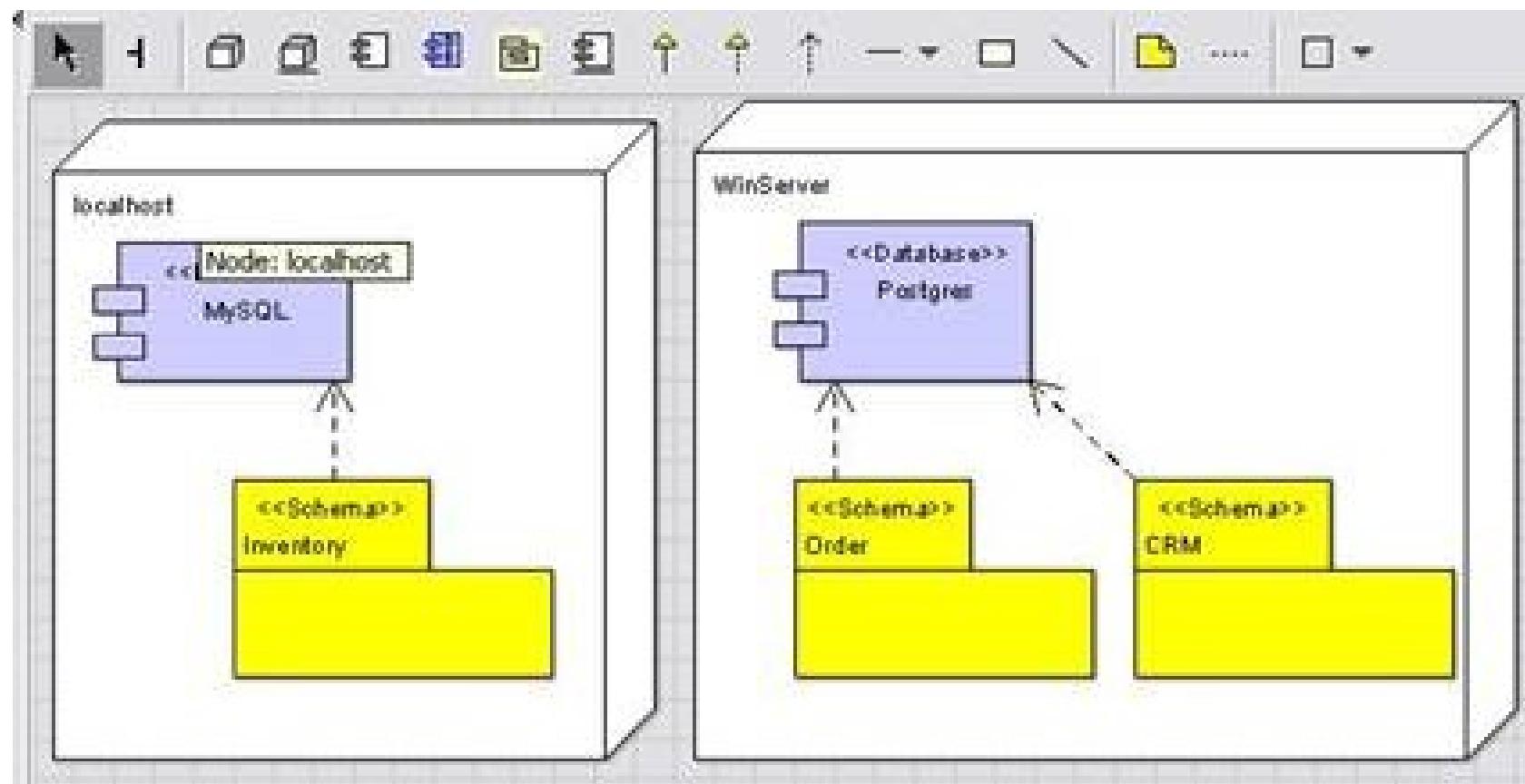
Les diagrammes de déploiement et de composants peuvent être intégrés



