Initiation à UML
AFCEPF

bernard.granier@sfr.fr

UML

- Introduction
- Cas d'utilisation
- Diagramme d'activité
- Diagramme de classes
- Diagramme d'état
- Diagramme de séquence
- Diagramme de packages

UML

- Diagramme de déploiement
- Diagramme de composant
- Conclusion

Introduction

Introduction

- Plan
  - Objectif du cours
  - Bibliographie
  - Qu'est ce qu'UML
  - A quoi sert UML
  - Vue d'ensemble
  - Ce qui n'est pas couvert par UML

Introduction

- **Objectif du cours 1/1**
  - **Introduction à UML**
    - **Ce n'est pas un cours avancé, ni d'expertise**
  - **Fournir les bases essentielles**
  - **Présenter les schémas usuels**
  - **Comprendre la documentation et la bibliographie**

Introduction

- **Bibliographie 1/1**
  - **http://www.omg.org/ : norme**
  - **UML 2.0 de Martin Fowler, ed. Campus Press**
    - **Bon livre, didactique, clair**
  - **Tête la première Analyse et conception orientées objet, ed. O'Reilly**
    - **Original, pédagogique, complet**

Introduction

- Qu'est-ce qu'UML 1/2
  - UML : Unified Modeling Language
  - UML est un langage
    - Ensemble de schémas modélisant les SI
      - Tous les éléments d'un SI, d'une application
  - UML sous-tend une méthode
    - Mais ce n'est pas une méthode
    - Une méthode est à mettre en place (itérative)

Introduction

- Qu'est-ce qu'UML 2/2
  - Outils
    - IBM Rahpsody
    - Sparx
    - Modelio
    - Plug-in UML d'Eclipse : Papyrus
    - MagicDraw
    - Soyatech
    - StarUML / White Star UML
    - ObjectAid
    - ArgoUML, … LabUML

Introduction

- **À quoi sert une UML 1/2**
  - **À débugger le logiciel**
    - **Une bonne description du logiciel permet de cerner les bugs**
  - **A le faire évoluer**
    - **Une bonne description du logiciel permet de savoir comment le faire évoluer**

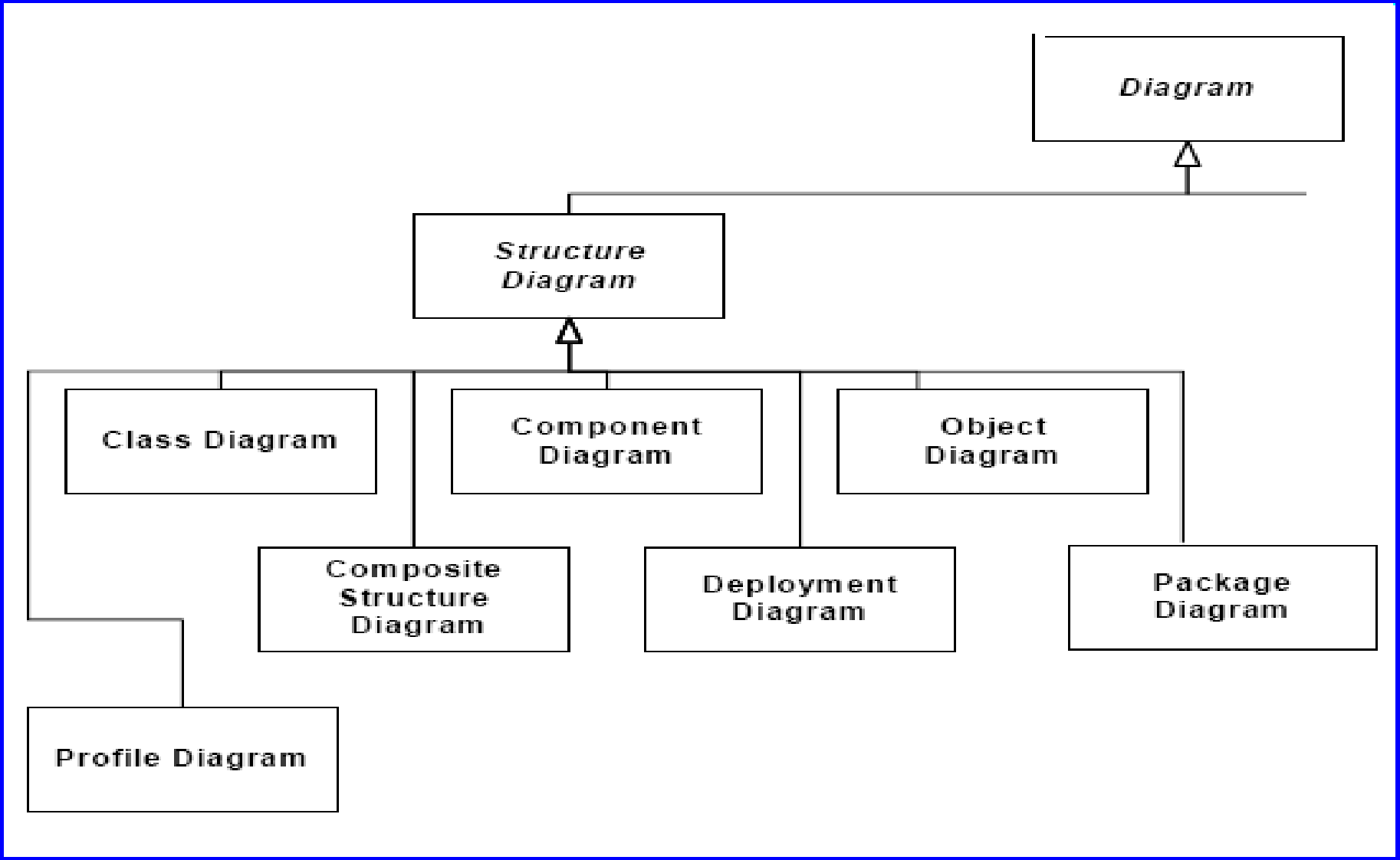
Ce sont probablement les points les plus importants

