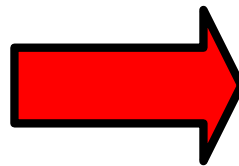
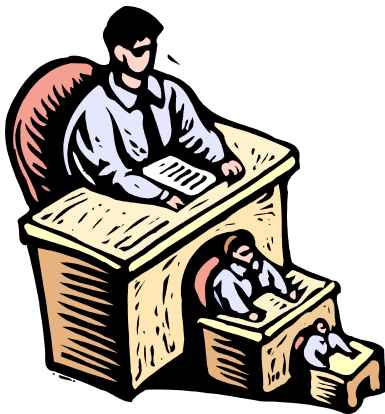


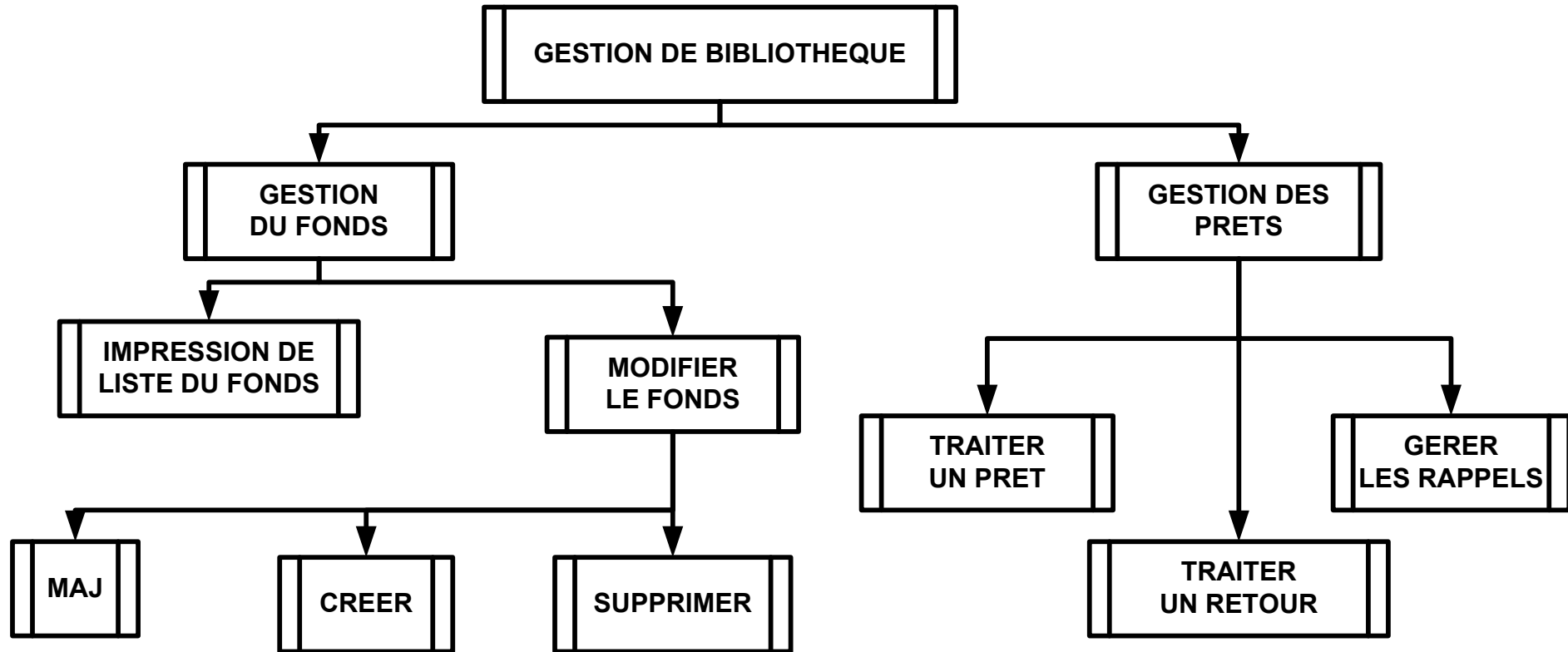
# UML - MERISE

ARCNAM - NPDC  
ANNEE 2004 - 2005



## LA DECOUPE FONCTIONNELLE

La découpe fonctionnelle d'un problème informatique ressort d'une démarche très intuitive  
Exemple de la gestion d'une bibliothèque de prêt



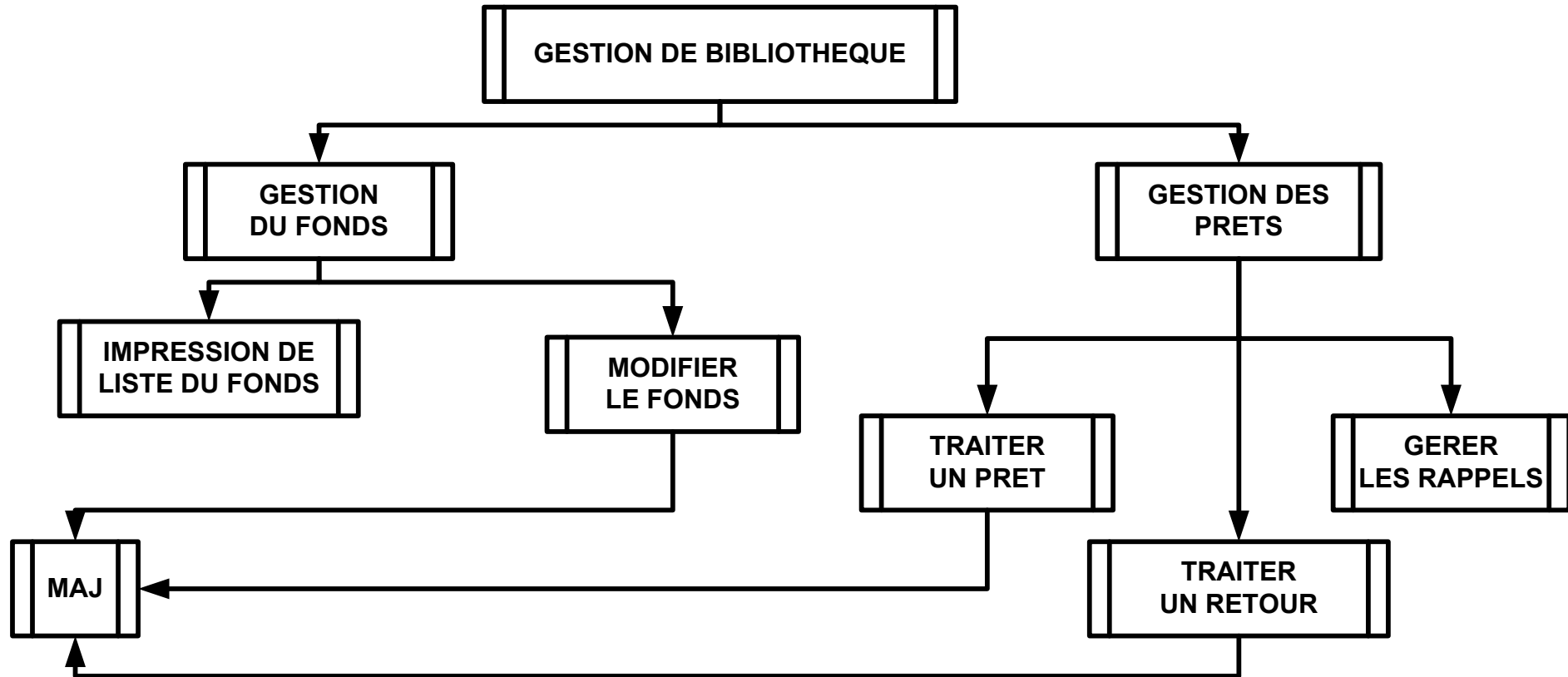
A partir de cette approche intuitive issue d'une EBG, MERISE permet de "modéliser" la CONCEPTION au travers de trois modèles:

- Le MCC pour modéliser l'ensemble des messages échangés par les ACTEURS,
- Le MCD pour modéliser les DONNEES nécessaires après étude des messages,
- Le MCT pour modéliser l'approche fonctionnelle en FONCTIONS de traitement.

L'affinage se faisant au travers des MLD ( gestion des ATTRIBUTS et de leur FORMATS), des MOT ( compléments ORGANISATIONNELS des MCT ). L'ensemble sous tendu par l'élaboration des RG, des RO et des RP.

## LA DECOUPE FONCTIONNELLE "FACTORISEE"

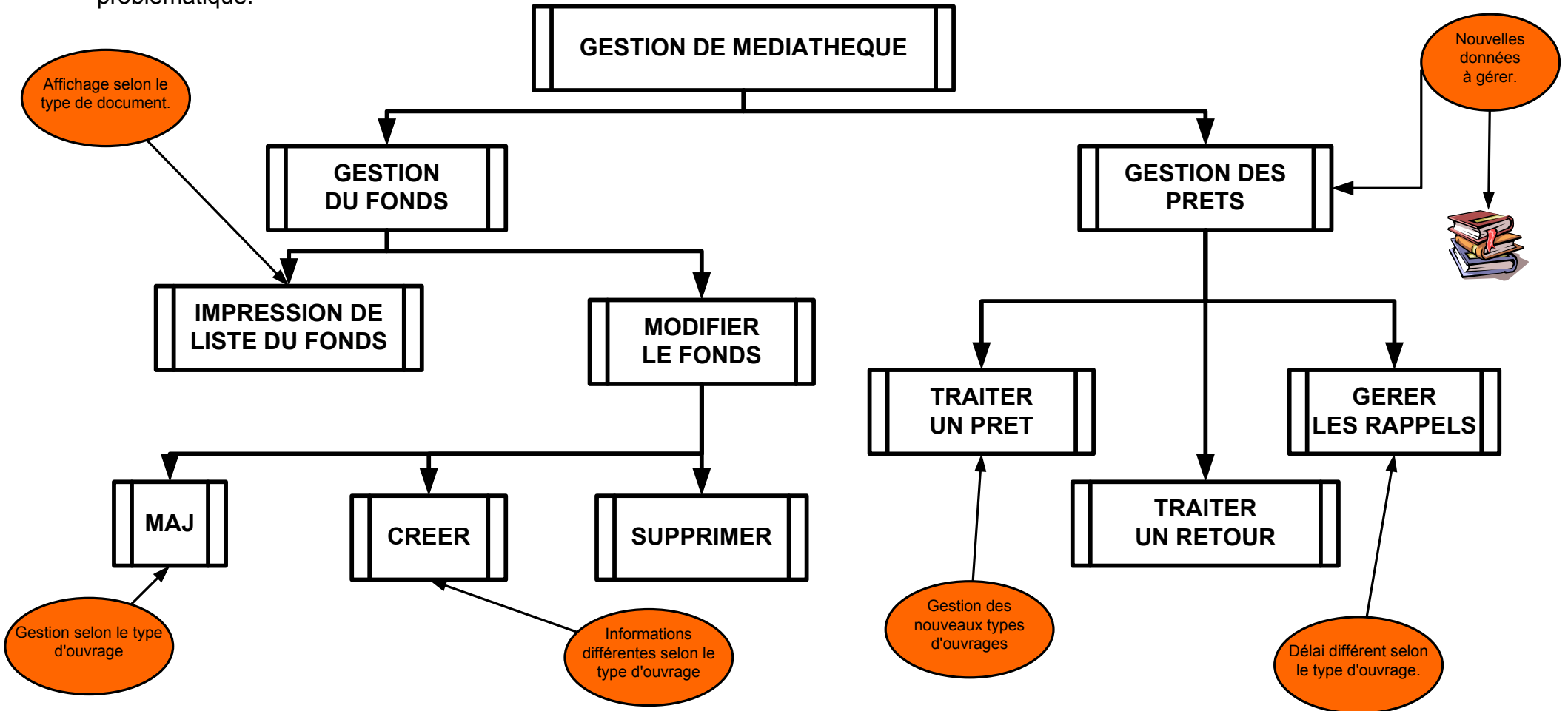
Une découpe fonctionnelle "intelligente" d'un problème informatique consiste à "factoriser" certains comportements communs du logiciel. Exemple de la gestion d'une bibliothèque de prêt



La fonction "Mise A Jour est dite factorisée si elle permet, par un paramétrage, de créer, modifier et supprimer des informations. Ce point est fondamental en conception dite Merisienne.

# MAINTENANCE COMPLEXE EN CAS D'EVOLUTION FONCTIONNELLE

Factoriser les éléments fonctionnels peu poser des problèmes en cas de demandes d'évolutions. En effet, les fonctions sont devenues interdépendantes. La multiplication des points de maintenance rend la mise au point des évolutions laborieuse et très problématique.



Compte tenu de ce qui est décrit, est-ce que la solution ne réside pas dans le fameux concept ( très MERISIEN ) de la séparation des DONNEES et des TRAITEMENTS cher au modèle Relationnel ?

## LA SEPARATION DES DONNEES ET DES TRAITEMENTS PEUT ETRE UN PIEGE A INFORMATIENS

Réaliser une maintenance EVOLUTIVE pour passer d'une gestion de BIBLIOTHEQUE à une gestion de MEDIATHEQUE afin de tenir compte de nouveaux types d'ouvrages ( Cassettes vidéos, CD-ROM, etc.. ) va nécessiter:

- De faire évoluer les structures de données manipulées par les fonctions
- D'adapter les traitements qui ne manipulaient à l'origine qu'un seul type de documents ( les livres )

Deux améliorations seront à apporter au logiciel initial. Amélioration qui auraient pu être prévues dès la phase de conception initiale si l'informaticien a pensé ( ? ) lors de cette phase, à la maintenance évolutive possible !

- Rassembler les valeurs qui caractérisent un type d'ouvrage
- Centraliser les traitements associés à un type d'ouvrage et ce auprès du type d'ouvrage

Par exemple sur le premier point, un délai avant édition d'un rappel est bien une caractéristique propre à tous les ouvrages.  
( Avez vous dit héritage ? )



Par exemple sur le deuxième point, si l'on créait une fonction rassemblant la structure de données décrivant les ouvrages et la fonction de calcul du délai avant rappel ? En cas de nouveau type d'ouvrage, nous n'aurions alors qu'un seul bout de code à modifier !  
( Avez vous dit encapsulation ? )

### CONCLUSION

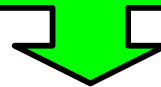
Les modifications apportées nous ont amené à transformer, ce qui était à l'origine, une structure de données, manipulée par des traitements, en une entité autonome qui regroupe un ensemble de propriétés cohérentes et de traitement associés.

**BRAVO ! NOUS VENONS DE FABRIQUER UN OBJET!**

## LES CONCEPTS ORIENTES OBJETS DEFINITIONS

### OBJET

- Un OBJET est une entité aux frontières précises qui possède une IDENTITE, un ETAT et un COMPORTEMENT
- Un OBJET est caractérisé par un ensemble d'ATTRIBUTS ( INFORMATIONS )
- Un OBJET est caractérisé par un ensemble d'OPERATIONS ( METHODES ) qui en définissent les COMPORTEMENTS
- Un OBJET est une instance de CLASSE ( occurrence d'un TYPE abstrait )
- Une CLASSE est un type de DONNEES abstrait, caractérisé par des PROPRIETES ( ATTRIBUTS et METHODES ) communes à des objets et permettant de créer des OBJETS instanciés possédant ces propriétés.