Auteur du diagramme : *Ralph L. Vinciguerra*

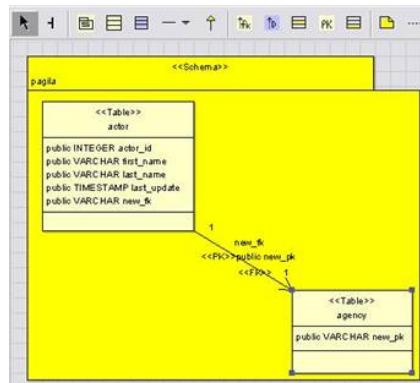
Introduction à UML

Diagramme de déploiement + composants

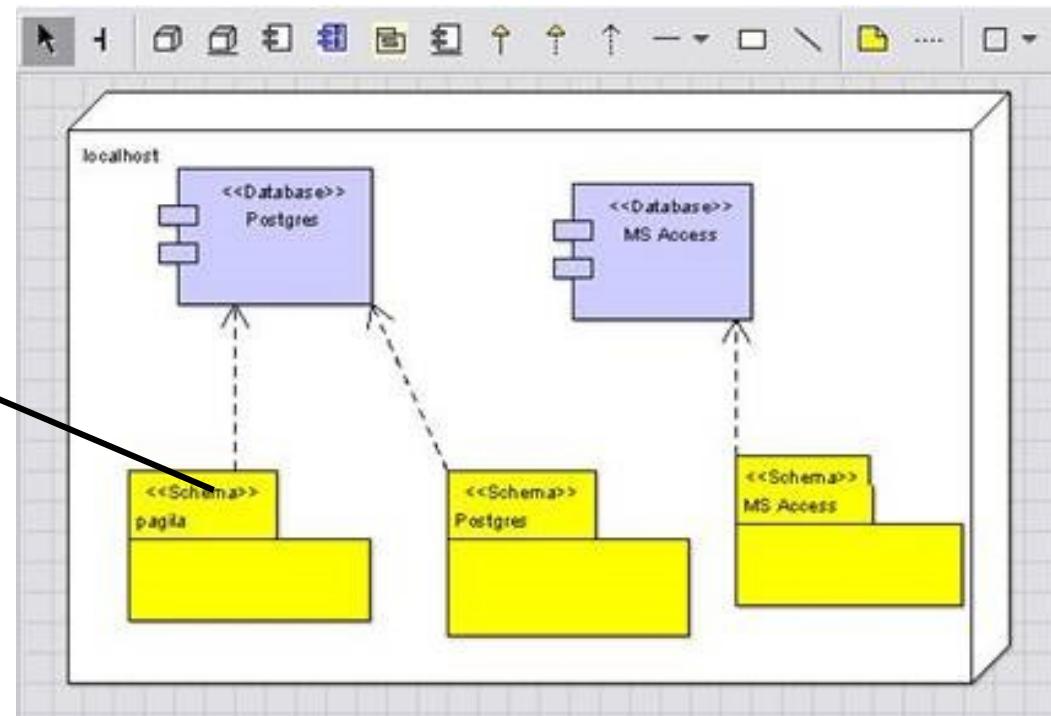


Source : ArgoUML Database Modeling project
at <http://argouml-db.tigris.org/>

Diagramme de déploiement + composants



Eventuellement décrit
dans un diagramme de classes



Source : ArgoUML Database Modeling project
at <http://argouml-db.tigris.org/>

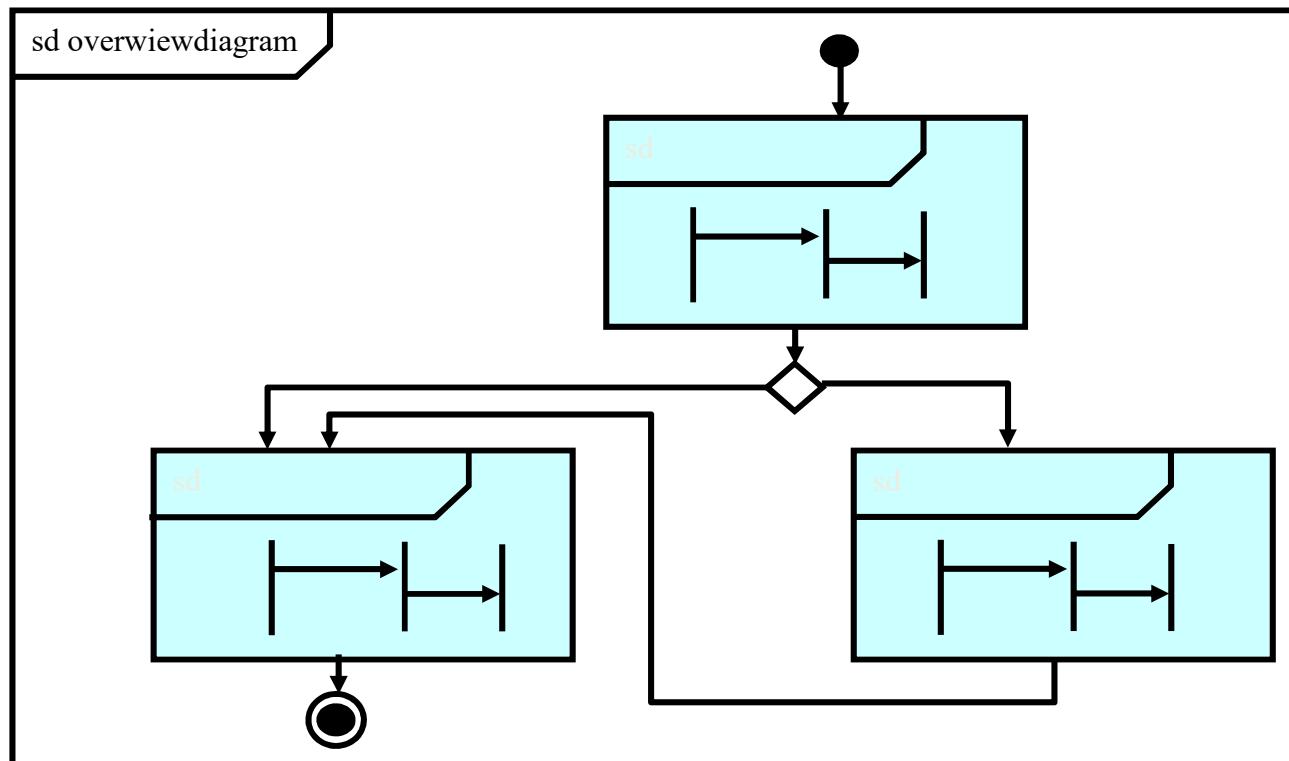
diagramme global d'interaction

INTERACTION OVERVIEW DIAGRAM

Diagramme d'interaction

213

Enchaînement de diagrammes de séquence connectés via le formalisme d'un diagramme d'activités



© Jacques Callot

Introduction à UML

diagramme de temps

TIMING DIAGRAM

Diagramme de temps

215

Indique les changements d'états d'un objet par rapport au temps (pour formaliser les systèmes composés d'objets dont la synchronisation est délicate comme la modélisation de moteurs à injection, de réacteurs, etc)

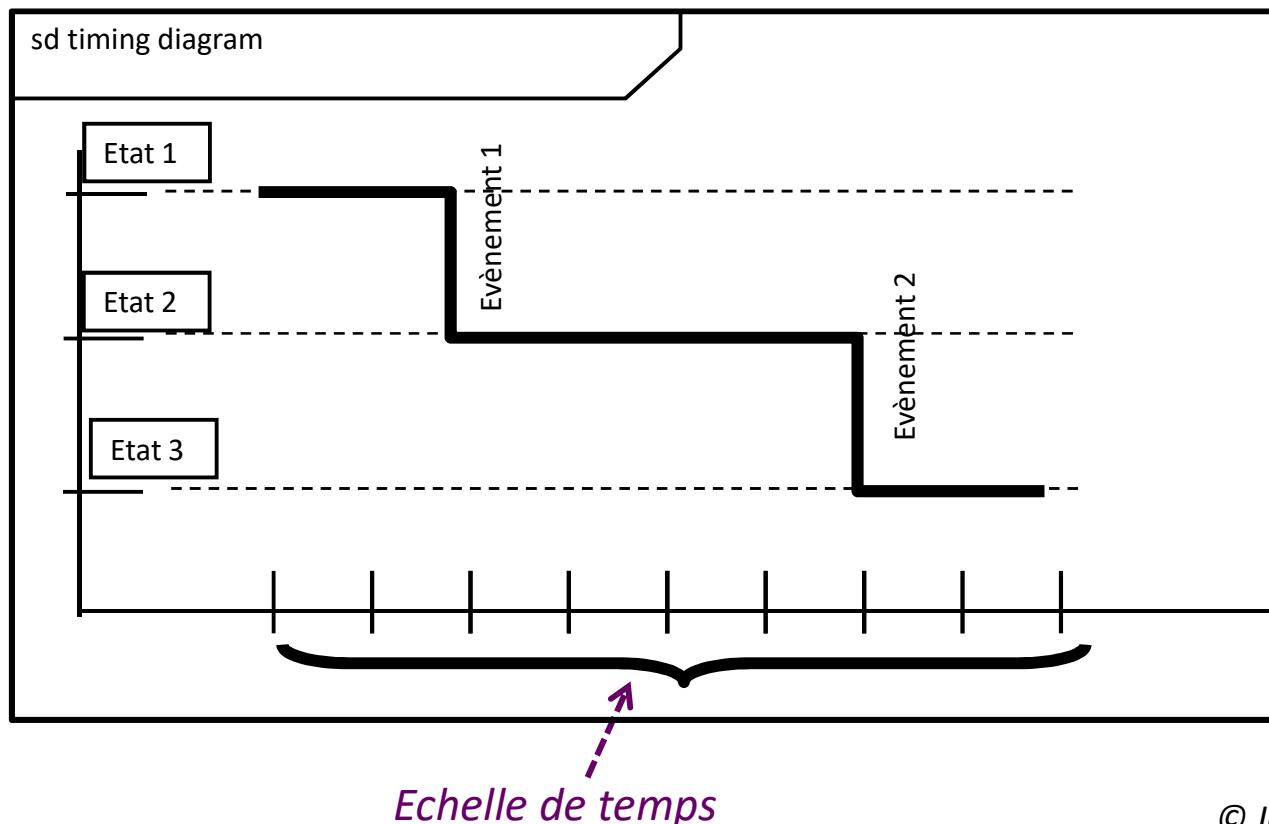
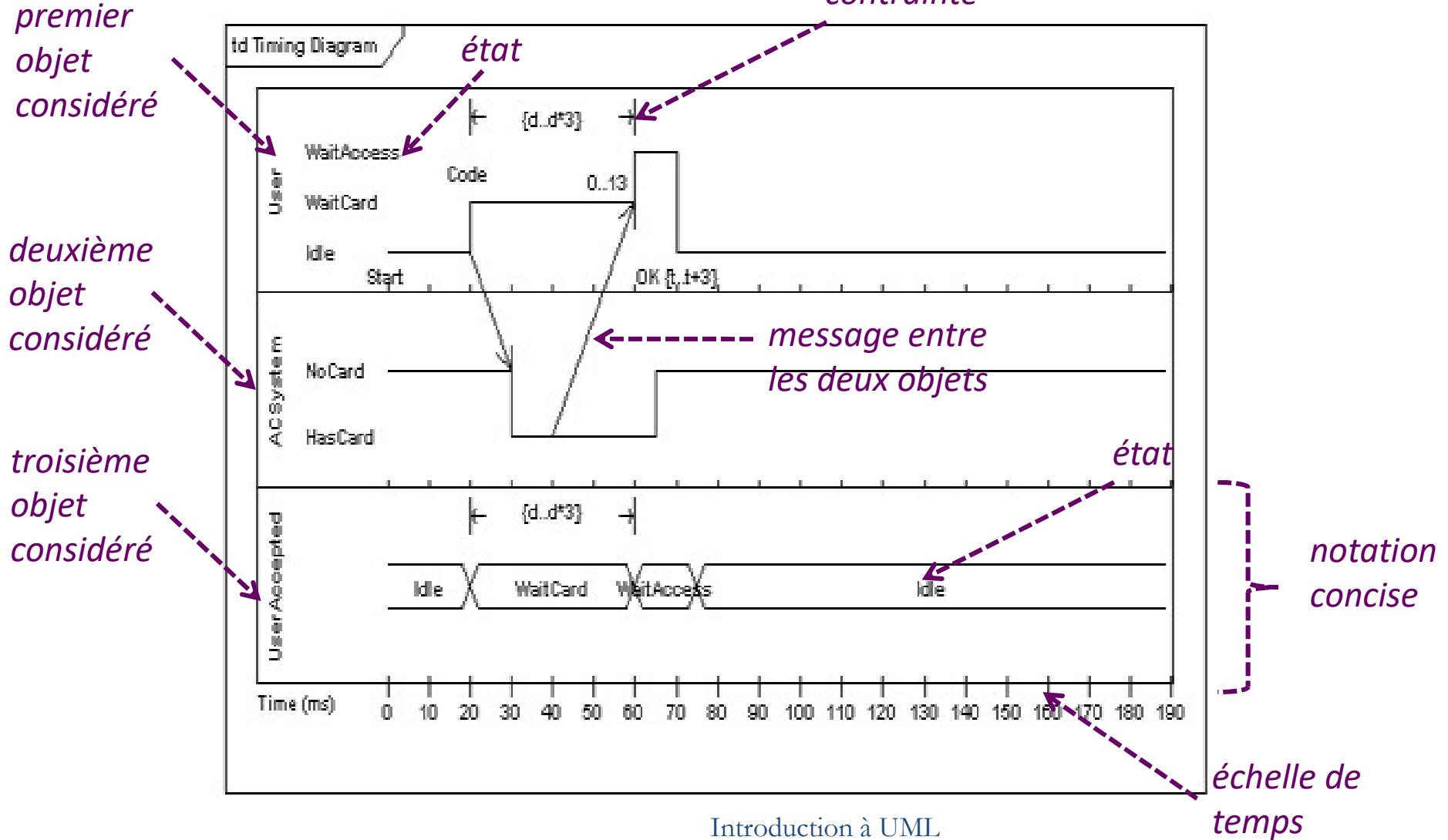