Introduction

- À quoi sert une UML 2/2
  - A bien concevoir un logiciel
    - A répondre aux besoins
  - UML : technique la plus courante dans le monde objet
    - Utilisable partiellement
    - Sans concurrent dans le monde objet
    - Mais suivant les domaines :
      - Merise
      - SADT
      - SART
      - ...

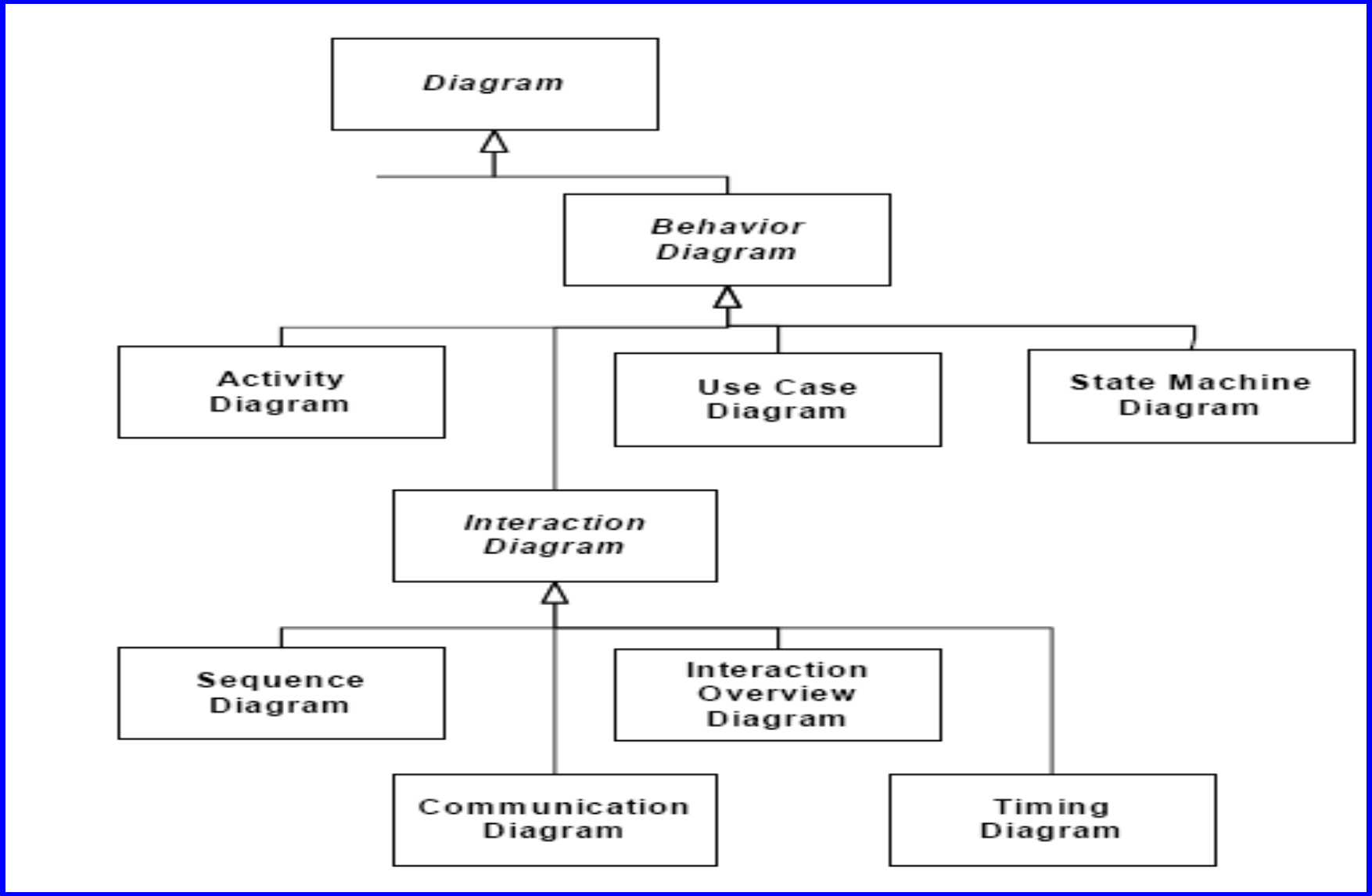
Introduction

- **Vue d'ensemble 1/2**



Introduction

- **Vue d'ensemble 2/2**



Introduction

- Ce qui n'est pas couvert par UML ... parce que ce n'est pas son objet 1/2
  - Règles de codage
  - Gestion de configuration
  - Gestion du build
  - Gestion des tests
  - Gestion des traces
  - Gestion des exceptions

Introduction

- Ce qui n'est pas couvert par UML ... parce que ce n'est pas son objet 2/2
  - La modélisation des IHM
  - L'analyse des systèmes
    - Point adressé par SysML
    - Thème « ancien » ayant ses propres méthodes
    - N'utilisant pas UML culturellement
    - UML ayant quelques défauts
      - Les « preuves mathématiques » d'UML sont à développer

Cas d'utilisation

Cas d'utilisation

- Plan
  - À quoi cela sert
  - Acteurs de cas d'utilisation
  - Liste de cas d'utilisation
  - Description de cas d'utilisation