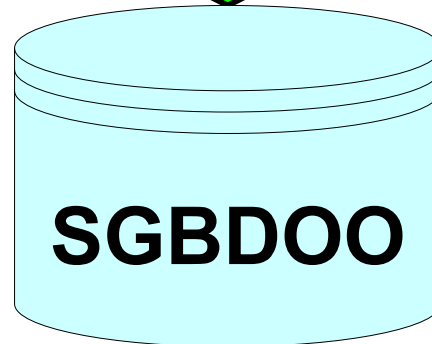
CLASSE: VOITURE	
ATTRIBUTS	METHODES
Marque, Couleur, Immatriculation, Puissance, Type	Démarrer, Conduire, Arrêter, Réviser, Vendre

OBJET INSTANCIE: TWINGO	
ATTRIBUTS	METHODES
Renault, Jaune, 4904 ZT 59, 4 CV, deux portes	Démarrer, Conduire, Arrêter, Réviser, Vendre

## LES CONCEPTS ORIENTES OBJETS DEFINITIONS

### ENCAPSULATION

- Concept permettant de "masquer" les détails d'IMPLEMENTATION d'un OBJET, en définissant une INTERFACE
- L'INTERFACE est une vue externe d'un OBJET, elle définit les SERVICES accessibles aux utilisateurs de l'objet. ( Création et Mises à Jours de données par exemple )
- L'ENCAPSULATION facilite l'évolution d'une APPLICATION car elle stabilise l'utilisation des OBJETS. Par exemple, on peut modifier l'implémentation des ATTRIBUTS d'un OBJET sans modifier son INTERFACE.
- L'ENCAPSULATION garantit l'INTEGRITE des DONNEES car elle permet d'interdire l'ACCES direct aux ATTRIBUTS des OBJETS.



## LES CONCEPTS ORIENTES OBJETS DEFINITIONS

### HERITAGE ET POLYMORPHISME

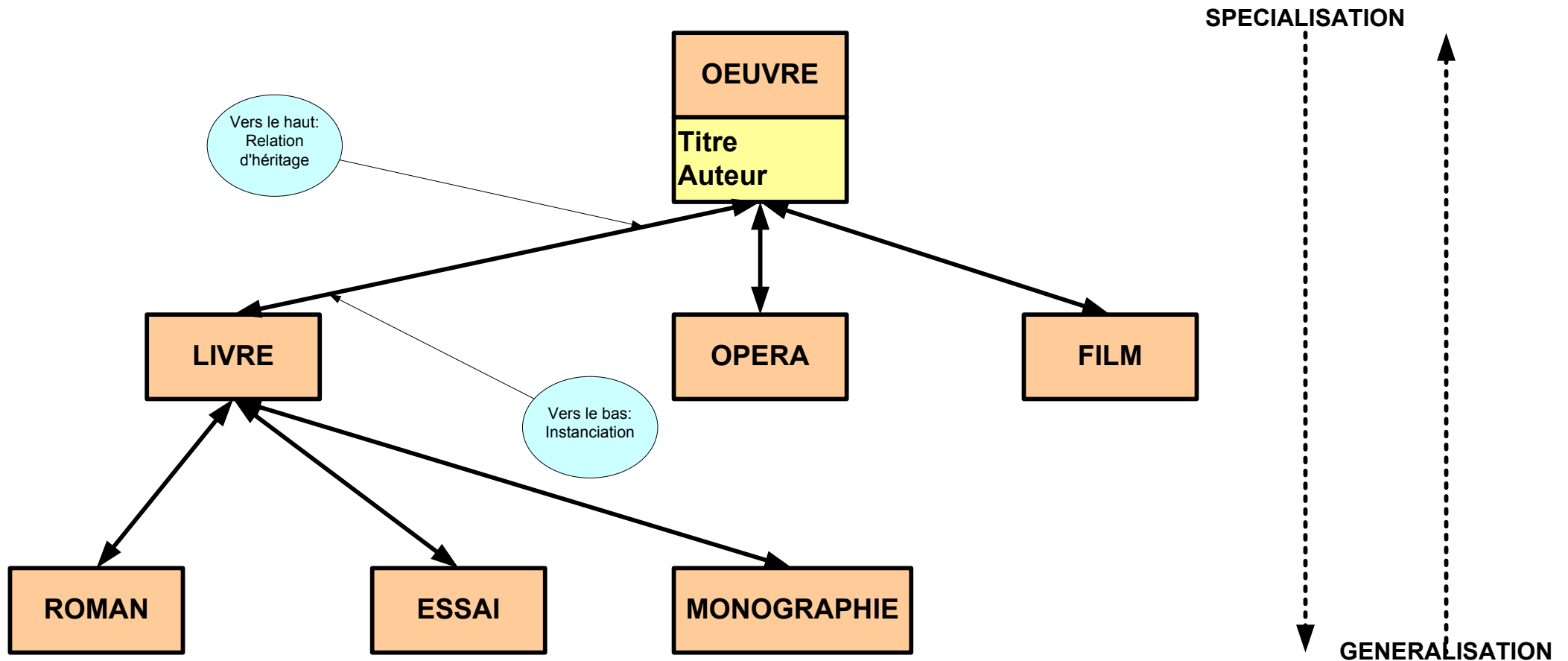
- L'HERITAGE est un mécanisme de transmission des PROPRIETES d'une CLASSE ( ATTRIBUTS et METHODES ) vers une SOUS CLASSE.
- Une CLASSE peut être spécialisée en d'autres CLASSES afin d'y ajouter des CARACTERISTIQUES spécifiques ou d'en adapter certaines.
- Plusieurs CLASSES peuvent être généralisées en une CLASSE qui les factorise afin de regrouper les CARACTERISTIQUES communes d'un ensemble de CLASSES.
- La SPECIALISATION et la GENERALISATION permettent de construire des HIERARCHIES de CLASSES. L'HERITAGE peut être SIMPLE ou MULTIPLE.
- L'HERITAGE évite la Duplication et encourage la Réutilisation
- Le POLYMORPHISME représente la faculté d'une METHODE à pouvoir s'appliquer à des OBJETS de CLASSES différentes.
- le POLYMORPHISME augmente la GENERICITE du code.

Voir exemples  
pages suivantes



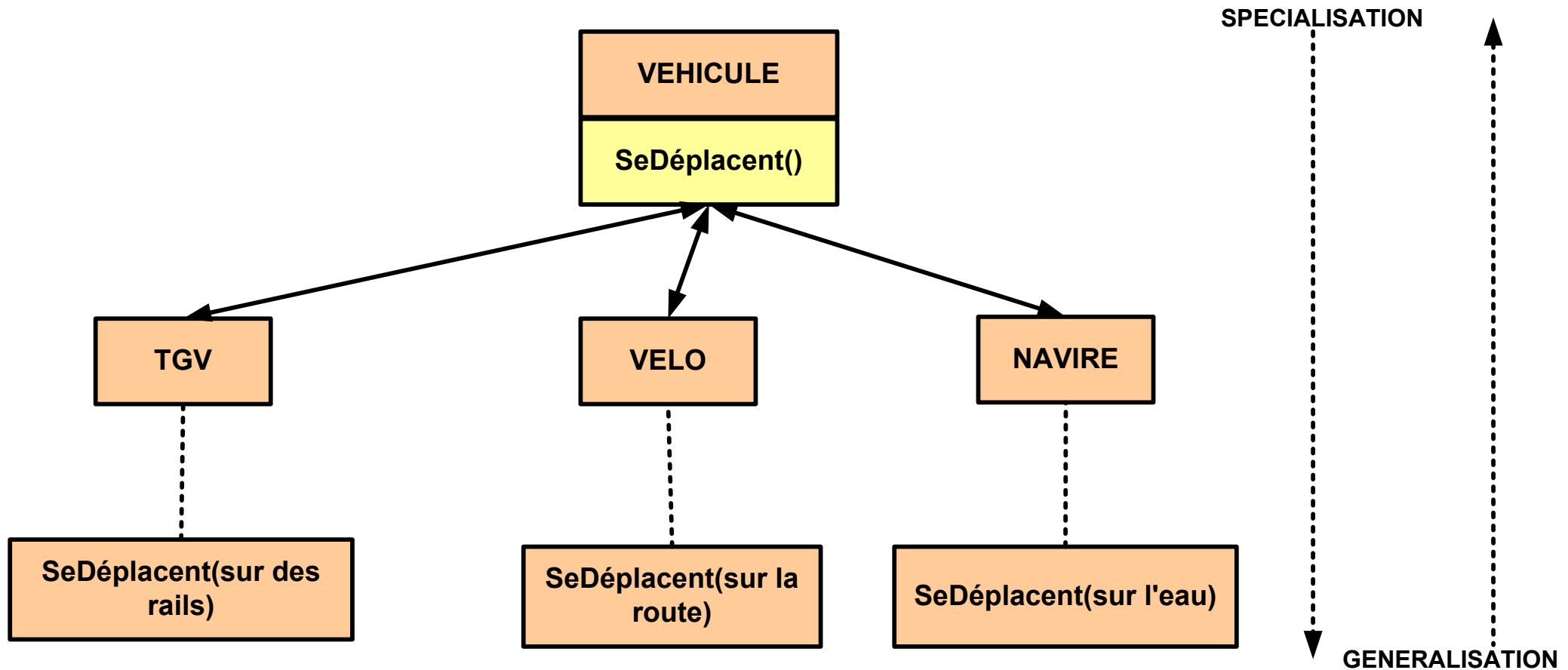
# LES CONCEPTS ORIENTES OBJETS DEFINITIONS

## HERITAGE ( HIERARCHIE DE CLASSES )



# LES CONCEPTS ORIENTES OBJETS DEFINITIONS

## POLYMORPHISME



## L'APPROCHE UML ( Unified Modeling Language ).

### AGREGATION

- Il s'agit d'une relation entre deux CLASSES, spécifiant que les OBJETS d'une CLASSE sont des composants de l'autre CLASSE. Une relation d'AGREGATION permet donc de définir des OBJETS composés d'autres OBJETS.
- L'AGREGATION permet d'assembler des OBJETS de base, afin de construire des OBJETS plus complexes

Exemple

**OBJET : GUERIDON**

OBJET: PIED

OBJET: PIED

OBJET: PIED

OBJET: PLATEAU



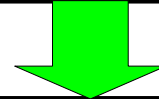
Un Guéridon est composé d'un plateau et de trois pieds

## L'APPROCHE UML ( Unified Modeling Language ).

### MODELE, METHODE ou LANGAGE ?

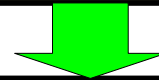
➤ **UML n'est pas une METHODE et/ou un PROCESSUS.**

UML est un LANGAGE et si l'on parle de méthode UML c'est un abus de langage! La différence est qu'une METHODE repose sur une THEORIE ( systémique pour MERISE ) et propose aussi un PROCESSUS qui régit notamment l'enchaînement des activités d'une entreprise. Néanmoins, UML permettra de modéliser les activités de l'entreprise



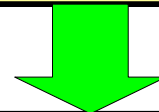
➤ **UML est un LANGAGE pseudo-formel**

UML est un LANGAGE fondé sur un META-MODELE qui définit la SYNTAXE ( Concepts et forme des éléments du langage ) de modélisation et la SEMANTIQUE qui définit la définition et le sens des éléments du langage. UML propose une NOTATION graphique qui permet de représenter les éléments du META-MODELE.



➤ **UML cadre l'ANALYSE OBJET**

Par les différentes vues complémentaires d'un SYSTEME et par plusieurs niveaux d'ABSTRACTION qui permettent de mieux contrôler la complexité dans l'élaboration de solutions objets.



➤ **UML est un support de COMMUNICATION.**

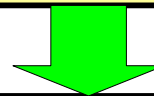
UML est un LANGAGE universel indépendant des langages d'implémentation et des langages métiers. L'aspect formel de ses notations permet de limiter les ambiguïtés et les incompréhensions. Son aspect graphique permet d'exprimer visuellement des solutions.

## L'APPROCHE UML ( Unified Modeling Language ).

### QU'EST-CE Q'UN MODELE?

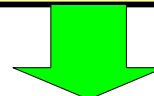
➤ **Un MODELE est une abstraction de la réalité.**

L'ABSTRACTION est un des piliers de l'approche OBJET. Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une ENTITE en vue d'une utilisation précise dans le cadre du travail à réaliser.



➤ **Un MODELE est une vue SUBJECTIVE mais PERTINENTE de la réalité.**

Un MODELE n'est pas la REALITE, mais une vue TRES subjective de la REALITE. Il définit une FRONTIERE entre la REALITE et la PERSPECTIVE de l'observateur. Le MODELE doit refléter les aspects importants de la REALITE du point de vue du système à réaliser.



➤ **Exemple de MODELES**

- ✓ Modèle d'Observation météorologique: permettant, à partie de données d'observations, de prévoir les conditions météo en un futur proche,
- ✓ Modèle Mathématique: Permettant de prévoir certains phénomènes. Par exemple l'orbite de certaines planètes par rapport à d'autres ( modèle Newtonnien à opposer au modèle Einsteinnien ? )
- ✓ Modèle Economique: Permettant de prévoir l'évolution des cours de la bourse.
- ✓ Etc...

Le Maître mot, en termes de modèles est:PREVOIR

# L'APPROCHE UML ( Unified Modeling Language ).

## COMMENT MODELISER AVEC UML?

➤ **UML est un LANGAGE qui permet de représenter des MODELES , mais qui ne définit pas le PROCESSUS d'élaboration desdits MODELES.**

Cependant, dans le cas d'une modélisation d'une application informatique, il est préconisé d'utiliser une démarche:

- ✓ Itérative et Incrémentale
- ✓ Guidée par les utilisateurs futurs ( la Maîtrise d'Ouvrage ).
- ✓ centrée sur l'architecture logicielle.



### Une démarche ITERATIVE et INCREMENTALE?

L'idée est que pour MODELISER un système, il vaut mieux s'y prendre en plusieurs fois en affinant l'analyse étape par étape. Démarche applicable au cycle de vie du développement ( 3V ou Spirale ). La modélisation et la réalisation OO se prête bien à cette démarche. Ce qui permet de prendre en compte les modifications d'ENVIRONNEMENT chers aux adeptes de la systémique.



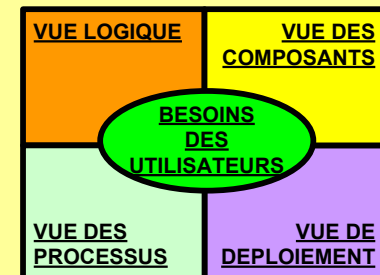
### Une démarche pilotée par les UTILISATEURS?

Le but de tout projet informatique étant l'ADEQUATION aux besoins, quoi de plus naturel que de chercher à impliquer les utilisateurs ? ce qui suppose que lesdits utilisateurs soient à même de comprendre la signification des modèles élaborés et de visualiser très rapidement le résultat de ces modèles ( Maquettes statiques et dynamiques, par exemple ).



### Une démarche centrée sur l'ARCHITECTURE logicielle ?

Cette architecture doit décrire les grands choix stratégiques permettant de définir la qualité du logiciel ( adaptabilité, fiabilité, performances, etc.. Cette architecture peut s'élaborer au travers de quatre grandes VUES du futur système:



## L'APPROCHE UML ( Unified Modeling Language ).

### L'APPROCHE PAR LES VUES

#### La vue LOGIQUE.

Il s'agit d'une vue de haut niveau concentrée sur l'ABSTRACTION et l'ENCAPSULATION. Elle modélise les éléments et mécanismes principaux du système. Elle identifie les éléments du domaine ainsi que les relations et interactions entre ces éléments. ( Eléments du DOMAINE liés aux METIERS gagnant à être réutilisés ). Cette VUE doit aussi permettre de répartir les tâches entre les équipes, de regrouper ce qui peut être générique et d'isoler ce qui est propre à une version donnée d'un système.