Diagramme de temps

216



S'applique à tous les diagrammes

DIAGRAMMES : GÉNÉRALITÉS

Formalisme (rappel)

218

<entête>

<contenu>

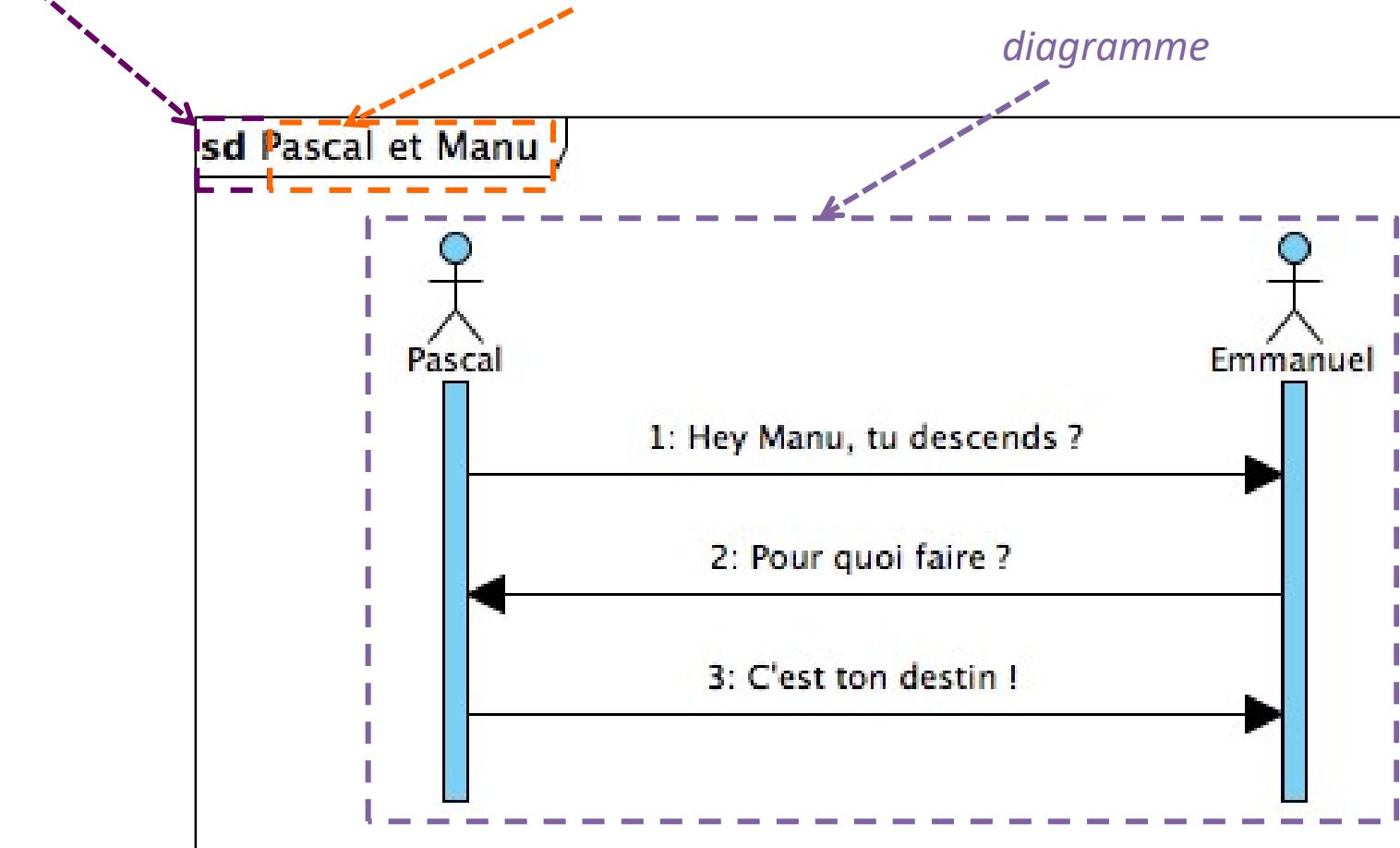
Formalisme

219

type de diagramme

nom du diagramme

diagramme



Introduction à UML

Formalisme (types de diagramme)

- activity
- class
- component
- deployment
- interaction
- package
- state machine
- use case

et abréviations suivantes :

- act (for activity frames)
- cmp (for component frames)
- dep (for deployment frames)
- sd (for interaction frames)
- pkg (for package frames)
- stm (for state machine frames)
- uc (for use case frames)