  - Relations entre cas d'utilisation
  - Diagramme de contexte
  - Difficultés des cas d'utilisation

Cas d'utilisation

- Plan
  - Critères d'évaluation
  - QCM

Cas d'utilisation

- **À quoi cela sert 1/3**

USE·CASE·DÉTAILLÉ·«CRÉATION·PUB»¶	
Référence¶	5.2.1¶
Justification·¶	Un·client·se·connecte·au·site·pour·créer·sa·publicité¶
Description¶	1.→#CUST·visualise·la·page·d'accueil·du·site¶ 2.→#CUST·s'identifie¶ 3.→#CUST·click·sur·le·lien·«·créer·une·publicité·»¶ 4.→#CUST·saisie·le·texte·de·sa·publicité¶ 5.→#CUST·upload·une·bannière¶ 6.→#CUST·saisie·la·localisation·du·lieu·d'affichage¶ 1.→#CUST·se·déconnecte¶
Acteurs¶	#CUST¶
Use·cases·inclus¶	N/A¶
Déclenchement¶	Après·avoir·payé·son·adhésion·#CUST·saisie·une·publicité¶
Pré-conditions¶	Le·site·est·opérationnel·et·#CUST·est·correctement·enregistré¶
Post-conditions¶	La·publicité·est·stockée¶
Scénario·(s)¶	Erreur·de·saisie,·taille·de·la·bannière·incorrecte,·pertes·de·connexion¶
Note¶	N/A¶

Cas d'utilisation

- **À quoi cela sert 2/3**

C'est chouette de raconter des histoires pour spécifier un logiciel

Les clients, ils nous ennuient et ils ne savent pas ce qu'ils veulent

Un soft ne répond vraiment aux besoins qu'après plusieurs versions ...