#### La vue des COMPOSANTS.

Il s'agit d'une vue de bas niveau qui montre et recense:

- ✓ L'allocation des éléments de modélisation dans les modules ( fichiers sources, bases de données, modules exécutables, bibliothèques dynamiques, etc... )
- ✓ L'organisation des composants, c'est à dire la distribution du code en gestion de configuration, les dépendances entre composants, etc...
- ✓ Les contraintes de développement ainsi que les gestions d'interfaces.

#### La vue des PROCESSUS.

Elle montre, en environnements multi-tâches:

- ✓ La décomposition du système en termes de processus; Les interactions entre processus ( communication ), la synchronisation et la communication des activités parallèles ( threads )
- ✓

#### La vue de DEPLOIEMENT.

Très importante dans les environnements distribués ( architectures 3 1/3 par exemple ), elle décrit les ressources matérielles et la répartition du logiciel dans ces ressources.( Dispositions et natures physiques des matériels, implantation des modules principaux sur les noeuds du réseau, exigences en termes de performances ).

#### La vue des BESOINS UTILISATEURS.

Le nom exact de cette vue est : " cas d'utilisations"; elle guide toutes les autres.Elle définit les besoins du Client du futur système et conduit à la définition d'un modèle d'architecture pertinent et cohérent.

# L'APPROCHE UML ( Unified Modeling Language ).

## LES DIFFERENTS TYPES DE DIAGRAMMES UML

**IL EXISTE CINQ VUES STATIQUES DU SYSTEME:**

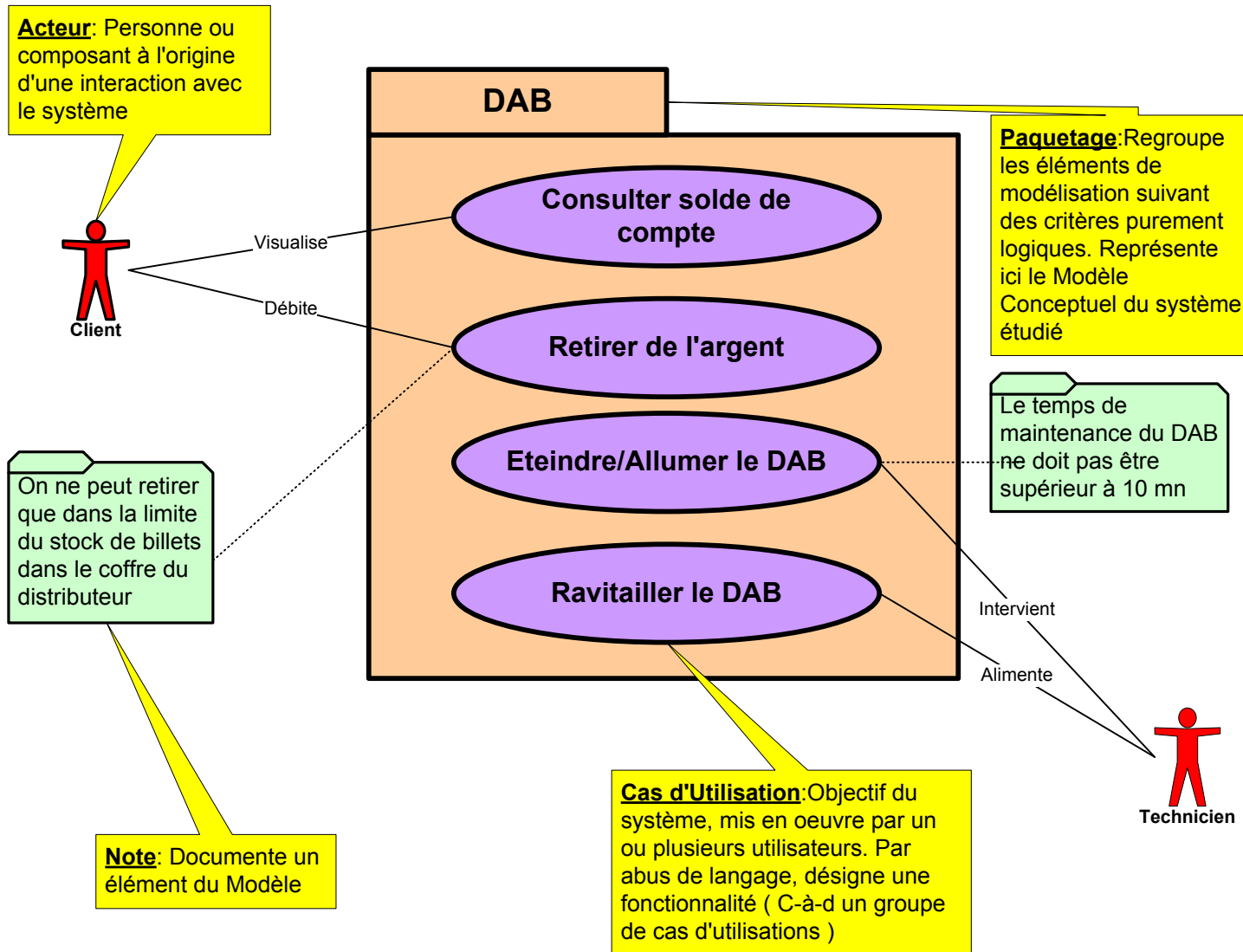
- Diagrammes de CAS d'UTILISATIONS**
- Diagrammes d'OBJETS**
- Diagrammes de CLASSES**
- Diagrammes de COMPOSANTS**
- Diagrammes de DEPLOIEMENT**

**IL EXISTE QUATRE VUES DYNAMIQUES DU SYSTEME:**

- Diagrammes de COLLABORATION**
- Diagrammes de SEQUENCE**
- Diagrammes d'ETATS-TRANSITIONS**
- Diagrammes d'ACTIVITE**

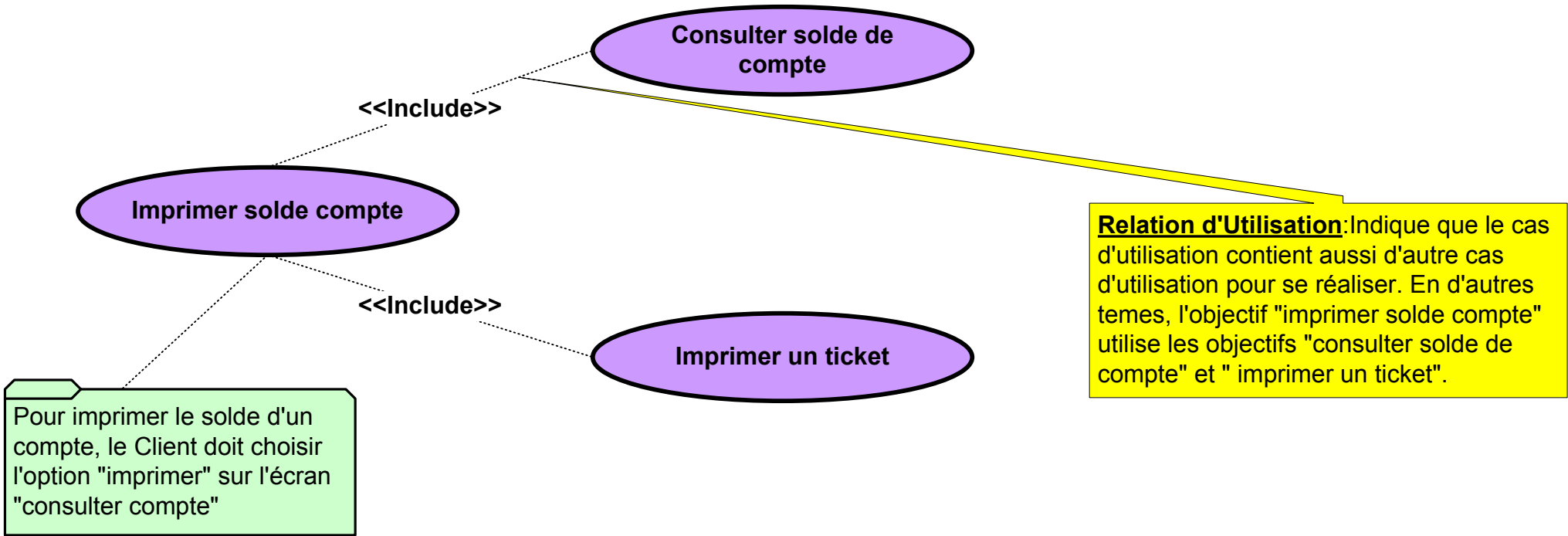
	Vue des Cas D'utilisation	Vue Logique	Vue des Composants	Vue des Processus	Vue de Déploiement
Diagramme de cas d'utilisation	Acteurs Cas d'utilisation				
Diagramme d'Objets	Acteurs Objets Liens	Acteurs Classes d'Objets Liens			
Diagramme de Collaboration	Acteurs Objets Liens Messages	Acteurs Classes Objets Liens		Classes Objets Liens	
Diagramme de séquence	Acteurs Objets Messages	Acteurs Objets Messages			
Diagramme de Classes		Acteurs Classes Paquetages Relations		Objets Messages	
Diagramme d'Etats-Transitions	Etats Transitions	Etats Transitions		Etats Transitions	
Diagramme d'Activités	Activités Transitions	Activités Transitions		Activités Transitions	
Diagramme de Composants			Composants	Composants	Composants
Diagramme de Déploiement					Nœuds Liens

# LES VUES STATIQUES D'UML: LES CAS D'UTILISATION



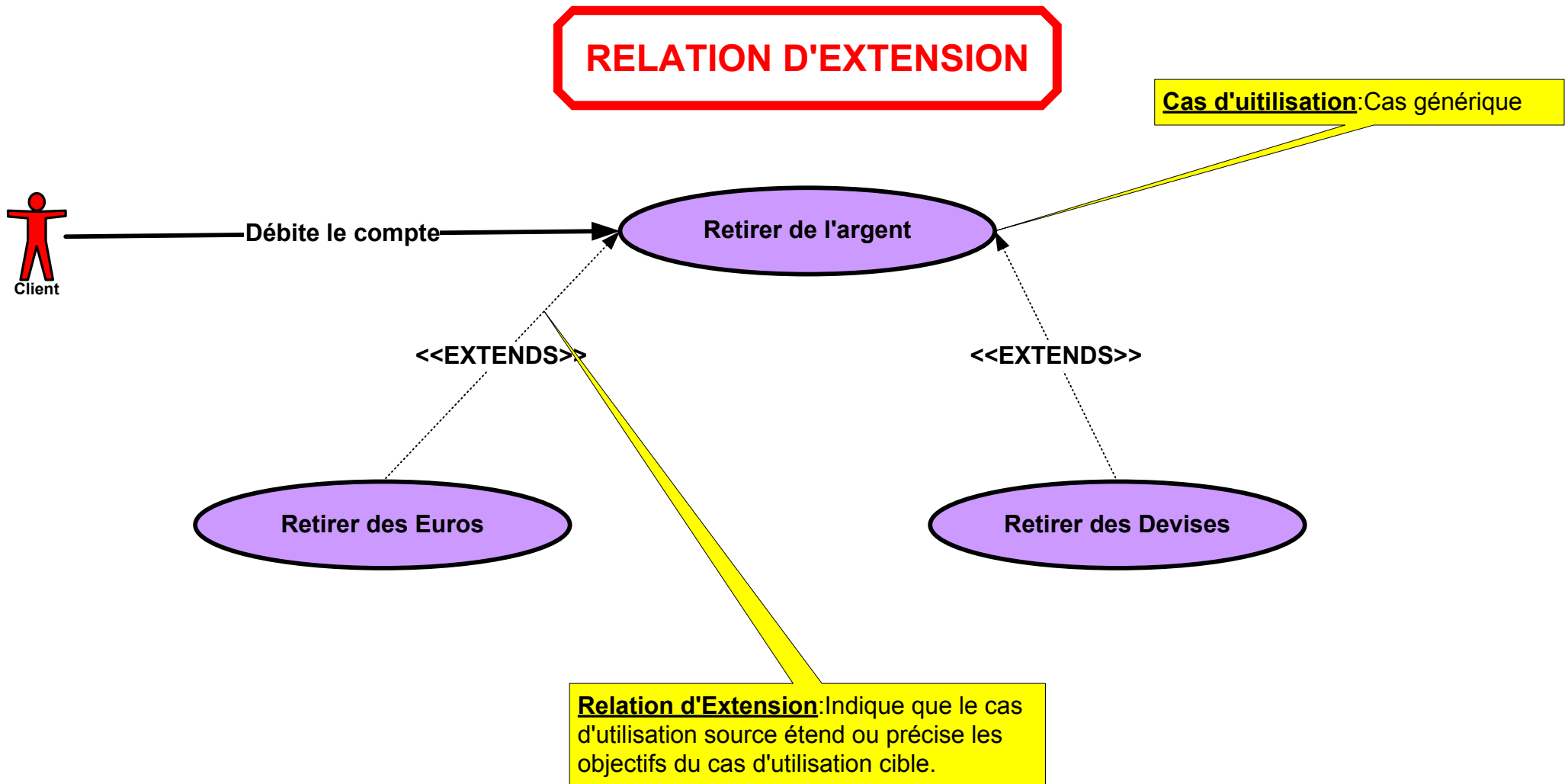
# LES VUES STATIQUES D'UML: LES CAS D'UTILISATION

## RELATION D'UTILISATION



# LES VUES STATIQUES D'UML: LES CAS D'UTILISATION

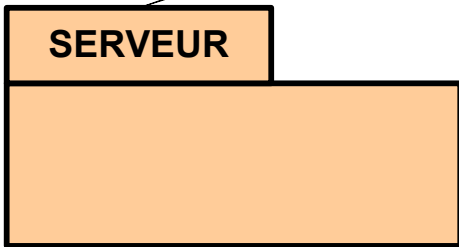
## RELATION D'EXTENSION



# LES VUES STATIQUES D'UML: LES CAS D'UTILISATION

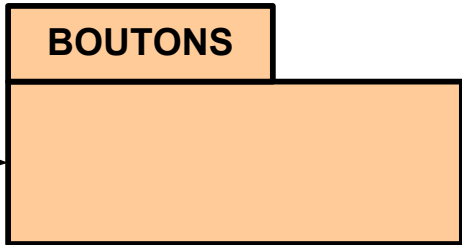
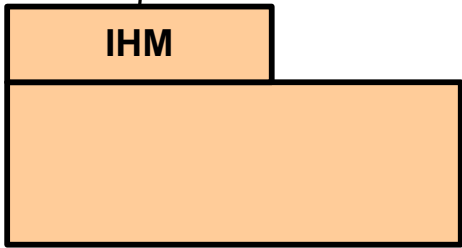
## LES PAQUETAGES

**Nom symbolique:** du paquetage



**Héritage entre paquetages:** Il existe ( au moins ) un élément du package source qui spécialise ( au moins ) un élément du package cible ).