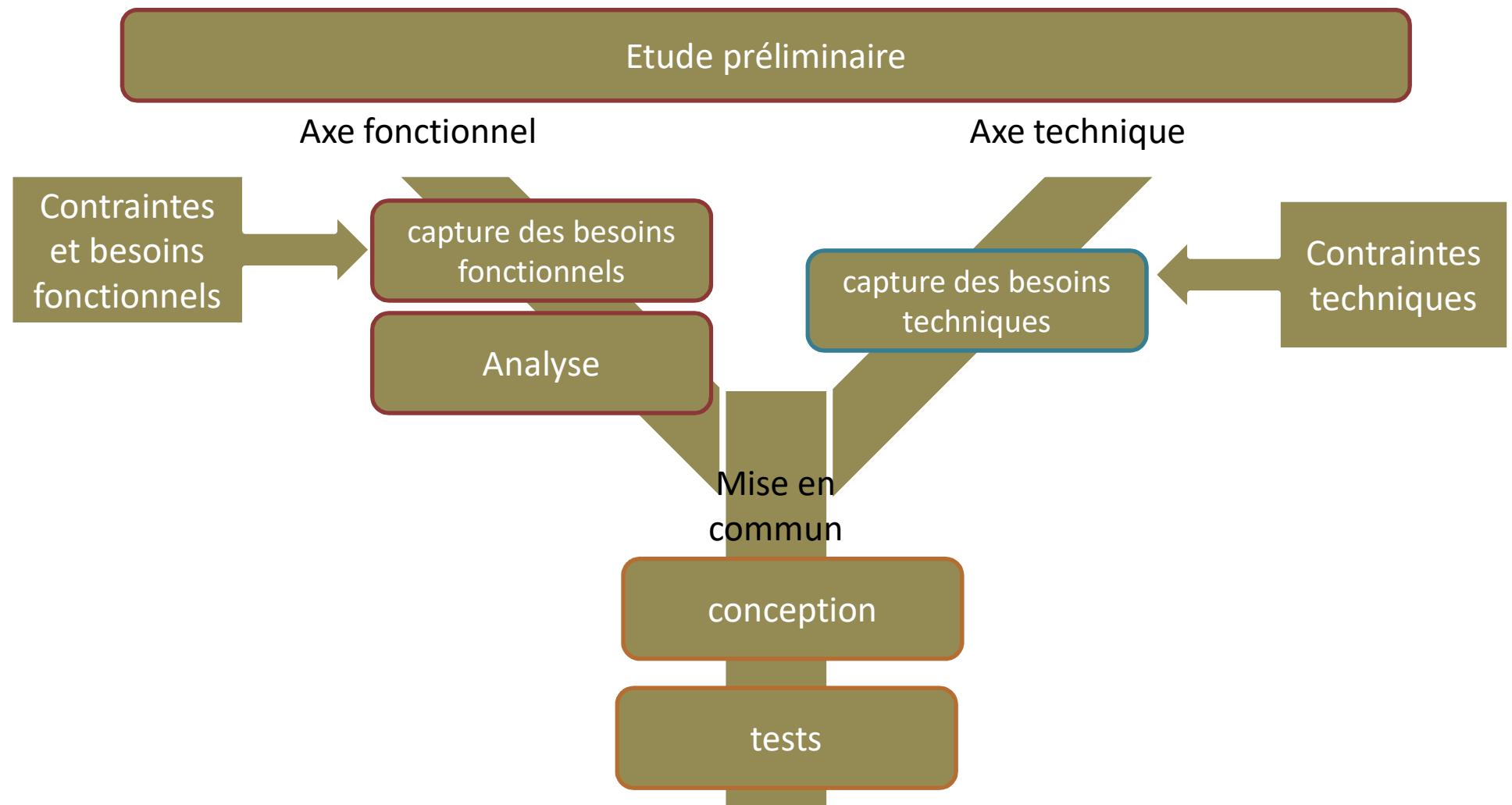
INGÉNIERIE DES SYSTÈMES

UML dans un Processus unifié

D'INFORMATION

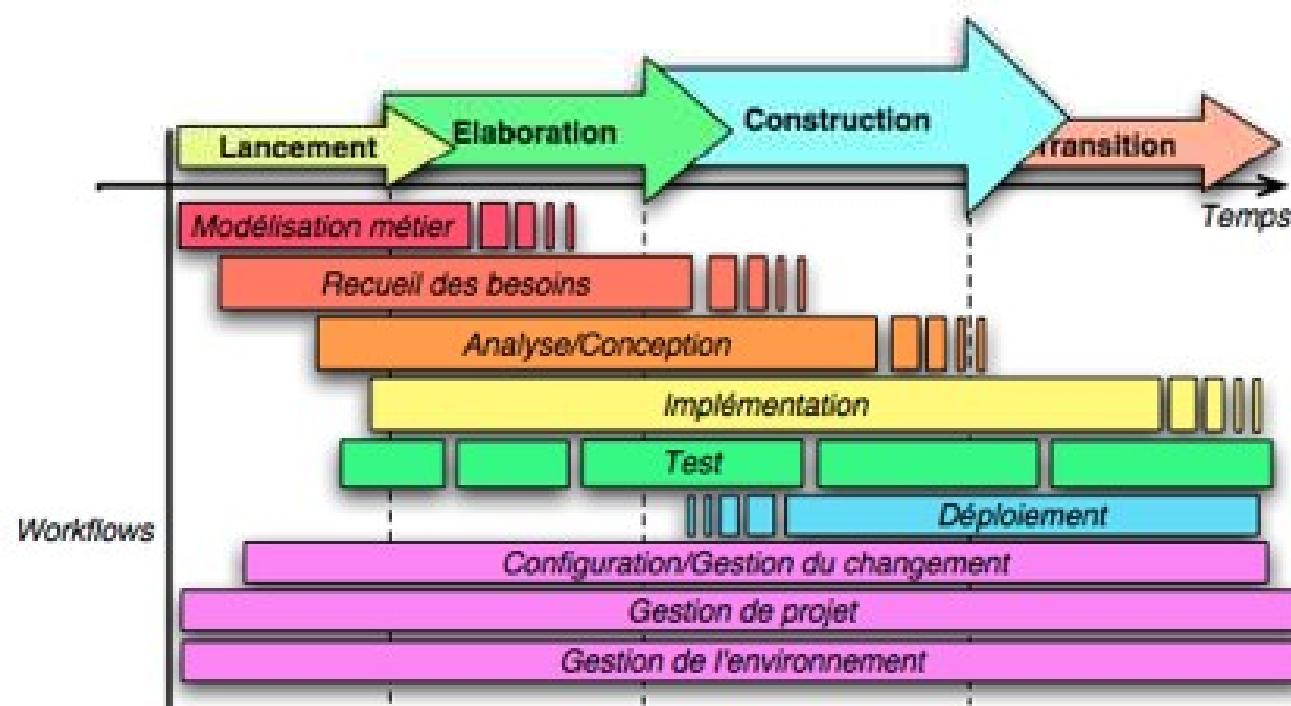
Processus uniifié

Exemple : 2TUP (2 Tracks Unified Process)

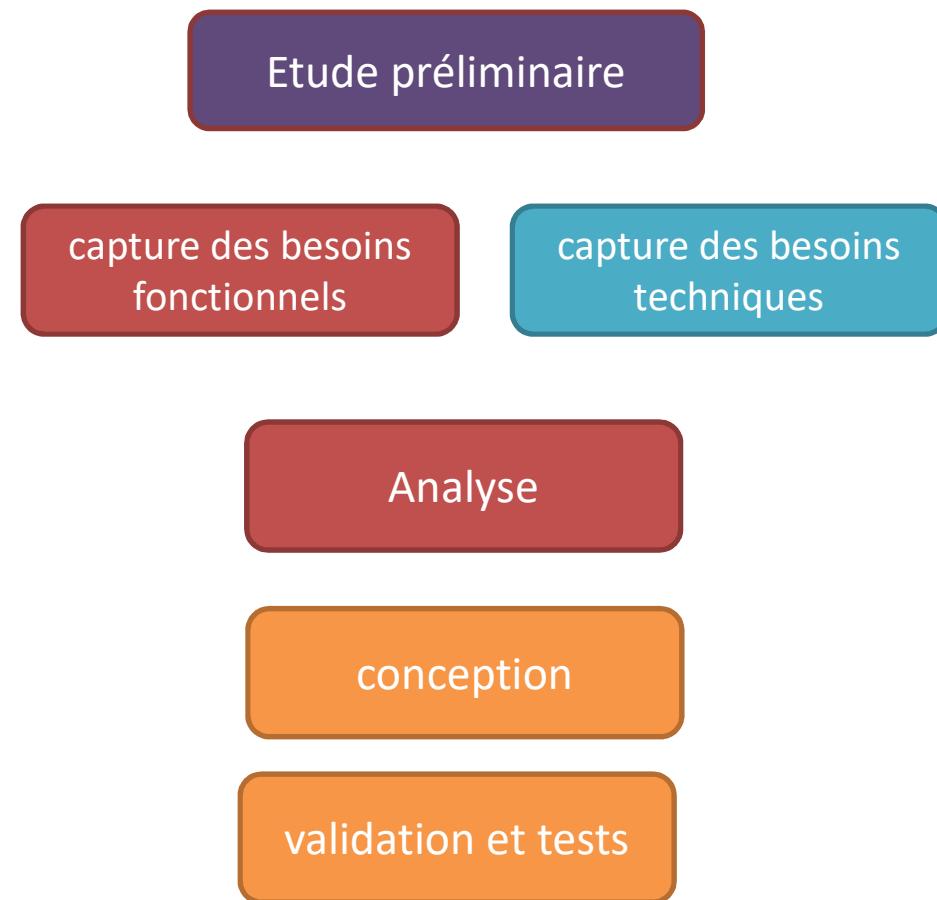


Appliquer un processus uniifié à la lettre ?