Cas d'utilisation

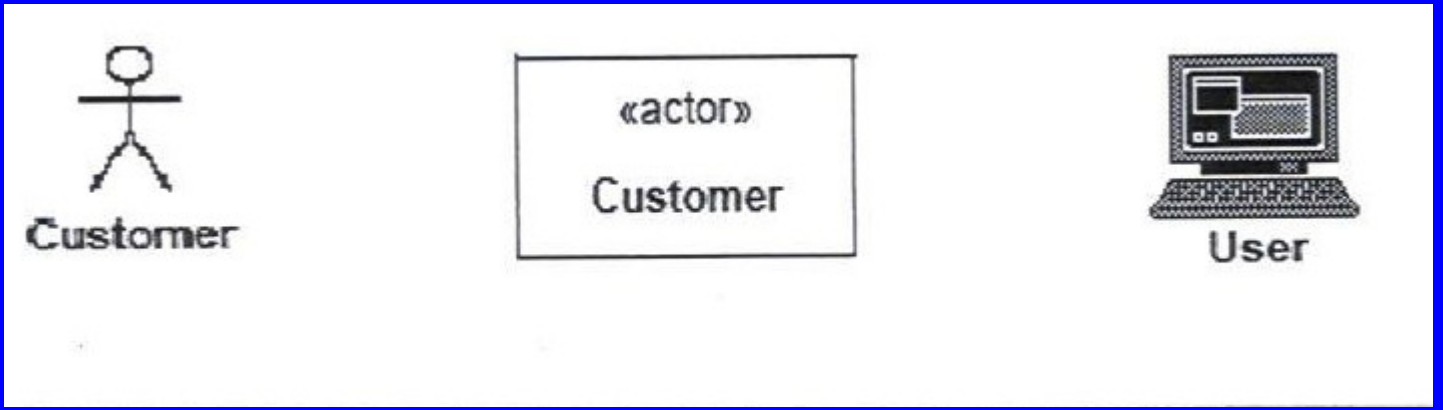
- **À quoi cela sert 3/3**
  - **Deux éléments principaux :**
    - **Les acteurs**
    - **Les cas d'utilisation**

Cas d'utilisation

- **Acteurs de cas d'utilisation 1/3**
  - **Éléments extérieurs interagissant avec le système**
    - **Personne physique, process, serveur, machine**
    - **Définissent les frontières du système**
  - **Deux types**
    - **Primaire s'ils activent un cas d'utilisation**
    - **Secondaire s'ils participent**
  - **Possèdent :**
    - **Un nom, une description, un id**

Cas d'utilisation

- **Acteurs de cas d'utilisation 2/3**
  - **Une abréviation**
  - **Représentation :**



Id	Abréviation	Nom	Description
1	#CUST	Customer	Personne se connectant au site pour acheter un produit

Cas d'utilisation

- **Acteurs de cas d'utilisation 3/3**

Et si je définis plus d'acteurs qu'il n'en faut ?

Un acteur est justifié par le fait qu'il participe à un cas d'utilisation

À ce stade, le risque est d'en oublier, donc autant en définir le plus possible et de les filtrer après

Cas d'utilisation

- **Liste de cas d'utilisation 1/4**
  - **Après avoir trouvé les acteurs : faire une liste des cas d'utilisation**
    - **Cette liste doit être exhaustive**
    - **Si elle est longue, il faut regrouper les cas d'utilisation par thème**
  - **Pour établir cette liste, tous les aspects du système doivent être pris en compte**
    - **Administration, backup**
    - **Surveillance**
    - **Rapports**

Cas d'utilisation

- **Liste de cas d'utilisation 2/4**
  - **Cette liste doit être lisible**
  - **Son objectif : fournir en une lecture simple une vue complète du système**
  - **Dans la liste, chaque cas d'utilisation possède**
    - **Un Id**
    - **Un acteur principal**
    - **Un nom**
    - **Une description sommaire**
  - **La Liste peut être établie à partir des exigences**

Cas d'utilisation

- **Liste de cas d'utilisation 3/4**

Id	Acteur	Nom	Description
1	#CUST	Connection	Le client se connecte au site
2	#CUST	Recherche rapide	Le client effectue une recherche de produit en saisissant un mot clé

  - **La liste contient les évolutions possibles, probables, improbables, ce qui va dépendre de clients spécifiques**