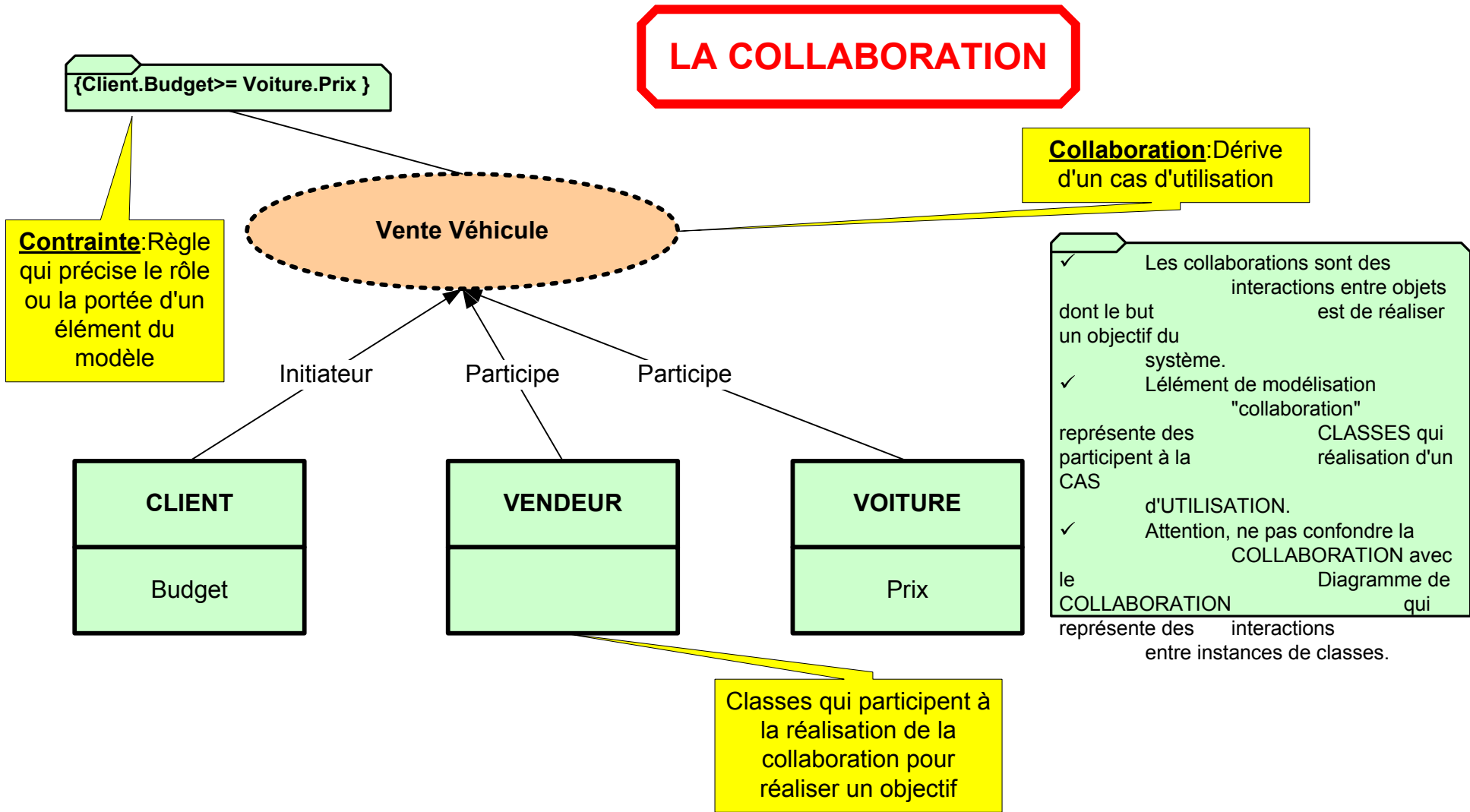
**Dépendance entre paquetages:** Il existe ( au moins ) un élément du package source qui utilise ( au moins ) un élément du package cible ).



- ✓ Les paquetages sont des éléments d'organisation des modèles.
- ✓ Ils regroupent des éléments de Modélisation selon des critères logiques.
- ✓ Ils permettent d'encapsuler des éléments de modélisation ( ils utilisent des interfaces )
- ✓ Ils permettent de structurer un système en catégories ( vues logiques ) et sous systèmes ( vues composants )
- ✓ Ils servent de "briques" de base dans la construction d'une architecture.
- ✓ Ils représentent le bon niveau de granularité pour la réutilisation
- ✓ les paquetages sont aussi des espaces de nomination.

# LES VUES STATIQUES D'UML: LES OBJETS

## LA COLLABORATION



**Contrainte:** Règle qui précise le rôle ou la portée d'un élément du modèle

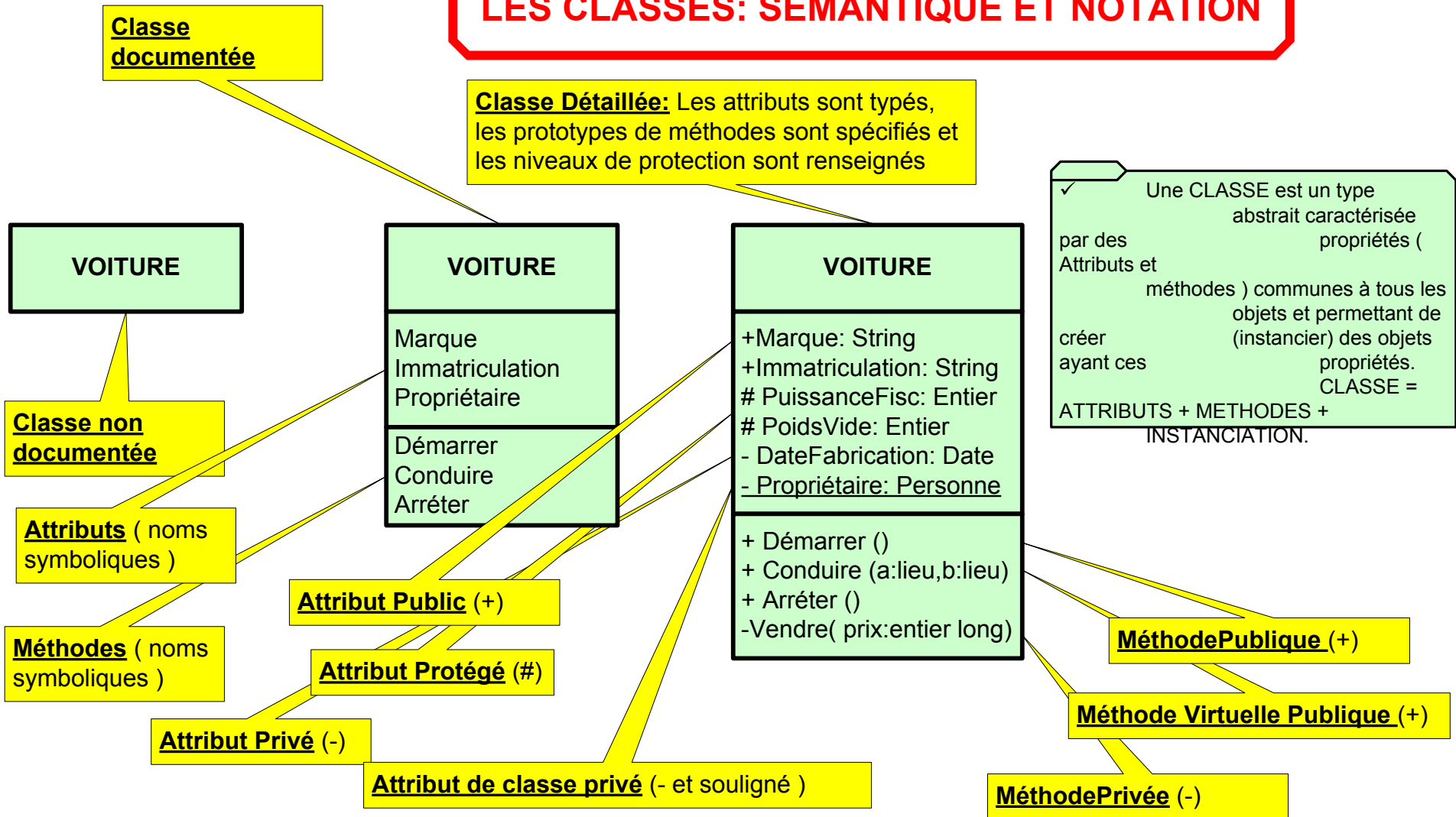
**Collaboration:** Dérive d'un cas d'utilisation

✓ Les collaborations sont des interactions entre objets dont le but est de réaliser un objectif du système.  
 ✓ L'élément de modélisation "collaboration" représente des CLASSES qui participent à la réalisation d'un CAS d'UTILISATION.  
 ✓ Attention, ne pas confondre la COLLABORATION avec le Diagramme de COLLABORATION qui représente des interactions entre instances de classes.

Classes qui participent à la réalisation de la collaboration pour réaliser un objectif

# LES VUES STATIQUES D'UML: LES OBJETS

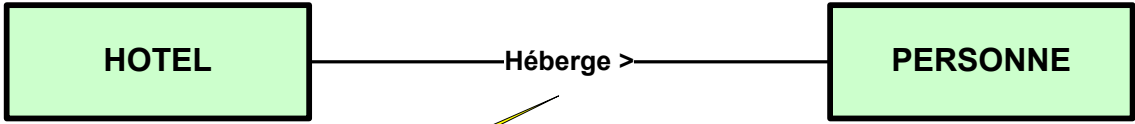
## LES CLASSES: SEMANTIQUE ET NOTATION



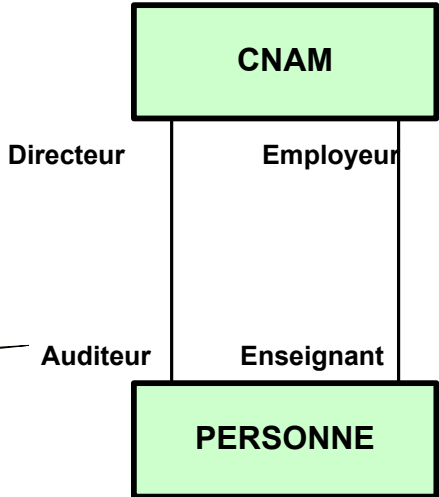
# LES VUES STATIQUES D'UML: DIAGRAMMES DE CLASSES

## ASSOCIATION, DOCUMENTATION ET ROLES

**Association:** relation sémantique entre classes qui définit un ensemble de liens



**Association en forme verbale active:** précise le sens de lecture principal d'une association



**Rôle:** spécifie la fonction d'une classe pour une association donnée

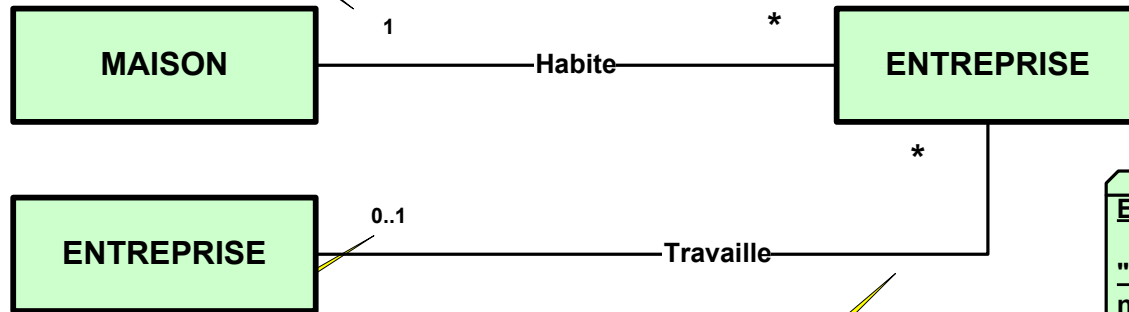
**DIAGRAMMES DE CLASSES: SEMANTIQUE.**

- ✓ Un DIAGRAMME de CLASSES est une collection d'éléments de modélisation statiques. (Il montre la structure d'un modèle ).
- ✓ Un DIAGRAMME de CLASSES fait abstraction des aspects dynamiques et temporels. Il est au niveau des Règles de Gestion
- ✓ Pour un MODELE complexe, plusieurs diagrammes de classes complémentaires doivent être réalisés. Par exemple, on pourra se focaliser, sur:
  - Les CLASSES qui participent à un CAS D'UTILISATION
  - Les CLASSES associées dans la réalisation d'un SCENARIO précis
  - Les CLASSES qui composent un PAQUETAGE
  - La structure hiérarchique d'un ensemble de CLASSES
- ✓ Pour représenter un contexte précis, un DIAGRAMME de CLASSES peut être instancié en un

# LES VUES STATIQUES D'UML: DIAGRAMMES DE CLASSES

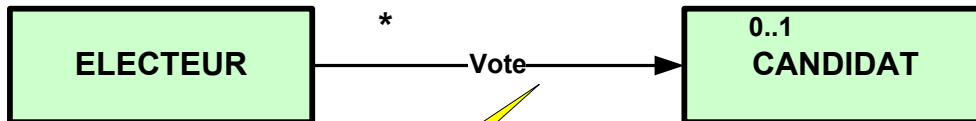
**Cardinalité**

## LES CARDINALITES ET L' ASSOCIATION DE NAVIGABILITE RESTREINTE



**Cardinalité**

Une PERSONNE travaille pour au plus une ENTREPRISE. Plusieurs PERSONNES travaillent pour une même ENTREPRISE



**Association à navigabilité restreinte**

**EXPRESSION DES CARDINALITES D'UNE RELATION EN UML:**

**"n"** : exactement "n" ( n, entier naturel >0, exemple : "1", "7" )

**n..m** : de "n" à "m" ( entiers naturels ou variables, m>n )

**\*** : de 0 à n, n étant quelconque.

**n\_** : Plusieurs ( équivalent à 0..n et 0..\* )

**n..\*** : "n" ou plus ( n, entier naturel ou variable, exemple: 0..5, 5..\* )

**ASSOCIATION A NAVIGABILITE RESTREINTE**

Par défaut, une association est navigable dans les deux sens. La réduction de la portée del'association est souvent réalisée en phase d'implémentation, mais peut aussi être exprimée dans un modèle où les instances d'une CLASSE ne "connaissent" pas les instances d'une autre.

Cela ne vous dit rien, vous qui êtes Merisiens ?