- Dans la pratique, non.
- Activités toujours présentes dans un UP :



Processus uniifié : quelques grandes étapes



Processus unifié : quelques grandes étapes



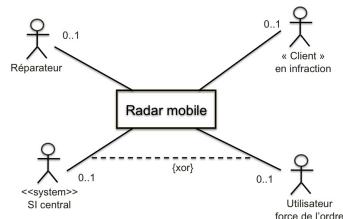
Objectifs :

- Appréhender et décrire le contexte : les acteurs, les problèmes, les interactions acteur/système (éventuellement via un audit du système actuel),
- Donner les premières orientations fonctionnelles et techniques.

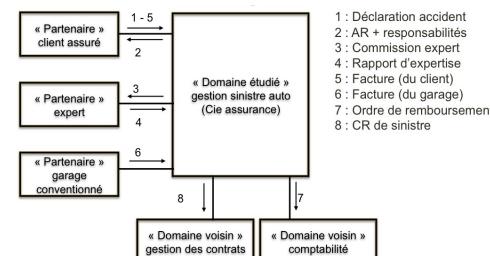
Etude préliminaire (cadrage)

Grandes étapes

- Identifier les *acteurs*
- Identifier les *messages*
- Modéliser le *contexte statique*
- Modéliser le *contexte dynamique*

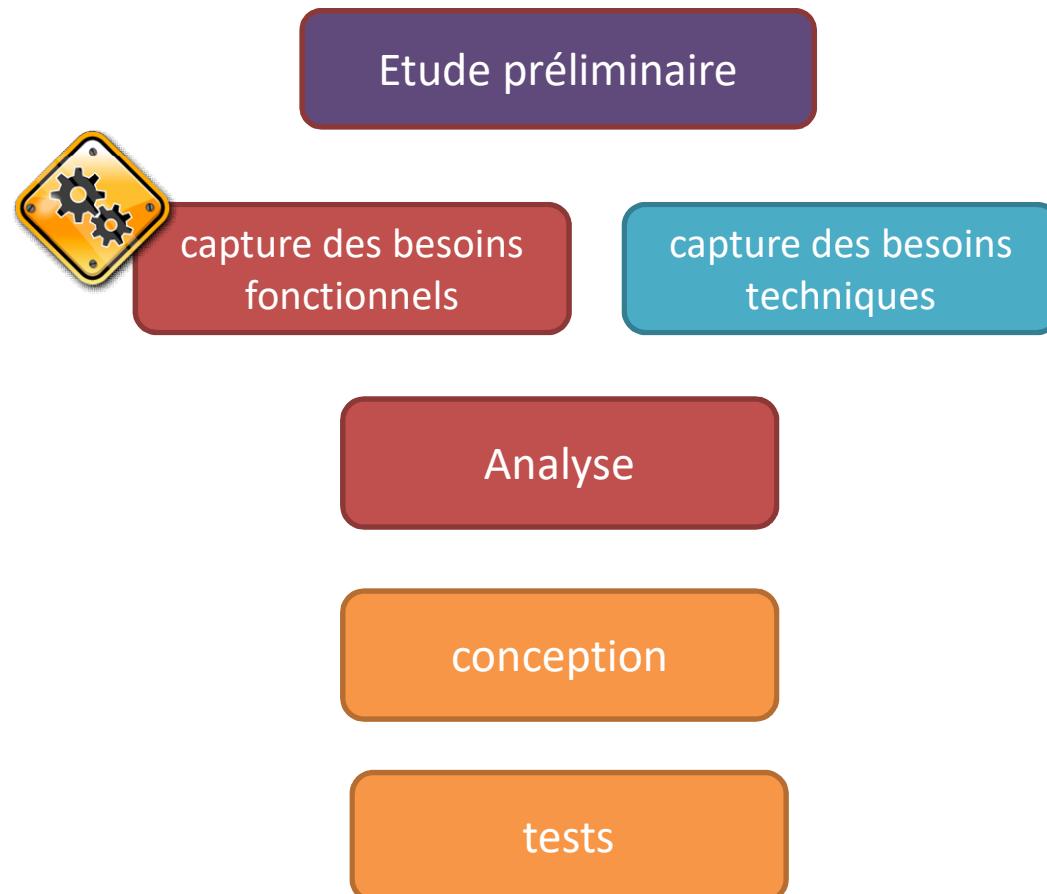


diag. de communication
sans messages



diag. de communication
avec messages

Processus unifié : quelques grandes étapes



Objectifs

- Recueillir de besoins utilisateurs (compléter étude précédente)
- Préparer l'analyse

Pour cette activité de capture des besoins fonctionnels, suivons 2TUP pour nous donner une guideline

Capture des besoins fonctionnels

Les cas d'utilisation : identification

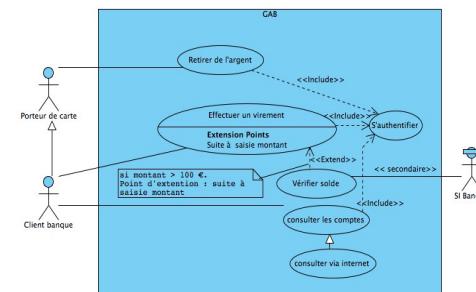
1- Identifier les cas d'utilisation (CU)

- Pour chaque acteur, identifier les CU.
- Ne conserver que ceux qui rendent un service « notable à l'acteur »

Attention à ne pas descendre trop bas dans la granularité: une CU n'est pas une fonction

Modifier une commande

Modifier la date de livraison dans une commande



diag. de cas d'utilisation

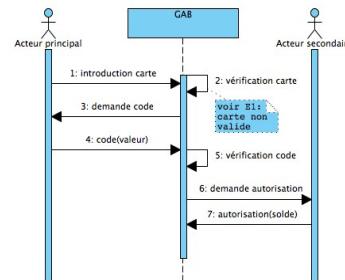
2 - Décrire textuellement les cas d'utilisation (CU)

Description textuelle l'expliquant appelée *fiche de cas d'utilisation* non normalisée par UML contenant généralement un descriptif général (titre, objectifs, dates, version, résumé, responsable, acteurs, ...), un descriptif des enchaînements (scenario nominal, enchaînements alternatifs, d'exception, pré-conditions, post-conditions), des éléments optionnels d'exigences non fonctionnelles

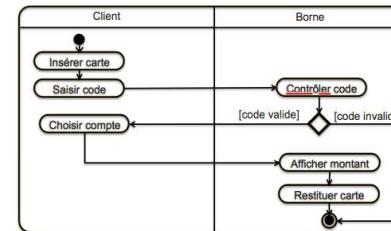
Capture des besoins fonctionnels

Les cas d'utilisation : description graphique

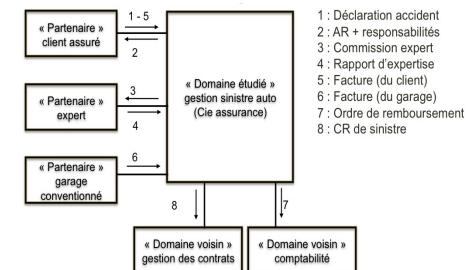
3 - Décrire graphiquement les cas d'utilisation (CU)