Cas d'utilisation

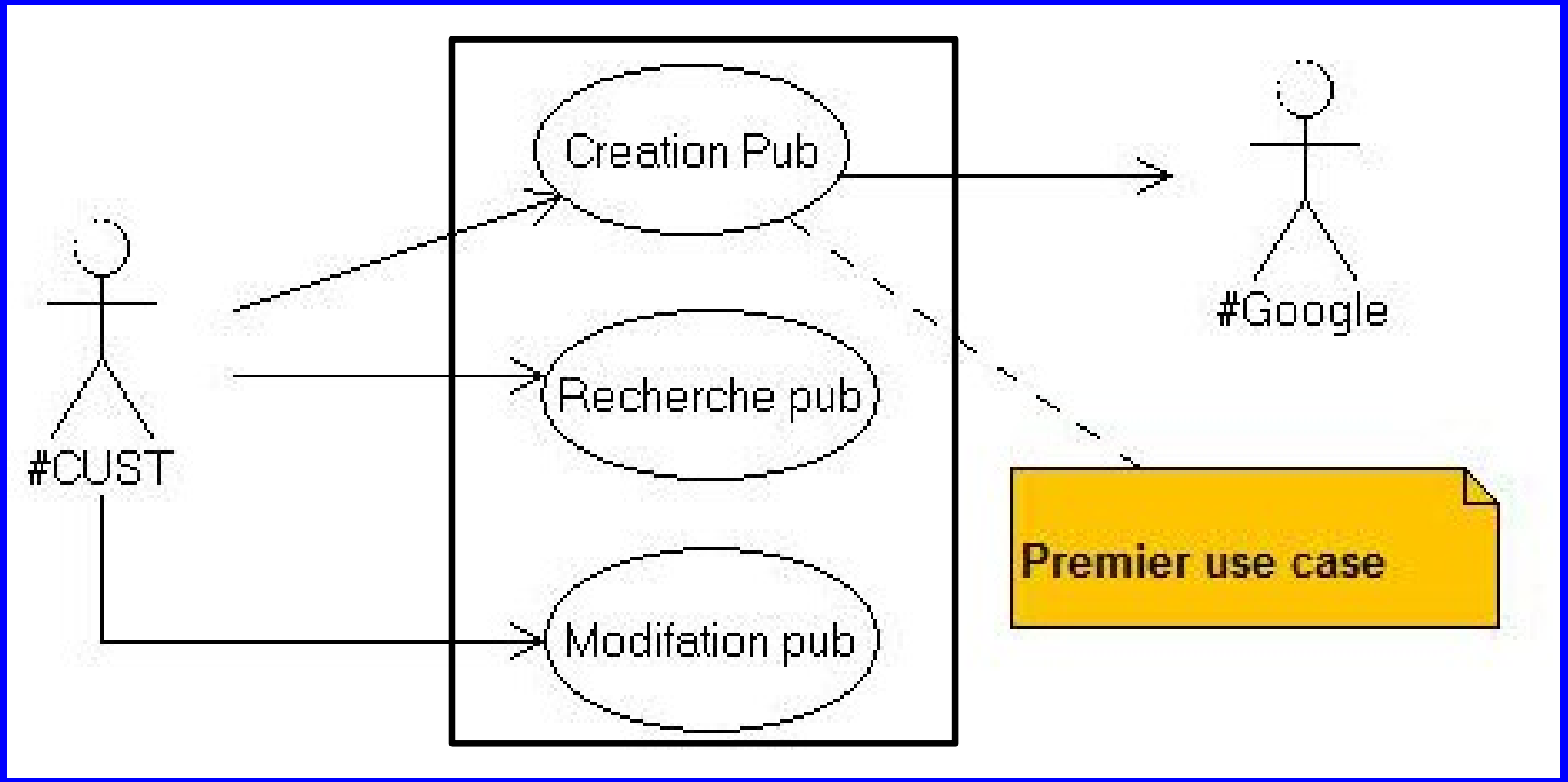
- **Liste de cas d'utilisation 4/4**

Les clients non seulement on ne comprend pas ce qu'ils veulent, mais en plus il faut imaginer ce qu'ils ne disent pas ?

Si l'on veut que le soft puisse évoluer dans le futur, il faut l'avoir anticiper ...

Cas d'utilisation

- **Description des cas d'utilisation 1a/4**



Cas d'utilisation

– Description des cas d'utilisation 2/4

USE·CASE·DÉTAILLÉ·«CRÉATION·PUB»¶	
Référence¶	5.2.1¶
Justification·¶	Un·client·se·connecte·au·site·pour·créer·sa·publicité¶
Description¶	1.→#CUST·visualise·la·page·d'accueil·du·site¶ 2.→#CUST·s'identifie¶ 3.→#CUST·click·sur·le·lien·«·créer·une·publicité·»¶ 4.→#CUST·saisie·le·texte·de·sa·publicité¶ 5.→#CUST·upload·une·bannière¶ 6.→#CUST·saisie·la·localisation·du·lieu·d'affichage¶ 1.→#CUST·se·déconnecte¶
Acteurs¶	#CUST¶
Use·cases·inclus¶	N/A¶
Déclenchement¶	Après·avoir·payé·son·adhésion·#CUST·saisie·une·publicité¶
Pré-conditions¶	Le·site·est·opérationnel·et·#CUST·est·correctement·enregistré¶
Post-conditions¶	La·publicité·est·stockée¶
Scénario·(s)¶	Erreur·de·saisie,·taille·de·la·bannière·incorrecte,·pertes·de·connexion¶
Note¶	N/A¶

Cas d'utilisation

- **Description des cas d'utilisation 3/4**
  - **Un cas d'utilisation contient plusieurs scénarios**
    - **Le scénario nominal et tous les scénarios d'exception ou conditionnels**
  - **Description textuelle « non normalisée »**
  - **Un des objectifs est de déduire des exigences**

N/A	E_098	Besoins sources : UC 4.1.2.1
L'API doit fournir une méthode de destruction des objets métiers		
Description des tests : Réalisation d'un programme qui se connecte à une base contenant des objets métiers et qui détruit un des objets métiers.		
Référence des tests : TST_15		

Cas d'utilisation

- **Description des cas d'utilisation 4b/4**

Les exigences viennent des cas d'utilisation, mais s'ils sont mal rédigés ?