# LES VUES STATIQUES D'UML: DIAGRAMMES DE CLASSES

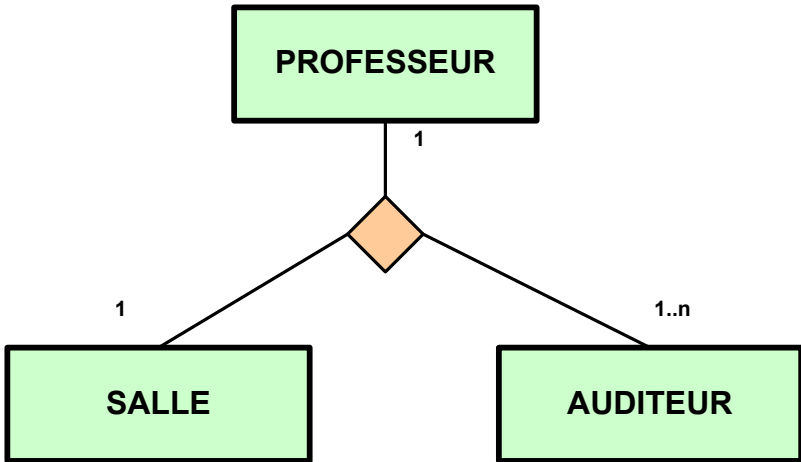
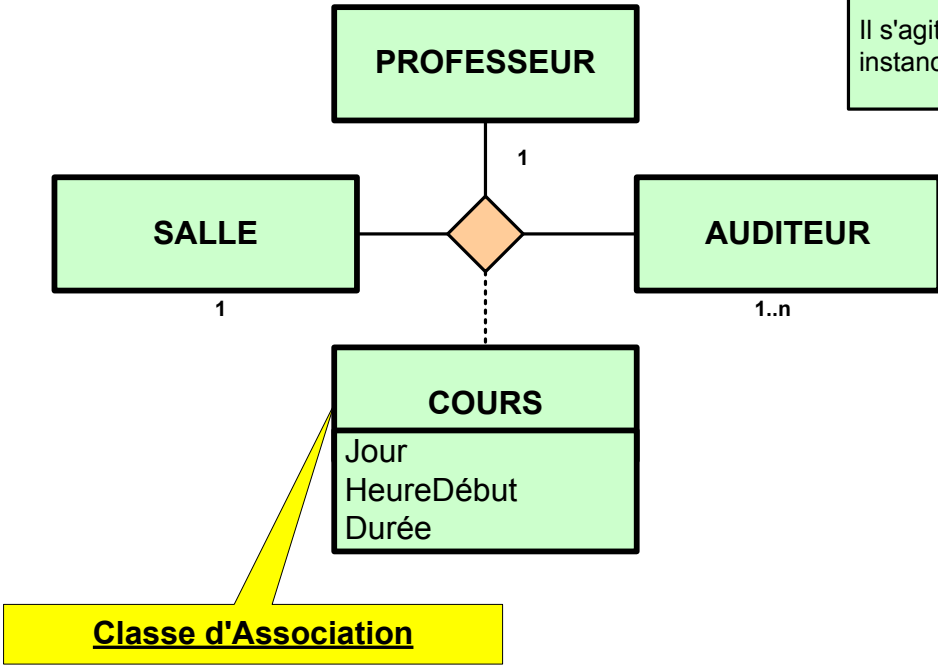
## LES CLASSES D'ASSOCIATIONS ET LES ASSOCIATIONS N- AIRES

Cela ne vous dit rien, vous qui êtes Merisiens ?



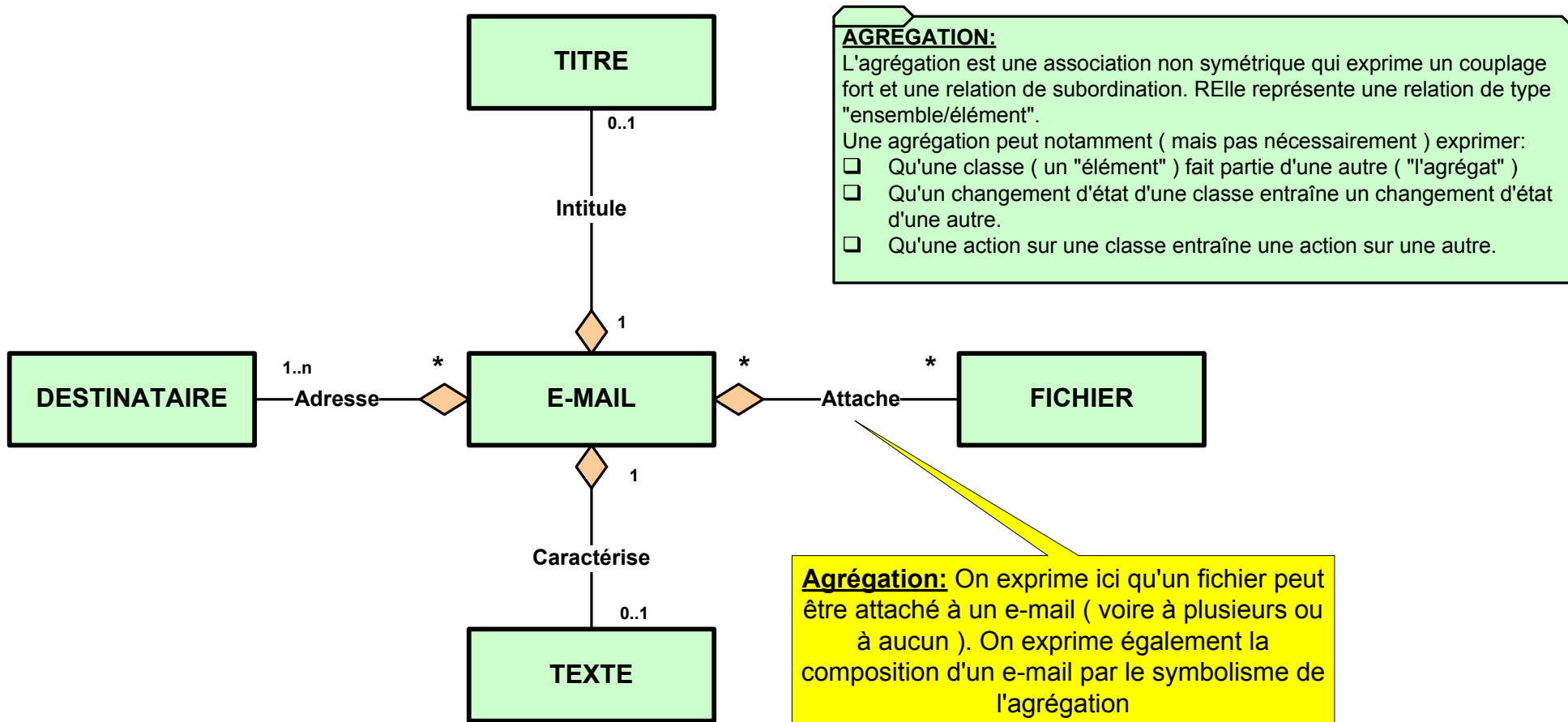
**ASSOCIATION N-AIRE:**  
 Il s'agit d'une ASSOCIATION qui relie plus de deux CLASSES. De telles associations sont difficiles à déchiffrer et peuvent induire en erreur. Il vaut mieux, pour plus de lisibilité, multiplier les associations.

**CLASSE D'ASSOCIATIONS:**  
 Il s'agit d'une classe qui réalise la description de la navigation entre les instances de plusieurs classes. .



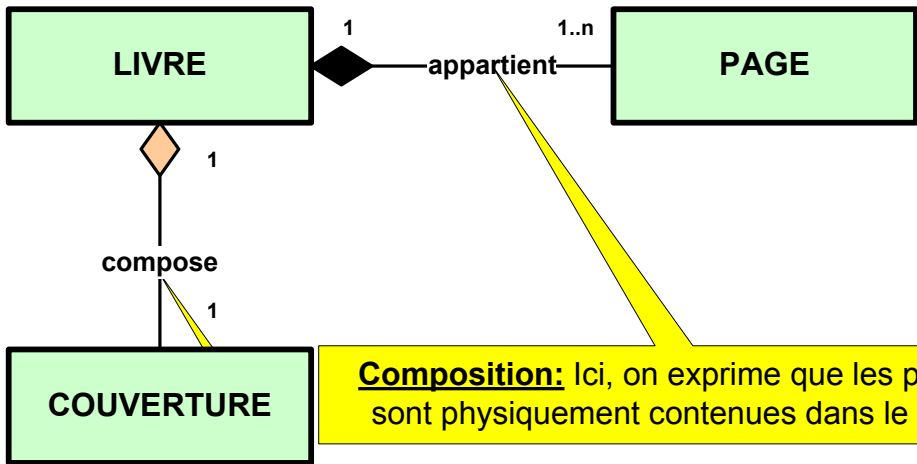
# LES VUES STATIQUES D'UML: DIAGRAMMES DE CLASSES

## L'ASSOCIATION D'AGREGATION



# LES VUES STATIQUES D'UML: DIAGRAMMES DE CLASSES

## L'ASSOCIATION DE COMPOSITION ET LES INTERFACES

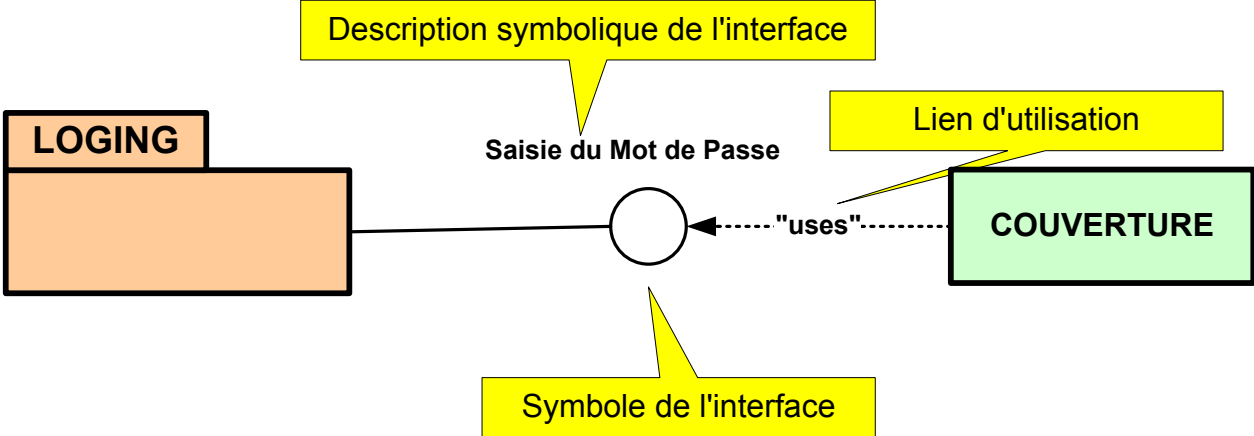


**COMPOSITION:**  
 La composition est une agrégation forte. Si l'agrégat est copié, ses composants le sont aussi. A un même moment, une instance de composant ne peut être liée qu'à un seul agrégat.

**INTERFACE:**  
 Une interface fournit une vue totale ou partielle d'un ensemble de services offerts par une classe de type "paquetage" ou un composant. Les éléments qui utilisent l'interface peuvent exploiter tout ou partie de l'interface