diag. de séquence système

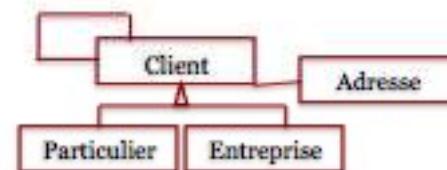


diag. d'activités ou d'états



diag. de communication

4 - Identifier les classes participantes à chaque CU

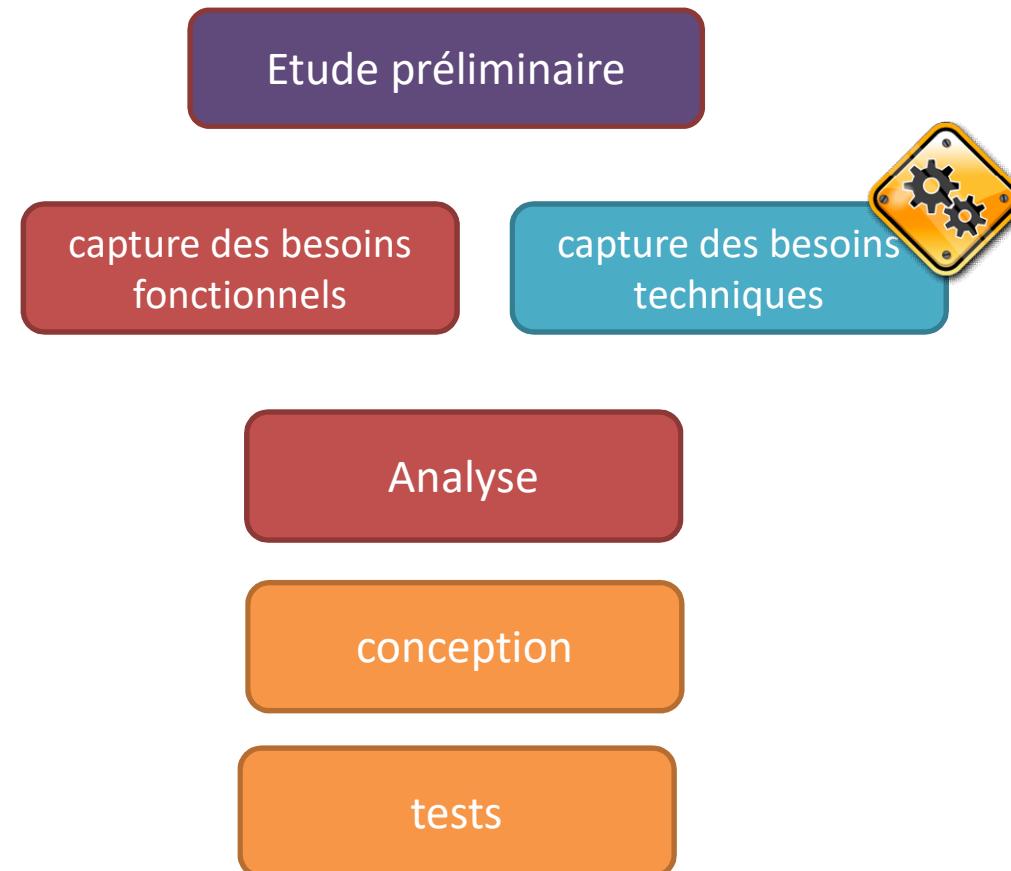


diag. de classes

Penser à identifier les besoins en termes de qualité des données pour chaque CU.

Description textuelle et éventuellement contraintes ajoutées sur les diagrammes précédents.

Processus unifié : quelques grandes étapes

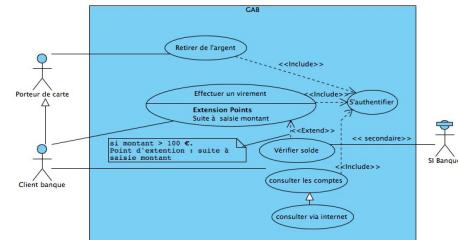


Objectifs :

- Recueillir les besoins et contraintes matérielles, logicielles

Contient (1/2):

- Identification des CU techniques

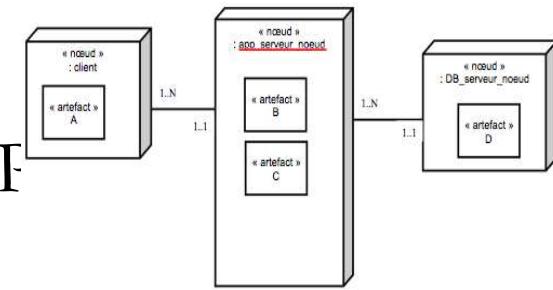


+ description de chaque CU

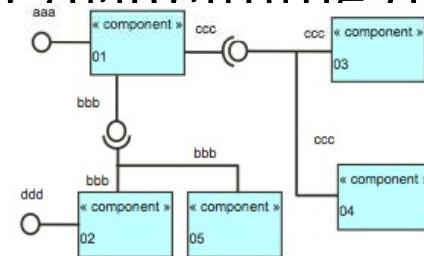
diag. de cas d'utilisation

Contient (2/2) :

- Configuration matérielle (serveurs, CI)

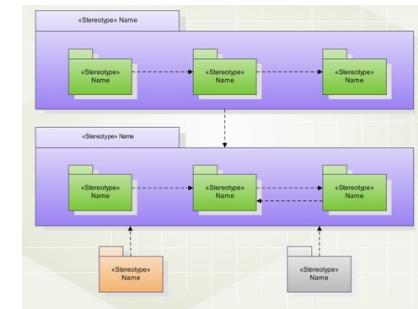


- Premier diagramme de composants



diag. de composants

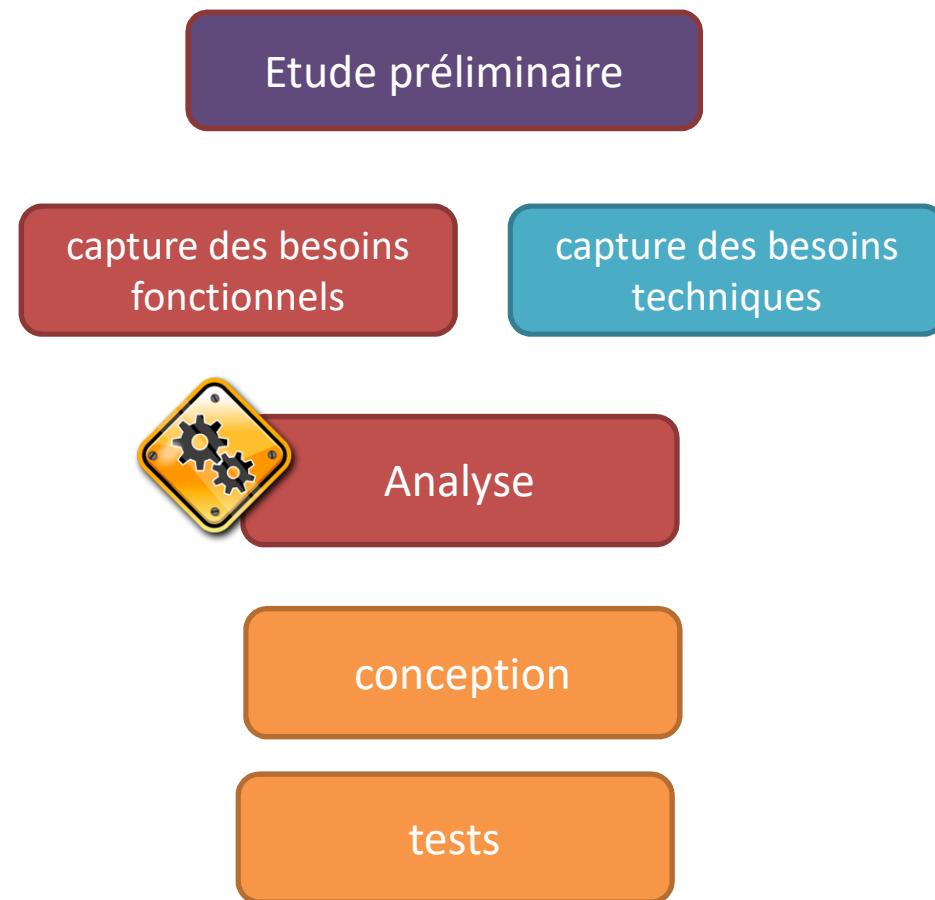
diag. de déploiement



diag. de packages

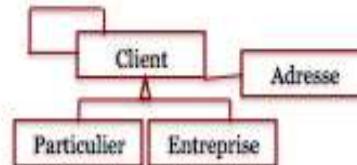
- La description du style d'architecture