Les exigences risquent d'être fausses ...

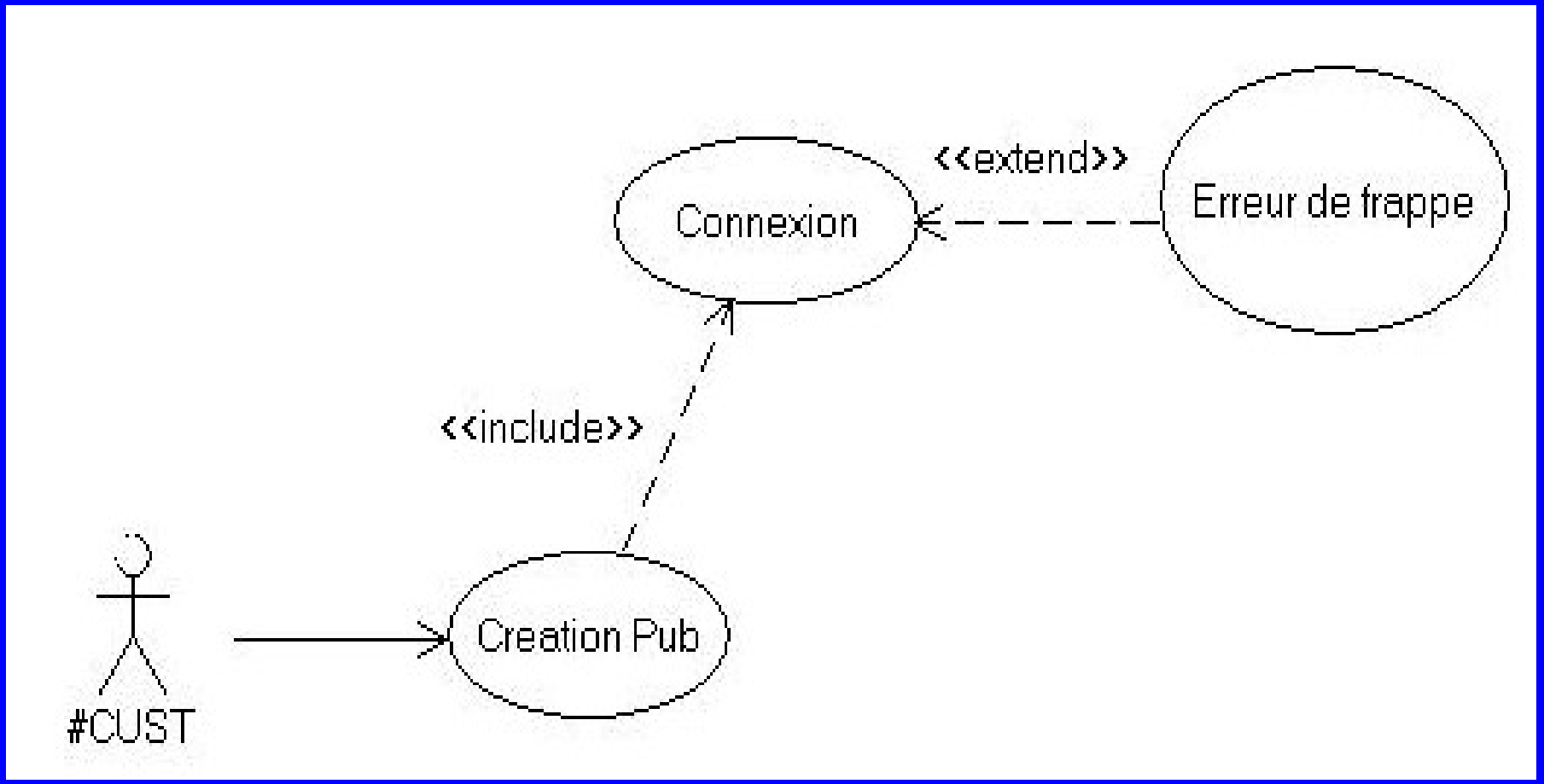
Il faut bien les revoir entre pairs alors ?