**Composition:** Ici, on exprime que les pages sont physiquement contenues dans le livre

**Agrégation:** Ici, on exprime qu'un livre peut être constitué d'une couverture



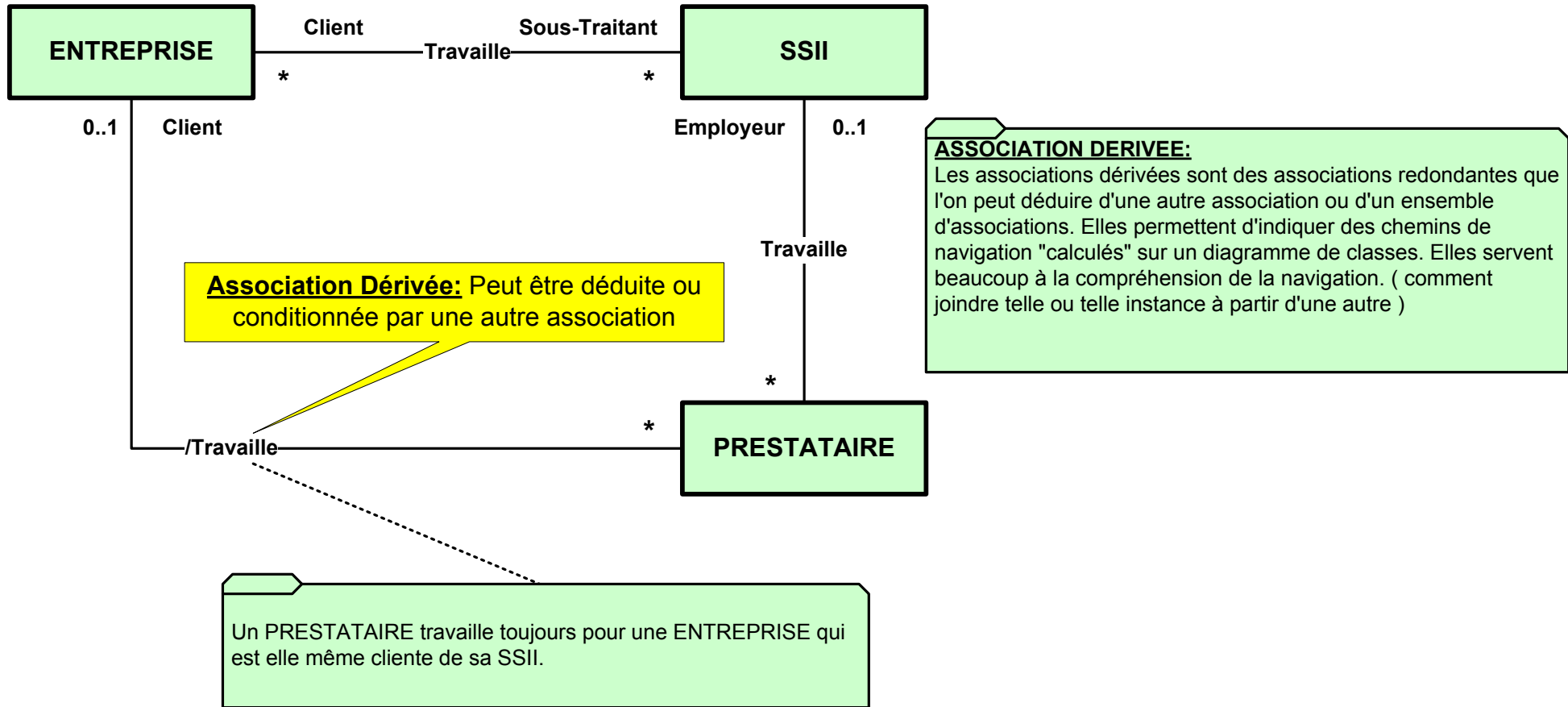
Description symbolique de l'interface

Lien d'utilisation

Symbole de l'interface

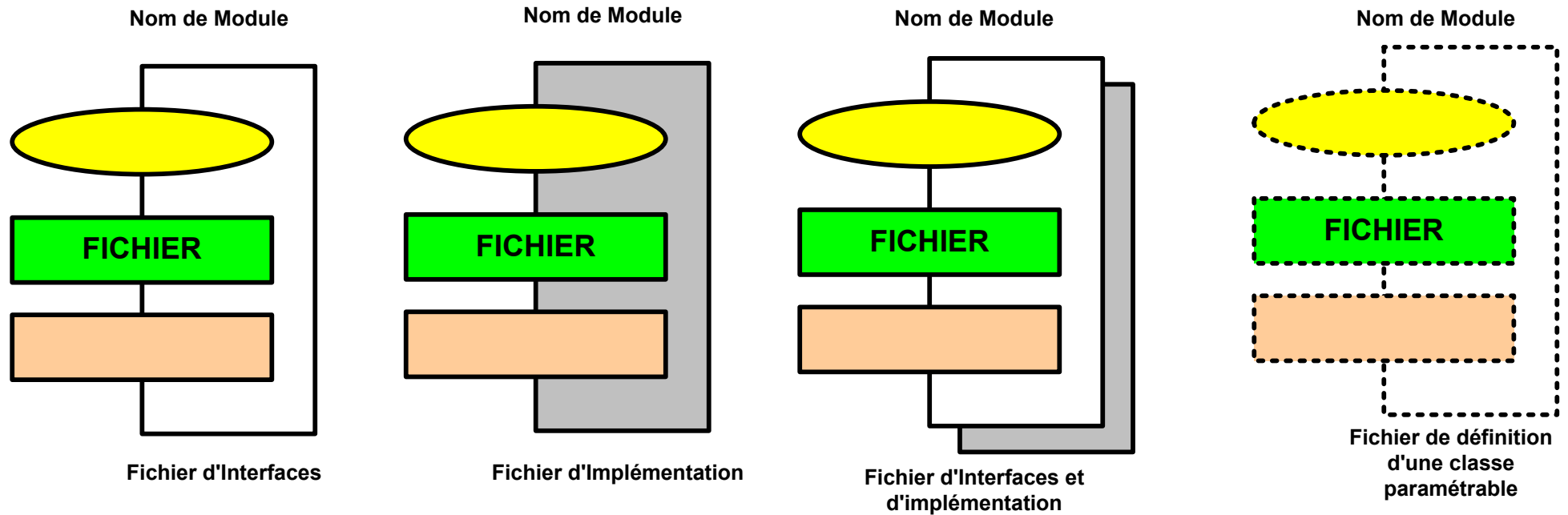
# LES VUES STATIQUES D'UML: DIAGRAMMES DE CLASSES

## LES ASSOCIATIONS DERIVEES



# LES VUES STATIQUES D'UML: DIAGRAMMES DE COMPOSANTS

## NOTATIONS



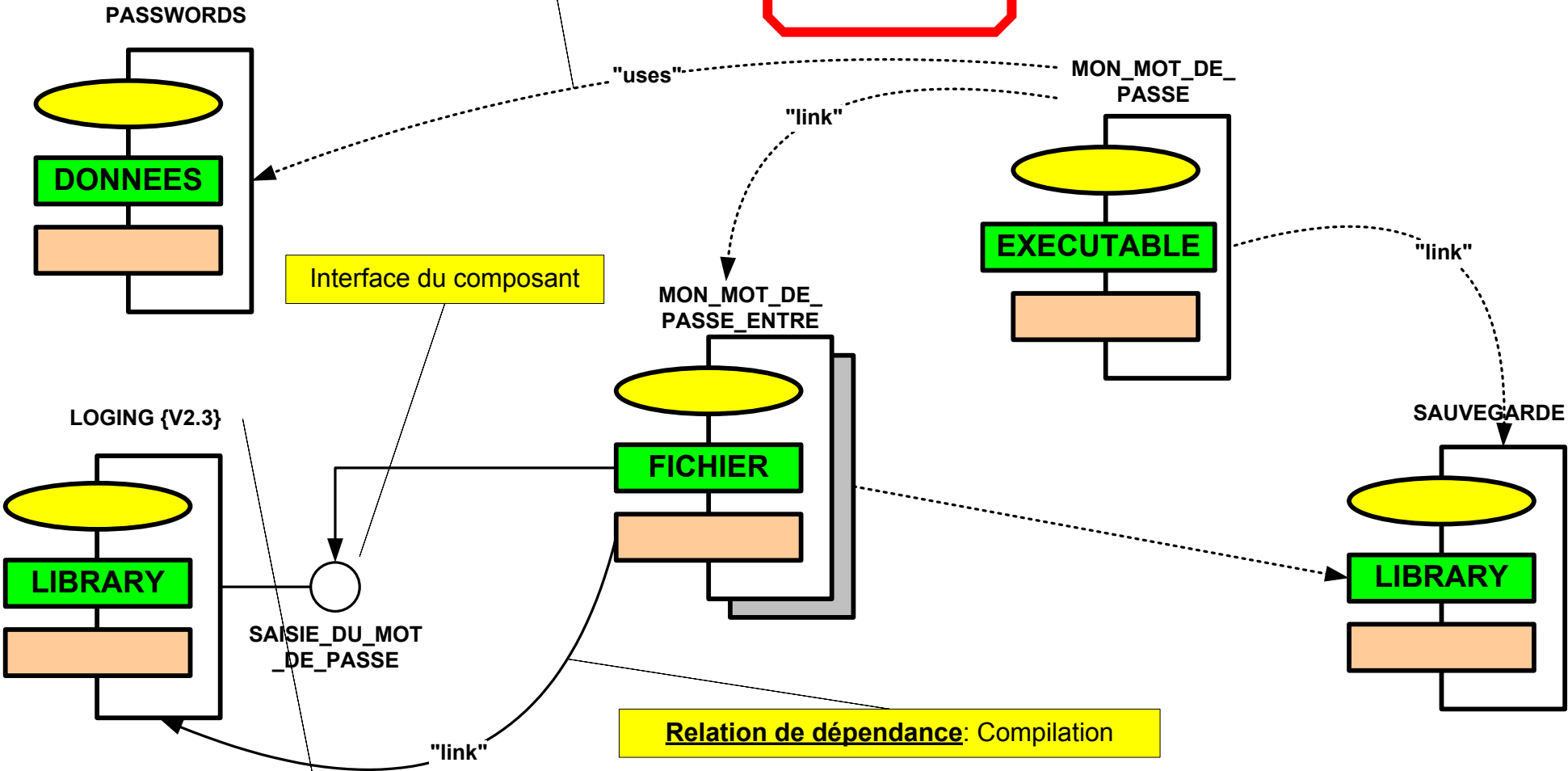
### DIAGRAMMES DE COMPOSANTS:

Les diagrammes de composants permettent de décrire l'architecture physique et statique d'une application en termes de modules ( fichiers sources, bibliothèques, exécutables, etc... ). Ils montrent la mise en oeuvre physique des modèles de la vue logique avec l'environnement de développement. Les dépendances entre composants permettent notamment d'identifier les contraintes de couplage et de mettre en évidence la réutilisation des composants. Les composants peuvent être organisés en paquetages qui définissent des sous systèmes.

# LES VUES STATIQUES D'UML: DIAGRAMMES DE COMPOSANTS

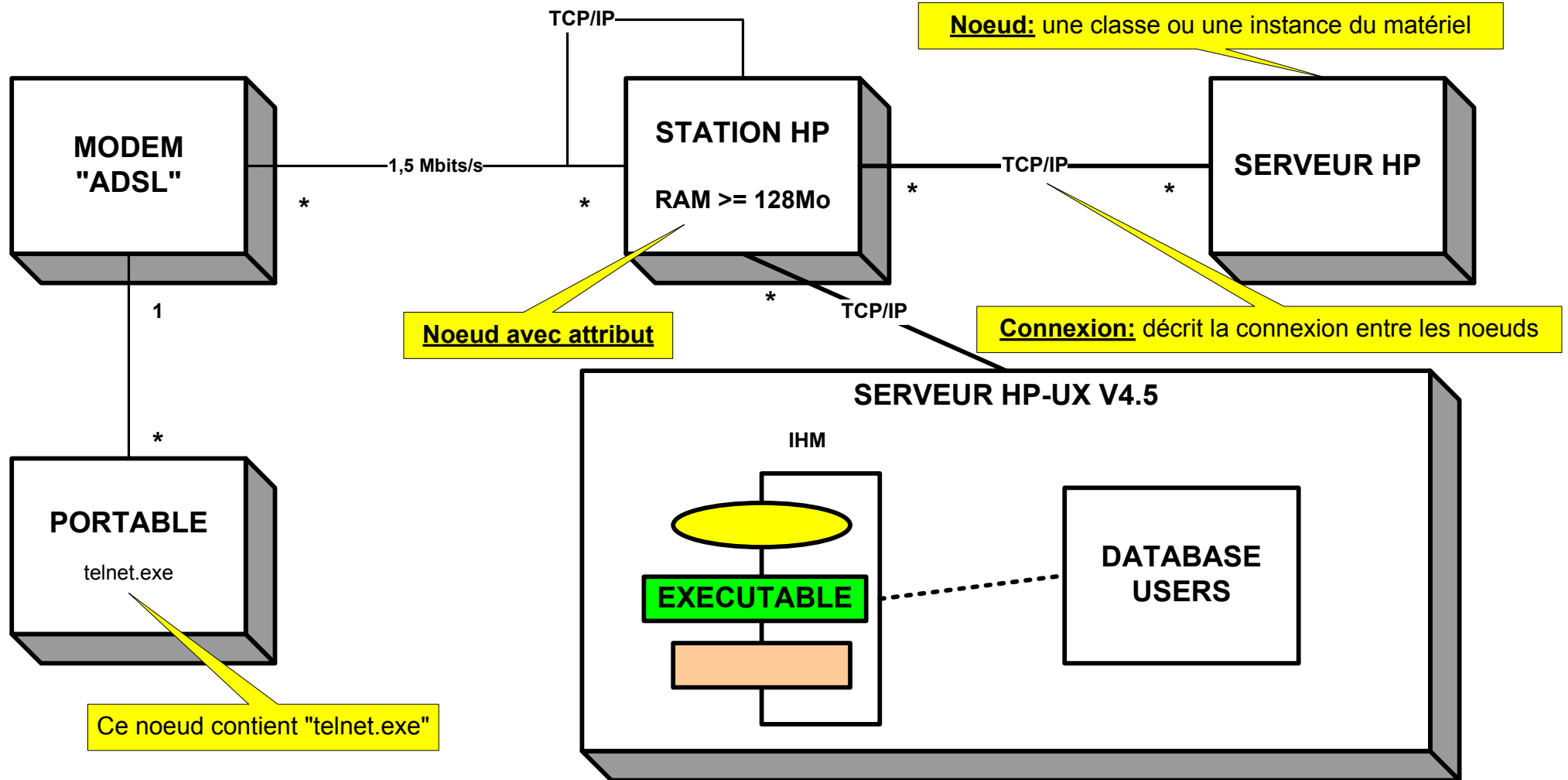
**EXEMPLE**

Relation d'utilisation



Contrainte sur le N° de version

# LES VUES STATIQUES D'UML: DIAGRAMMES DE DEPLOIEMENT

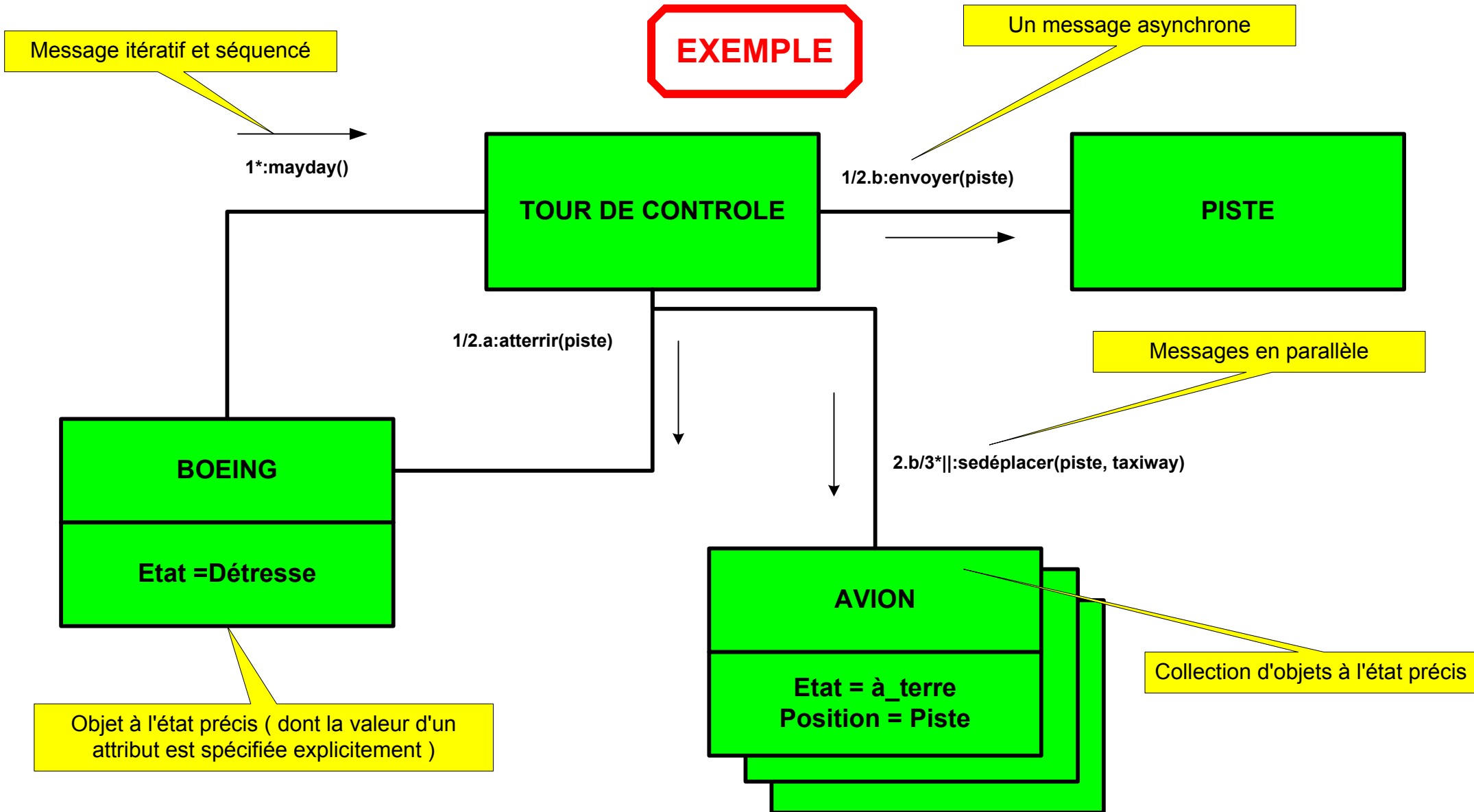


## DIAGRAMMES DE DEPLOIEMENT:

Les diagrammes de déploiement montrent la disposition physique des matériels qui composent la solution et la répartition des composants sur ces matériels. Les ressources matérielles sont représentées sous forme de noeuds. Les noeuds sont connectés entre eux à l'aide d'un support de communication. Les diagrammes de déploiement correspondent à la "vue de déploiement".

# LES VUES DYNAMIQUES D'UML: DIAGRAMMES DE COLLABORATION ET MESSAGES

## EXEMPLE



# LES VUES DYNAMIQUES D'UML: DIAGRAMMES DE COLLABORATION ET MESSAGES

## SYNTAXE D'UN MESSAGE

Pour chaque message, il est possible d'indiquer:

- Les clauses qui conditionnent son envoi
- Son rang ( son numéro d'ordre par rapport aux autres messages )
- Sa récurrence
- Ses arguments

La syntaxe d'un message est la suivante:

```
[ pré "/" ] [ [ " " "cond" ] ] [ séq ] [ "*" [ " || " ] [ " [ "iter" ] " ] ] : " [ r " : = " ] msg " ( [ par ] ) "
```

**pré:** Prédécesseurs ( liste de numéros de séquences de message séparés par une "/" . Indique que le message courant ne sera envoyé que lorsque tous ses prédécesseurs auront été traités ( permet la synchronisation de messages ) .

**cond:** Expression booléenne. Permet de conditionner l'envoi d'un message à l'aide d'une clause exprimée en langage booléen.

**séq:** Numéro de séquence du message. Il sont numérotés à la façon de chapitres dans un document, à l'aide de chiffres séparés par des points. Par exemple, l'envoi du message 1.3.5 suit immédiatement celui du message 1.3.4 qui font partie du flot de message 1.3. Pour représenter l'envoi simultané de deux messages, il suffit de les indexer par une lettre. par exemple, l'envoi des messages 1.3.a et 1.3.b est simultané.