Processus unifié : quelques grandes étapes



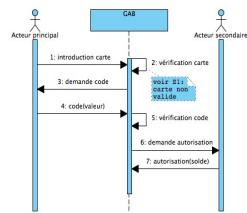
Analyse

- Développement du modèle statique

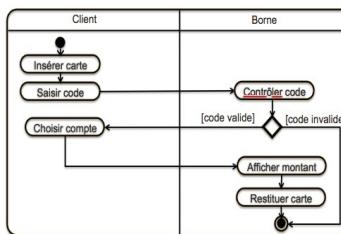


diag. de classes

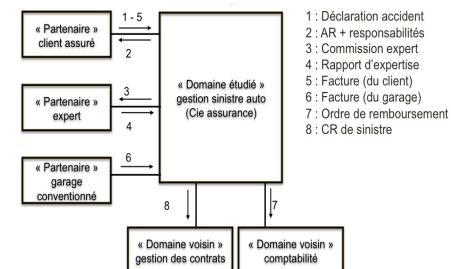
- Développement du modèle dynamique



diag. de séquence



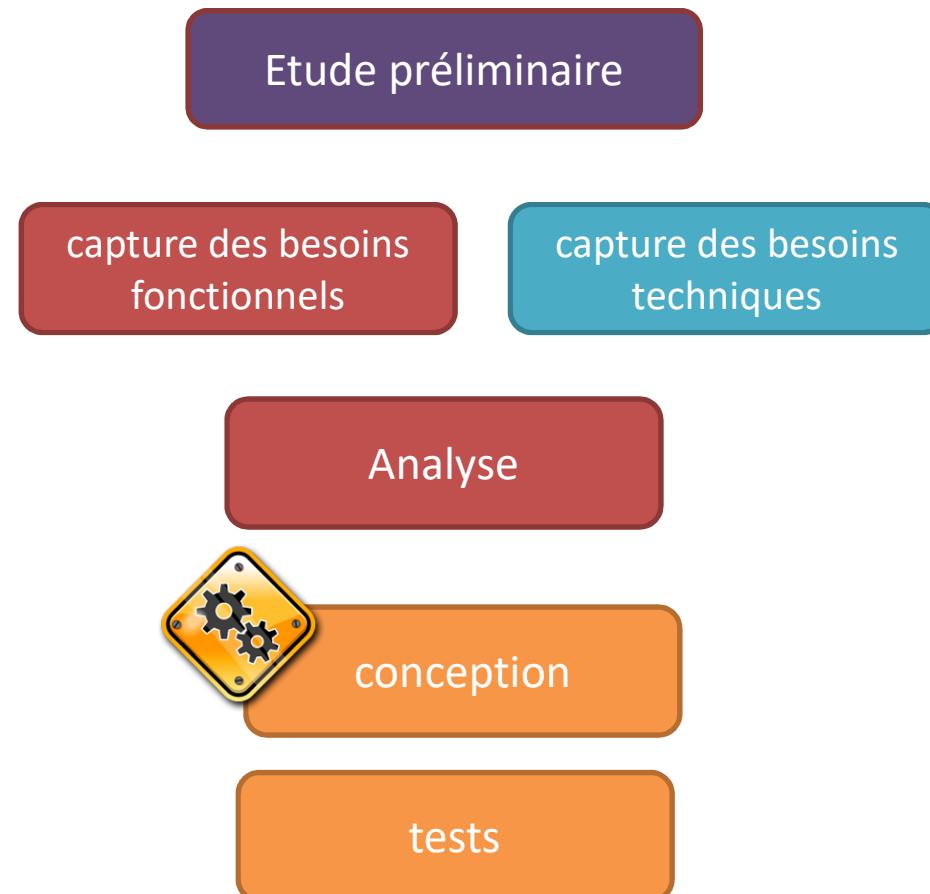
diag. d'activités ou d'états



diag. de communication

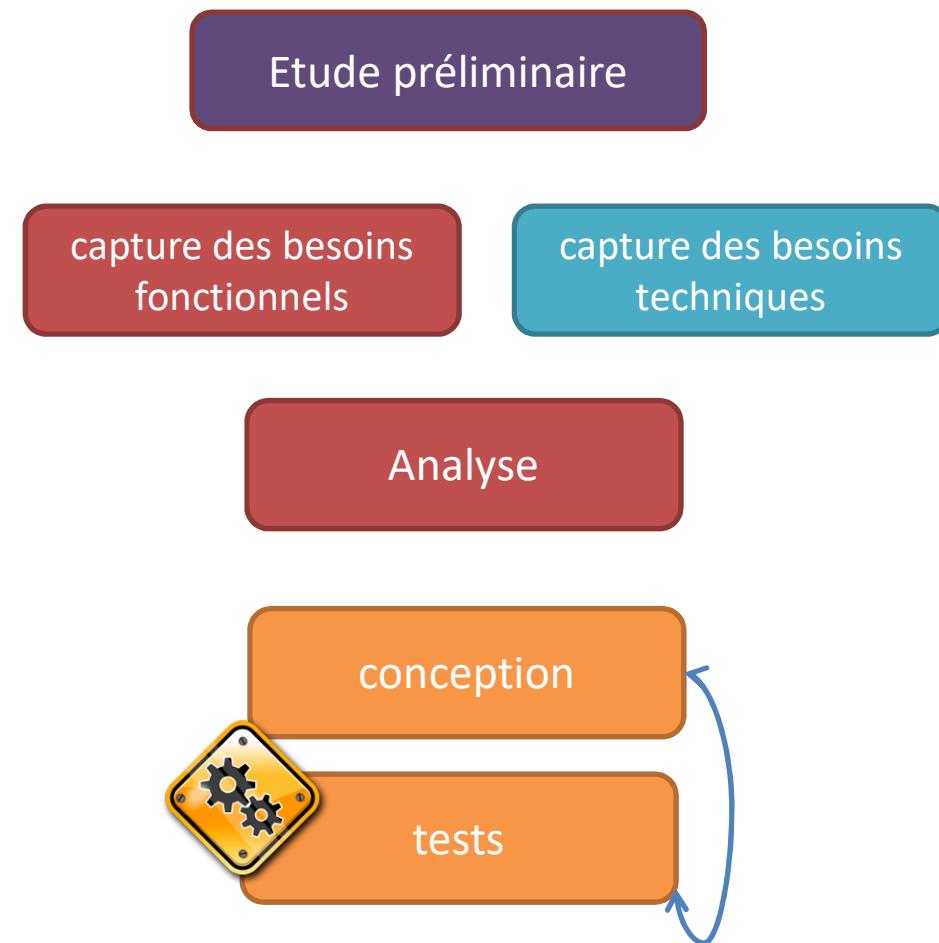
- Rapprochement des modèles

Processus unifié : quelques grandes étapes



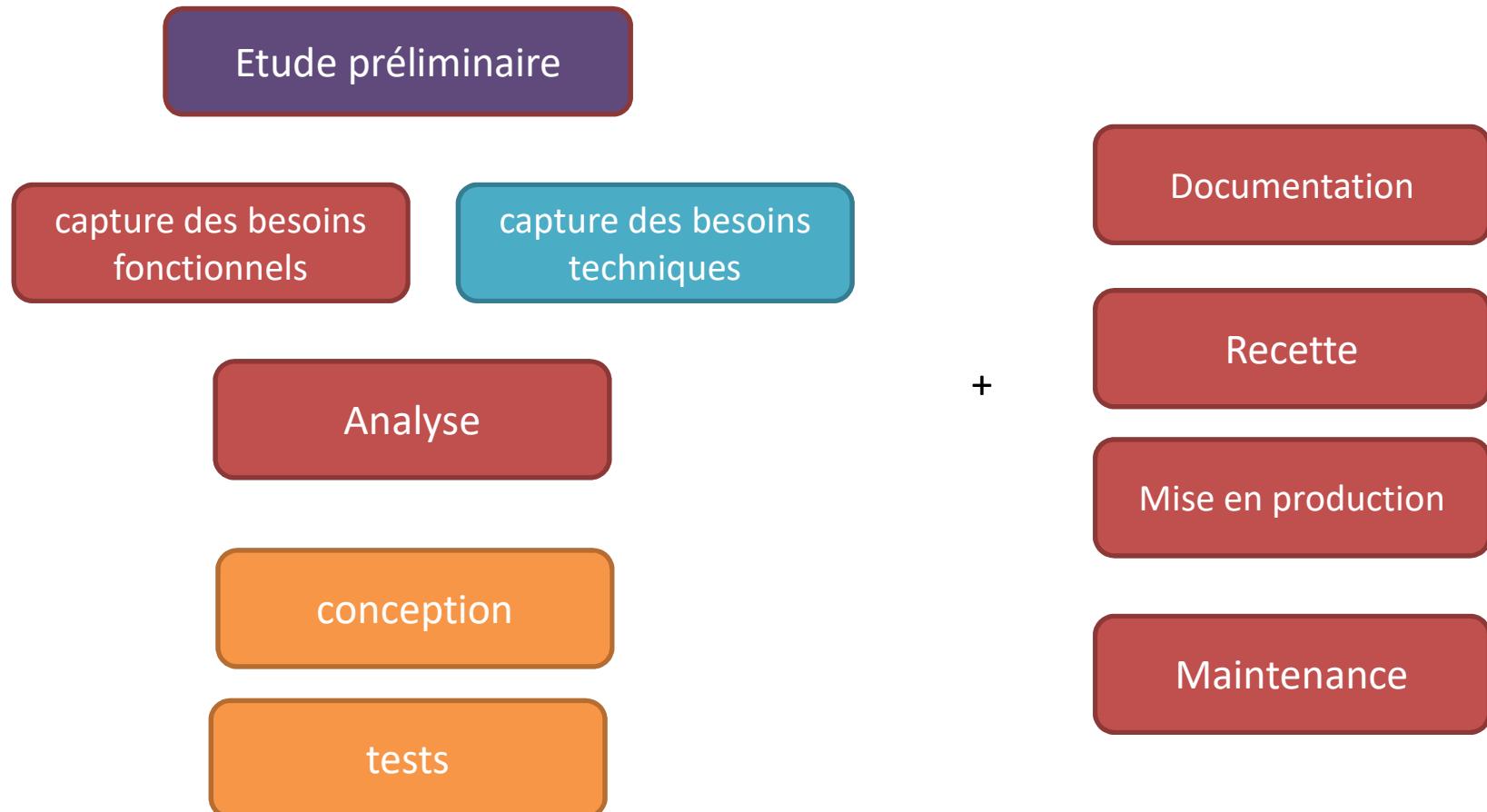
- Dans le cas d'un logiciel,
 - Des spécifications au prototype (éventuelle utilisation d'un atelier GL).
 - A partir de là, le langage de prog. intervient mais ce n'était pas le cas dans les étapes précédentes

Processus uniifié : quelques grandes étapes



- Tests unitaires, intégration, recette.
- Sont menés, entre autres, à partir des spécifications.

Processus unifié : quelques grandes étapes



Architecture logicielle vs. architecture SI

L'architecture de SI se différencie de l'architecture logicielle par les concepts manipulés :

- modules logiciels, classes, composants pour l'architecture logicielle ;
- modules applicatifs, référentiels, flux pour l'architecture de SI.

En outre, l'architecte logiciel doit concevoir une architecture implémentant des cas d'utilisations centrés sur une application, alors que l'architecte de SI devra concevoir une architecture implémentant des cas d'utilisation transverses à différentes applications... Si la démarche est globalement la même (définir la structure d'un système, ses composants, sa dynamique...), architecture logicielle et architecture de SI opèrent cependant à des niveaux différents de granularité et d'abstraction.

→ Gardez en tête que vous allez proposer un SI et pas un logiciel !



How the customer explained it



How the Project Leader
understood it



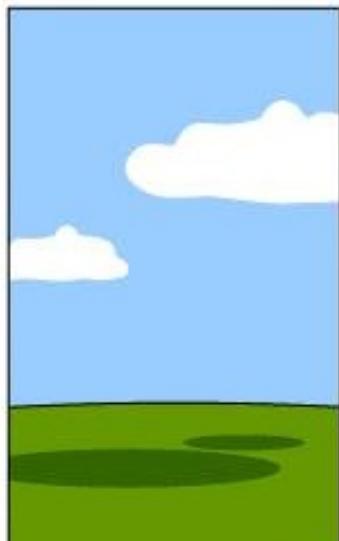
How the Analyst designed it



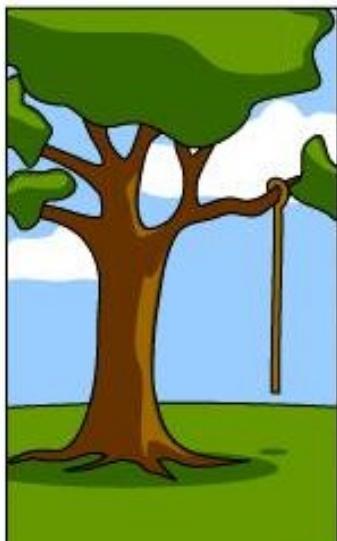
How the Programmer wrote it



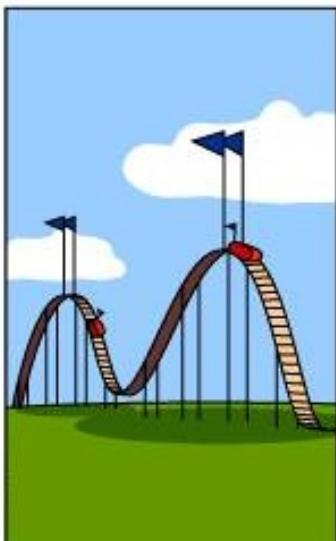
How the Business Consultant
described it



How the project was
documented

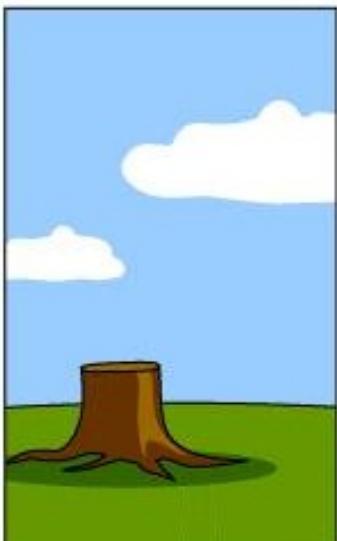


What operations installed



How the customer was billed

Introduction à UML



How it was supported



What the customer really
needed

Bibliographie

- P. Roques, F. Vallée. *UML 2 en action, 4^{ème} édition.* Ed. Eyrolles, 2009.
- P. Roques. *UML 2 par la pratique, 6^{ème} édition.* Ed. Eyrolles, 2008.
- Cours UML 2 de Jacques Callot, Laurent Audibert, Emmanuel Renaux, Laurent Piechocki, Tibo Delore.
- Site Web de Michel Volle.
- H.-E. Eriksson, M. Penker, B. Lyons, et D. Fado. *Uml 2 Toolkit.* Ed. John Wiley & Sons Inc., 2003.
- J. Gabay et D. Gabay. *UML 2 Analyse et Conception - Mise en Oeuvre Guidée avec Etudes de Cas.* Ed. Dunod, 2008.

A vous de jouer

Le domaine des SI est un champ vaste sur lequel s'ouvre encore des tas de travaux à découvrir et à explorer :

- différentes méthodes de capture des besoins,
- conduite du changement,
- urbanisation,
- capitalisation et gestion des connaissances,
- gestion de la qualité des données,
- les processus qualité liés aux SI (ISO, AFNOR et cie),
- SI coopératifs, distribués,
- retro ingénierie,
- etc. etc. etc. etc. etc.

→ Allez de vous-mêmes à la pêche aux informations (livres, livres blancs, sites web, etc.). Il s'agit là d'un champ passionnant !