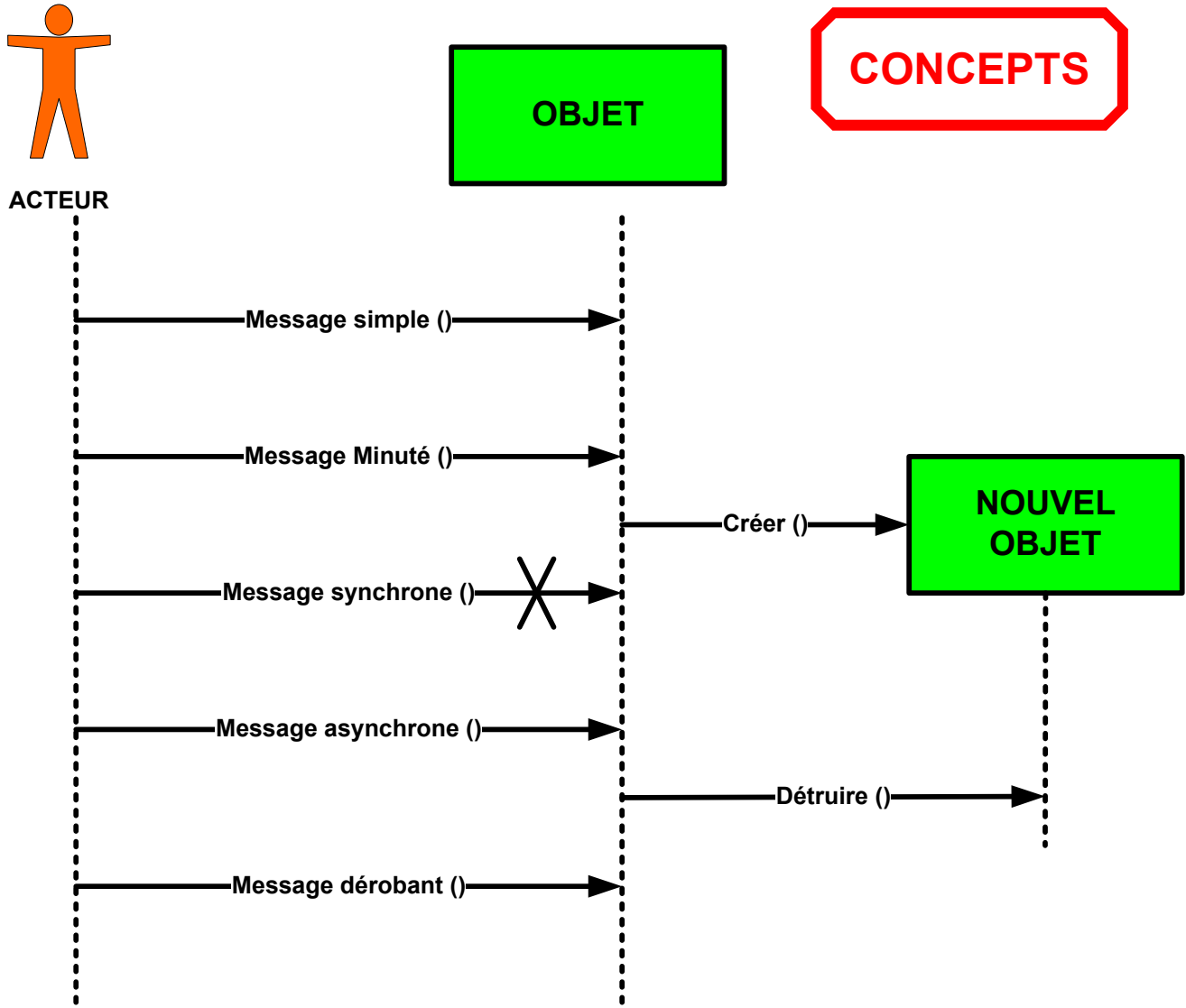
**Iter:** Récurrence du message. Permet de spécifier en langage naturel l'envoi séquentiel ( ou en parallèle avec " || " ). On peut aussi écrire en omettant "iter" et en n'utilisant que " \* " ou " \* || " ).

**r:** Valeur de retour du message. permet d'affecter la valeur de retour d'un message en tant que paramètre.

**msg:** nom du message.

**par:** paramètres ( optionnels ) du message.

# LES VUES DYNAMIQUES D'UML: DIAGRAMMES DE SEQUENCES

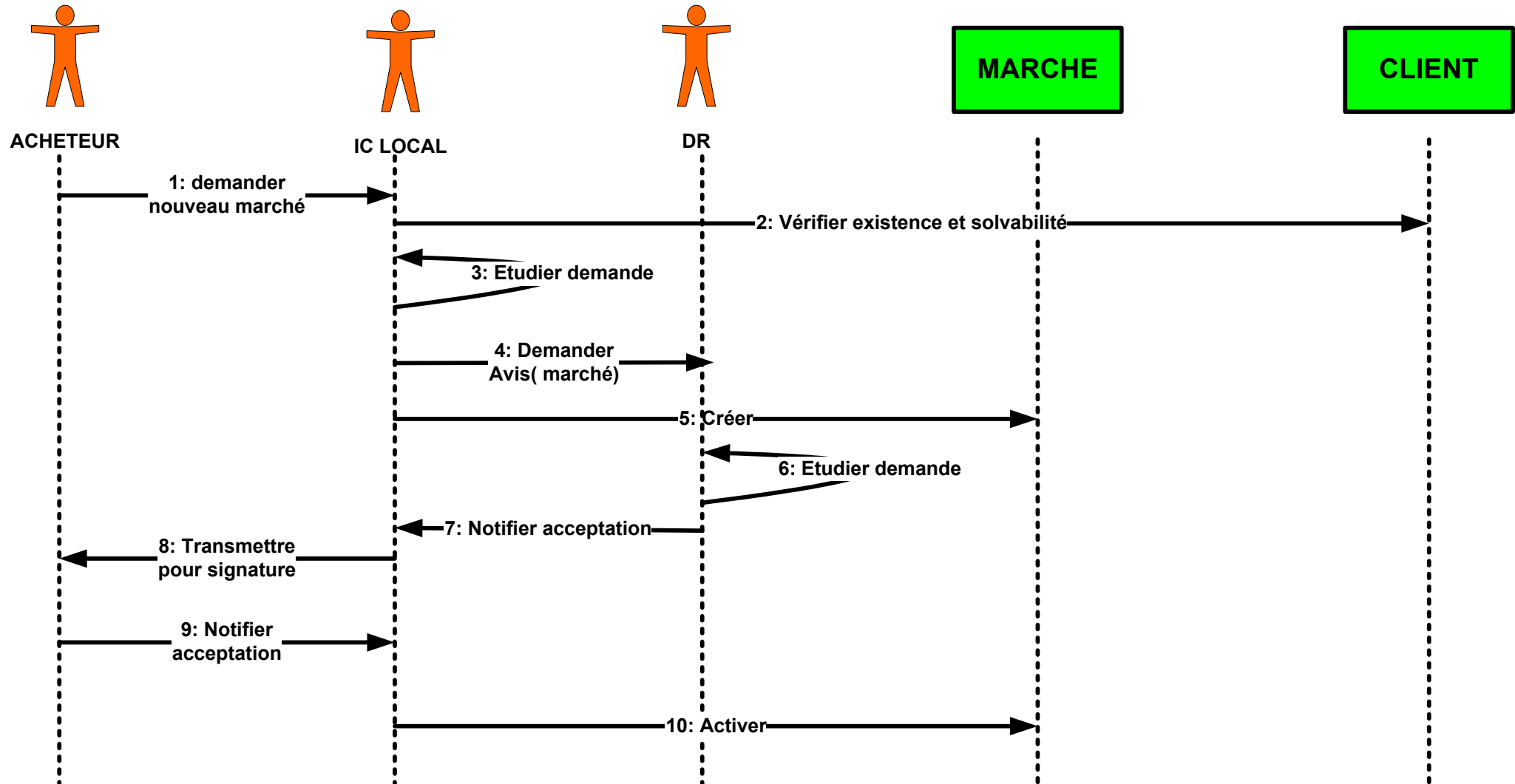


**CONCEPTS**

- ❑ Les diagrammes de séquences permettent de représenter des collaborations entre objets par l'envoi de messages. On met l'accent sur la chronologie d'envoi des messages.
- ❑ Les diagrammes de séquences peuvent servir à illustrer un cas d'utilisation ( Cf. transparent suivant )
- ❑ L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme appelé "ligne de vie" et qui s'écoule du haut vers le bas.
- ❑ Les diagrammes de séquence et les diagrammes d'états transitions sont les vues dynamiques importantes en modélisation UML.

## LES VUES DYNAMIQUES D'UML: DIAGRAMMES DE SEQUENCES

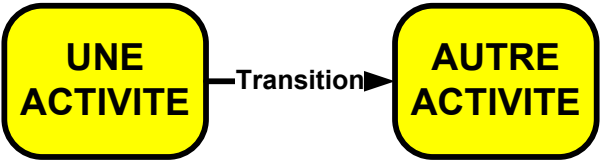
## EXEMPLE DE DIAGRAMME DE SEQUENCE DU CAS D'UTILISATION "GERER MARCHÉ"



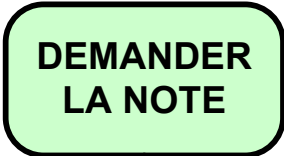


# LES VUES DYNAMIQUES D'UML: DIAGRAMMES D'ACTIVITES

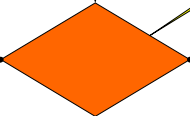
## NOTATION ET EXEMPLE



Garde: conditionne la transition

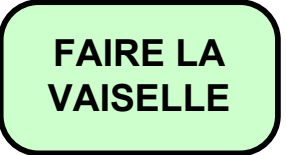
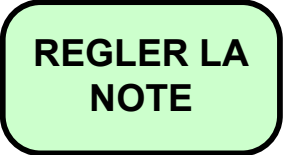


Branchement conditionnel



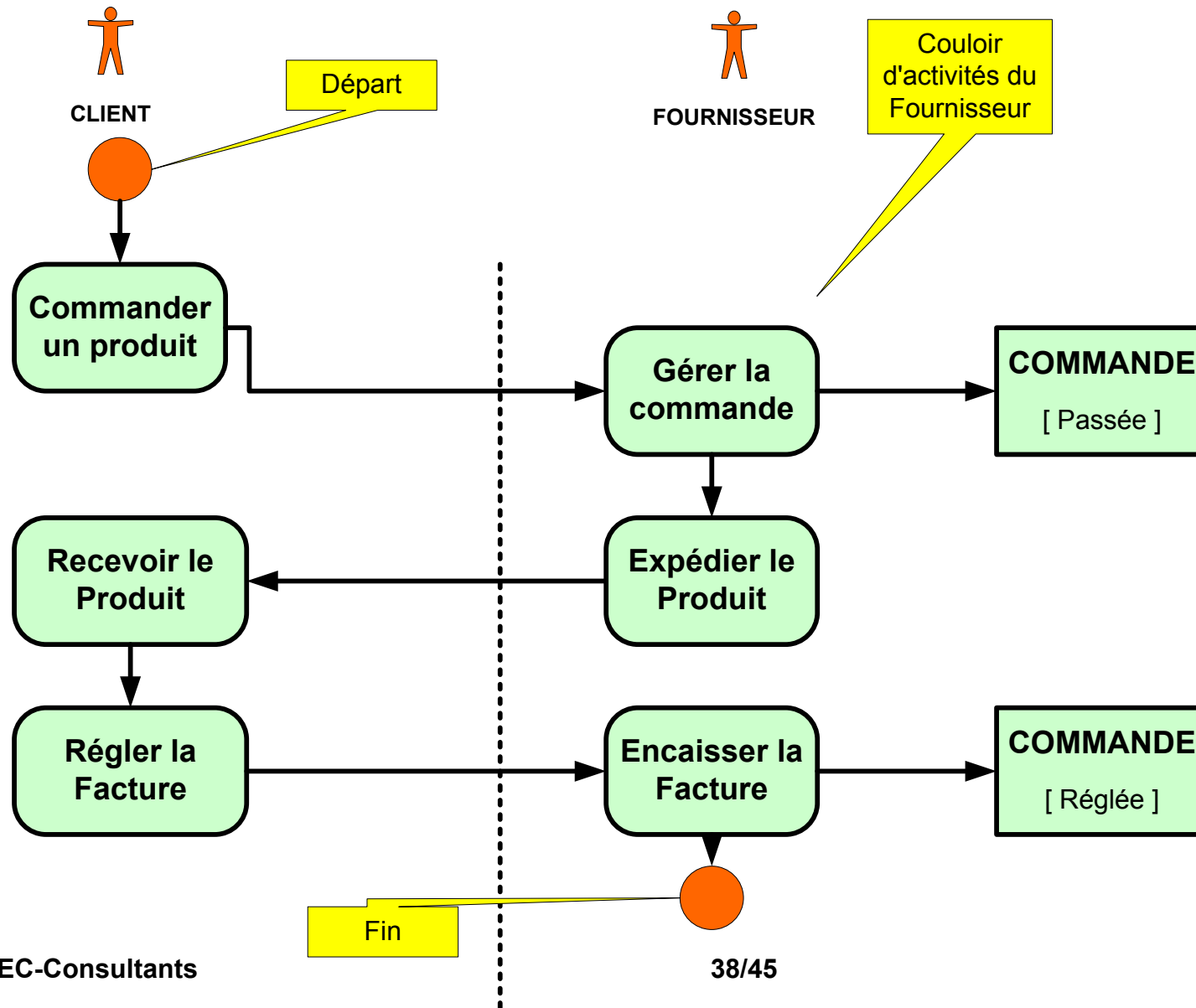
[ Prix <= SommeDisponible

[else]



- ❑ UML permet de représenter graphiquement le comportement d'une méthode ou le déroulement de cas d'utilisations, à l'aide de diagrammes d'états-transitions ( variantes des diagrammes d'états-transitions ).
- ❑ Les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre.
- ❑ En théorie, tous les mécanismes dynamiques pourraient être décrits par un diagramme d'activité, mais seuls les mécanismes complexes ou intéressants méritent d'être représentés.
- ❑ Afin d'organiser un diagramme d'activités selon les différents responsables des actions représentées, il est possible de définir des "couloirs d'activités" ( Cf Slide suivant )

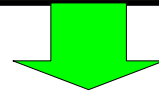
# LES VUES DYNAMIQUES D'UML: COULOIRS D'ACTIVITES



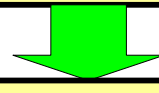
# MERISE ET UML: APPROCHE COMPARATIVE

## LES CINQ PRINCIPES FONDAMENTAUX DE MERISE

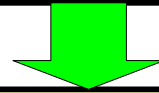
L'APPROCHE SYSTEMIQUE



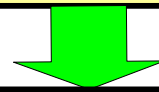
LES CYCLES DE CONSTRUCTION DU SI



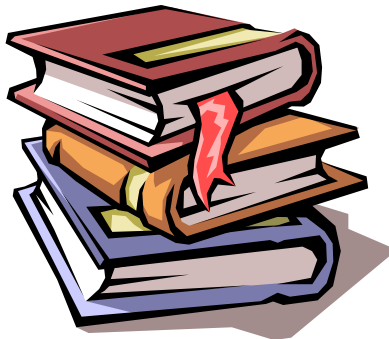
L'APPROCHE FONCTIONNELLE



SEPARATION DONNEES - TRAITEMENTS



L'APPROCHE TOP - DOWN



# MERISE ET UML: APPROCHE SYSTEMIQUE

## MERISE

La méthode MERISE s'intéresse aux systèmes ouverts en relation permanente avec leur environnement ( à partir du niveau 3. Cf. Cours de systémique ). Trois propriétés majeures sont alors considérées.

### GLOBALITE

Contrairement au modèle Cartésien, le comportement d'un système n'est pas la somme des comportements de ses parties

### RETROACTION

Le système réagit à toute stimulation en générant des résultats selon des boucles d'actions-réactions mises en oeuvre au sein de ses composants. ( feed-back d'ordre 3 )

### FINALITE

Le système ne peut être stabilisé que si on lui fournit des valeurs acceptables préalablement définies.

## UML

### L'APPROCHE UML

L'approche par les CAS D'UTILISATION constitue de fait une approche SYSTEMIQUE. Les ACTEURS et les MESSAGES échangés sont pris en compte.

### COMMENTAIRES

La Maîtrise des concepts de la systémique ( ENVIRONNEMENT, ENTREES, SORTIES, OBJECTIFS, STRUCTURE et PROCESSUS ) permet de réaliser une approche constructive et consciente des cas d'utilisation. Elle facilite la détermination des CAS D'UTILISATION selon un double process TOP-DOWN et BOTTOM-UP. L'avantage est que le PROCESSUS métier du système est mieux décrit et plus compréhensible par les utilisateurs.

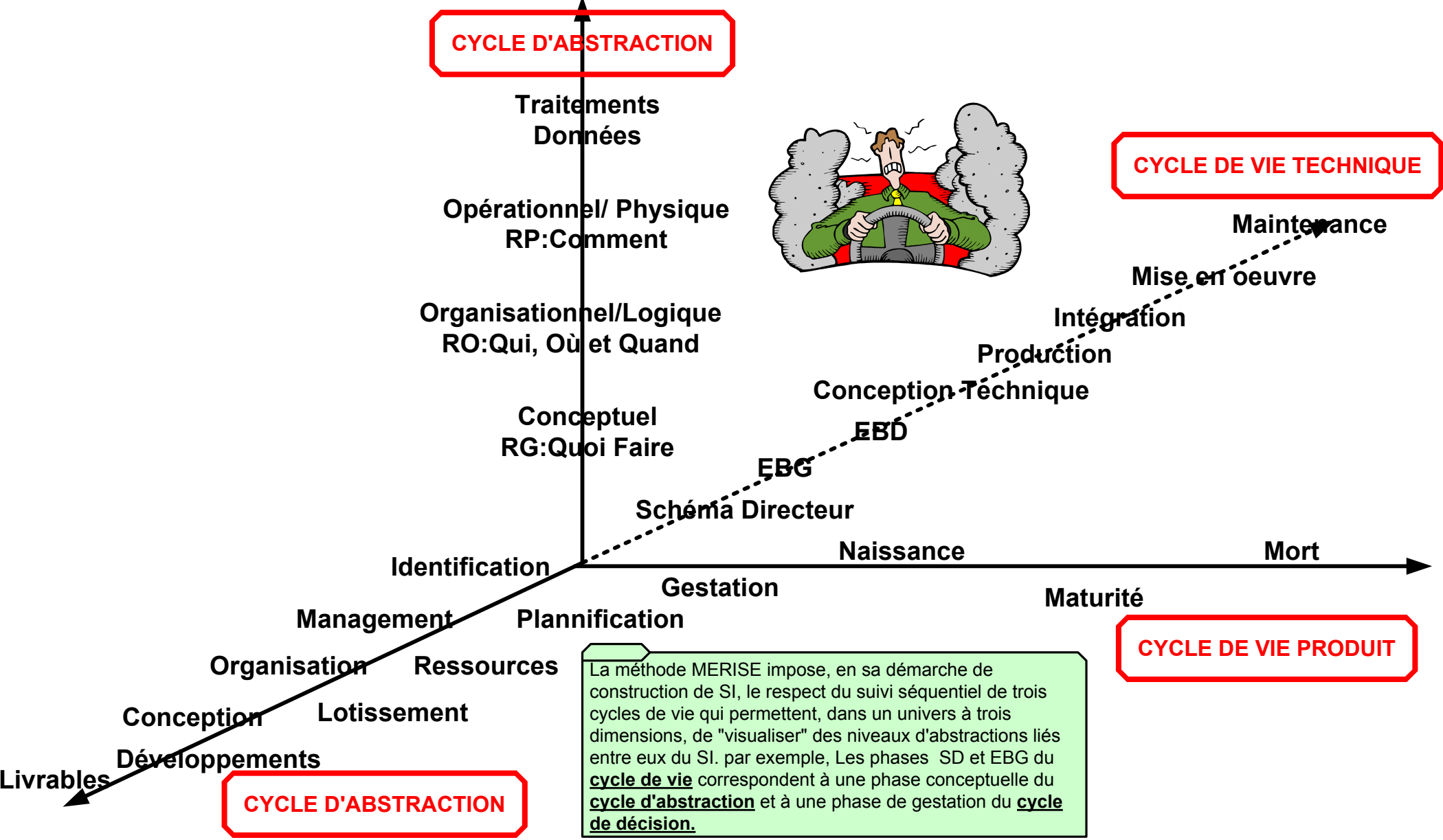
**FINALITES:** Raisons d'être, vocation d'un système. Se déclinent et se formalisent en BUTS. Elles ne sont pas opératoires. ( Cf. FACTEURS ). Par exemple: "Recentrer l'activité de l'entreprise sur son métier de base qui est la vente de contrats d'assurances".

**BUT:** Formalisation des FINALITES. Elles se déclinent en OBJECTIFS qui eux, doivent être opératoires. ( Cf. CRITERES ). Par exemple: "Recentrer les résultats de la compagnie sur les primes d'assurances plutôt que sur les marchés financiers"

**OBJECTIFS:** Concrétisation des BUTS sur la base de critères d'évaluations auxquels sont affectés des niveaux quantifiés à atteindre. ( Cf. METRIQUES ). Par exemple: "Les cotisations d'assurances doivent contribuer à 60% du résultat de la compagnie "

**ACTEURS:** Classe stérotypée représentant une abstraction faisant partie de l'ENVIRONNEMENT du système étudié.

# MERISE ET UML: LES CYCLES DE CONSTRUCTION DU SI



La méthode MERISE impose, en sa démarche de construction de SI, le respect du suivi séquentiel de trois cycles de vie qui permettent, dans un univers à trois dimensions, de "visualiser" des niveaux d'abstractions liés entre eux du SI. par exemple, Les phases SD et EBG du cycle de vie correspondent à une phase conceptuelle du cycle d'abstraction et à une phase de gestation du cycle de décision.

# MERISE ET UML: LES CYCLES DE CONSTRUCTION DU SI

## MERISE

## UML

### CYCLE DE VIE

Pour MERISE, la création d'un PRODUIT artificiel ( application ) doit suivre un cycle formel ( Gestation, Naissance, Maturité et Mort ) pour le PRODUIT et un cycle technique pour son élaboration ( SD, EBG, EBD, CT, PRODUCTION, INTEGRATION, MISE EN OEUVRE et MAINTENANCE ). La démarche est formalisée et méthodifiée

### CYCLE DE VIE

UML ne définit pas de cycle de vie. Il est implicite et correspond à un cycle itératif et incrémental guidé par les CAS D'UTILISATION. et centré sur les vues.

### CYCLE D'ABSTRACTION

Les trois niveaux retenus correspondent à des degrés de stabilité et d'invariance différents du SI ( Conceptuel avec les RG et le QUOI FAIRE, Logique avec les RO et le QUI FAIT, le OU on FAIT et QUAND on fait et Physique avec les RP et le COMMENT FAIRE ). On peut donc étudier le système progressivement en allant du général au Particulier ( TOP - DOWN ).

### CYCLE D'ABSTRACTION

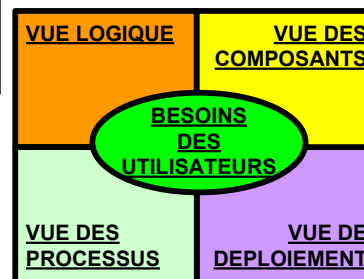
UML permet de modéliser les différents niveaux du SI . ( Conceptuel, Organisationnel et Physique ). Pour cette partie UML propose différents modèles ( CAS d'UTILISATION, PAQUETAGES, CLASSES, COMPOSANTS et NOEUDS ). UML laisse le soin à l'informaticien de présenter ces modèles de manière cohérente par rapport au niveau d'abstraction.

### CYCLE DE DECISION

Le cycle de décision permet d'organiser l'intervention des personnes de l'entreprise en fonction de la hiérarchie des décisions à prendre. MERISE est une méthode participative où chaque Acteur est amené à formuler ses besoins en fonction de ses niveaux de préoccupations. Le cycle de décision doit définir les points de passages ( jalons ) et les contrôles.

### CYCLE DE DECISION

Comme MERISE, UML associe étroitement les utilisateurs aux prises de décisions. La mise en oeuvre d'UML doit s'accompagner d'un cycle de décision complet avec points de contrôles et jalons.



# MERISE ET UML: APPROCHE FONCTIONNELLE

## MERISE

### GLOBALITE

Merise propose une approche descendante où le système réel est découpé en **ACTIVITES** ( MCC, MCD, MCT ) elles mêmes découpées en **FONCTIONS** ( MOT, MLD et MOC ). Les **FONCTIONS** sont composées de RG elles mêmes regroupées en **OPERATIONS**. Ces RG au niveau conceptuel génèrent des **MODULES** décomposés en **MODULES** plus simples pour obtenir des **MODULES** élémentaires. Les limites de cette approche résident dans la **REUTILISABILITE**. Il existe aussi des différences entre des modules plus "utilisateurs" ( Calculs de dates, calcul de validité de compte via une clé etc.. ) et des modules plus "techniques" ( module d'accès à un SGBD, Module de création de liste sur écran ).

## UML

### L'APPROCHE UML

L'approche fonctionnelle est une spécificité **MERISE** dont **UML** se démarque. Dans **UML**, les **FONCTIONS** cèdent le pas aux **CAS D'UTILISATION** qui permettent de situer les besoins des Utilisateurs dans un contexte réel. A chaque scénario correspondent des diagrammes d'interactions ( **SEQUENCE** et **COLLABORATION** ) entre les **OBJETS** et non pas des **FONCTIONS**. Pour compléter , **UML** introduit des diagrammes d'**ACTIVITES**. Cette approche rend le système le plus indépendant possible des besoins en donnant naissance à des **COMPOSANTS** réutilisables.

Nous avons affaire ici à la véritable différence ( culturelle et conceptuelle ) entre **MERISE** et **UML**. Cette différence impose que l'informaticien "pense" différemment.



**CONTRE**



# MERISE ET UML: SEPARATION DONNEES - TRAITEMENTS

## MERISE

### L'APPROCHE MERISE

Merise propose de considérer le système réel selon deux points de vues: Un point de vue statique ( les **DONNEES** ) et un point de vue dynamique ( Les **TRAITEMENTS** ). Cela permet d'avoir deux vues différentes à valider. Cette validation permet de contrôler que les besoins exprimés sont bien traduits en termes d'informations et que ces dernières sont bien toutes utilisées. La validation consiste à vérifier l'**EXISTENCE** des informations, des **ENTITES** et des **RELATIONS**; la cohérence des **CARDINALITES** et des **CONTRAINTES** ainsi que l'exhaustivité des actions élémentaires sur les **COMPOSANTS**. Ce qui permet de lier les **DONNEES** et les **TRAITEMENTS**.

### L'APPROCHE MERISE PAR LES ECHANGES

Merise pour la détermination du champ de l'étude, utilise le concept de **FLUX** entre **ACTEURS**. ( **MCC** avec **ACTEURS** et **MESSAGES** ou **GRAPHE de FLUX** ). Les Modèles de **FLUX** mettent en jeu les **ACTEURS** et leurs **ECHANGES** avec le système à construire vu de l'**EXTERIEUR**. Si l'on veut tenir compte d'**ACTEURS** internes, il convient de déterminer un **SOUS-SYSTEME**, les autres **STRUCTURES** du système premier devenant des **ACTEURS** analysables en termes d'**ECHANGES**. ( Cf. cours de MP )

### L'APPROCHE MERISE DU GENERAL AU PARTICULIER

Merise est une méthode de type "descendante" ( **TOP - DOWN** ). La vision globale du système est affinée par intégrations successives des différentes orientations retenues, classées selon le séquençement du cycle d'abstraction. Dans les faits, les premières étapes s'appuient sur l'étude de l'**EXISTANT** ( Interviews et documentation ). Les **MCC**, **MCD** et **MCT** synthétisent ces travaux. Le travail est ensuite complété par les **OBJECTIFS** fournis du système cible et déclinés en modèles organisationnels ( **MLD**, **MOT** et **MOC** ) pour ensuite produire les modèles physiques ( **MPD** et **MPT** ) rarement décrits en pratique.

## UML

### L'APPROCHE UML

L'approche **OBJET** associe les **INFORMATIONS** et les **TRAITEMENTS**. De cette façon, un certain niveau de cohérence est assuré. Il s'agit ici d'un autre point clé de différence. En effet, l'approche UML permet de modéliser les systèmes complexes et d'exprimer de manière **INTEGREE** la structure et la dynamique avec un même concept plus proche de la réalité des choses. Cette approche étant renforcée par les **CAS D'UTILISATION** qui se traduisent par des **COLLABORATIONS** entre les **OBJETS**.

### L'APPROCHE UML PAR LES ECHANGES

Dans UML, l'entreprise est décrite d'abord sous forme de **PROCESSUS "métiers"** et ensuite sous forme de **TRAVAILLEURS ( Rôles )** qui interagissent dans l'exécution du Processus. Les **ACTIVITES** de ces travailleurs sont ensuite décrites dans les diagrammes d'**ACTIVITES**. Ceci permet de fournir une vision simplifiée et de haut niveau de l'organisation tout en décrivant séparément les détails des activités et des échanges. .

### L'APPROCHE UML

UML permet aussi bien une approche descendante ( **TOP-DOWN** ) que montante ( **BOTTOM-UP** ). Les différents diagrammes fournissent des supports pour matérialiser une démarche progressive allant du global vers le détaillé, pour passer du **SYSTEME D'INFORMATIONS** au **SYSTEME INFORMATIQUE**. Un des avantages d'UML réside dans le fait que le concept de base ( l'**OBJET** ) peut être décliné sur tout le cycle d'abstraction.

## MERISE ET UML: MODELISATION DU METIER

### MERISE

#### L'APPROCHE MERISE

Merise s'appuie sur des concepts de **DOMAINES**, d'**ACTEURS**, de **FLUX**, de modèles liés aux niveaux **CONCEPTUELS** et **ORGANISATIONNELS** pour décrire le **METIER** indépendamment de ses **STRUCTURES**. Le **DOMAINE** est défini comme le ou les **PROCESSUS** métiers candidats à l'étude et en relation avec d'autres domaines ( par exemple, le domaine de la **GESTION COMMERCIALE** en relation avec le domaine de la **FABRICATION** ). Un domaine est subdivisé en **PROCESSUS** ( Gestion des Tarifs, Gestion des Clients, Gestion des ventes, Gestion du budget, etc... ). Il est néanmoins possible de "calquer" la définition des domaines sur les **ACTIVITES** de l'entreprise. Par exemple, il pourrait exister un domaine **VPC** et un domaine **VENTE EN MAGASINS**. Ce qui ne permet pas, selon **MERISE**, une factorisation aisée des **FONCTIONNALITES**.

ALors les gars ?  
On fait quoi ?  
MERISE ou UML ?

### UML

#### L'APPROCHE UML

Pour UML, la définition du **DOMAINE** est un préalable. Et, les **DOMAINES** étudiés dérivent directement des **ACTIVITES** de l'entreprise. Ce qui permet de trouver des scénarios de développements. Comme nous ne sommes plus en approche **FONCTIONNELLE**, l'approche est plus aisée. Un objet **CLIENT** reste le même que nous soyons en Gestion Commerciale ou en Gestion de Production. La spécialisation de ces **OBJETS** venant par la dérivation et l'héritage associées aux **CLASSES**.