Cas d'utilisation

- **Relations entre cas d'utilisation 1/3**
  - **Un cas d'utilisation peut :**
    - **Inclure d'autres cas d'utilisation**
    - **Étendre un autre cas d'utilisation**
      - **Présenter les alternatives**
    - **Hériter d'autres cas d'utilisation**
    - **Appartenir à un package**

Cas d'utilisation

- **Relations entre cas d'utilisation 2/3**



Cas d'utilisation

- **Relations entre cas d'utilisation 3/3**

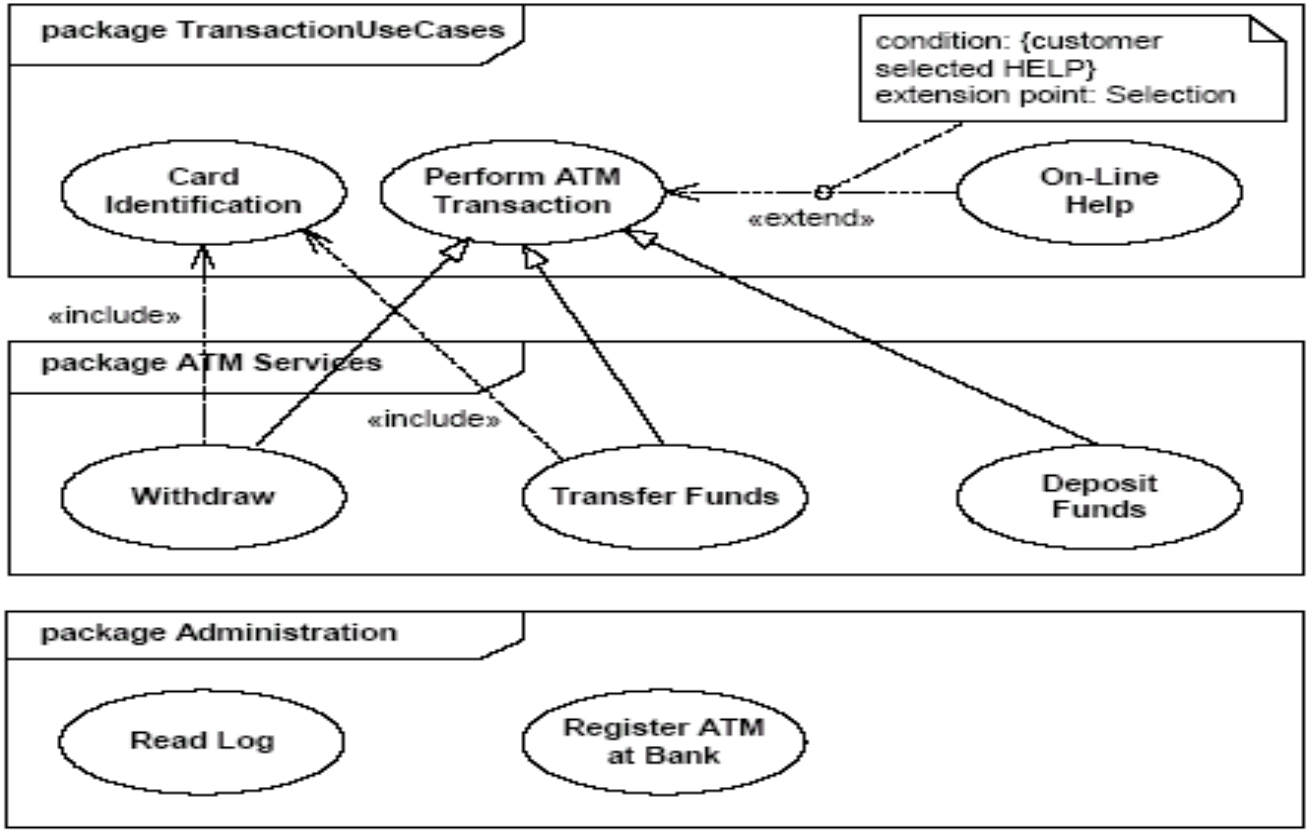


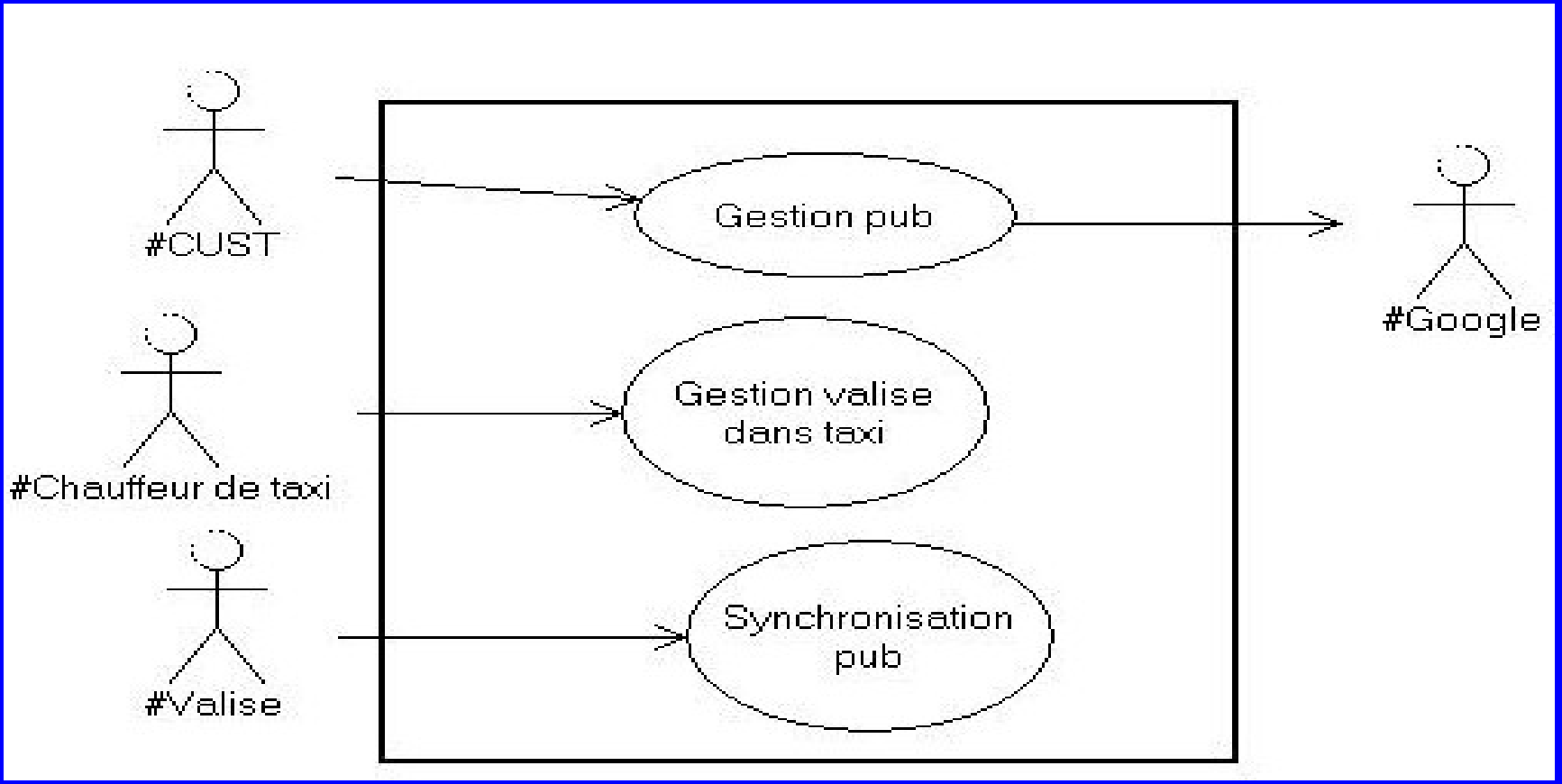
Figure 16.7 - Example of use cases owned by various packages

Cas d'utilisation

- Diagramme de contexte 1/2
  - Donner une vue de l'application en un schéma
  - Peut ne pas être complet
  - Ne fait pas partie d'UML
    - Mais existe dans beaucoup de méthode
    - Une possibilité en UML
      - Détourner un diagramme de use cases

Cas d'utilisation

- **Diagramme de contexte 1/**



Cas d'utilisation

- **Difficultés des cas d'utilisation 1/1**
  - Les cas d'utilisation sont verbeux
    - Il y en a beaucoup
  - Les alternatives ne sont pas simples à décrire
  - Les erreurs et exceptions sont difficiles à prendre en compte
  - Comprendre le métier des utilisateurs n'est pas aisé
  - Il y a le risque de coder en procédural

Cas d'utilisation

- Critères d'évaluation 1/1
  - La liste est-elle exhaustive ?
    - Comprend-t-elle les évolutions, l'avenir ?
  - Les cas principaux ont ils été décrits ?
  - Les utilisateurs ont-ils été impliqués ?
    - Ont-ils compris les descriptions ?
  - Toutes les parties prenantes ont-elles été consultées ?
  - Le vocabulaire utilisé est-il précis, clairement défini ?

Cas d'utilisation

- QCM 1/1
  - Les cas d'utilisation ne servent-ils qu'aux spécifications YIN ?, à quoi d'autres ?
  - Les cas d'utilisation concernent-ils d'autres fonctionnalités que celles attendues ?
  - Quels sont les éléments des cas d'utilisation le plus importants ?
  - Citer des éléments difficilement pris en compte par dans les cas d'utilisation ?
  - Faut-il décrire tous les cas d'utilisation ?

Diagrammes d'activité

Diagrammes comportementaux

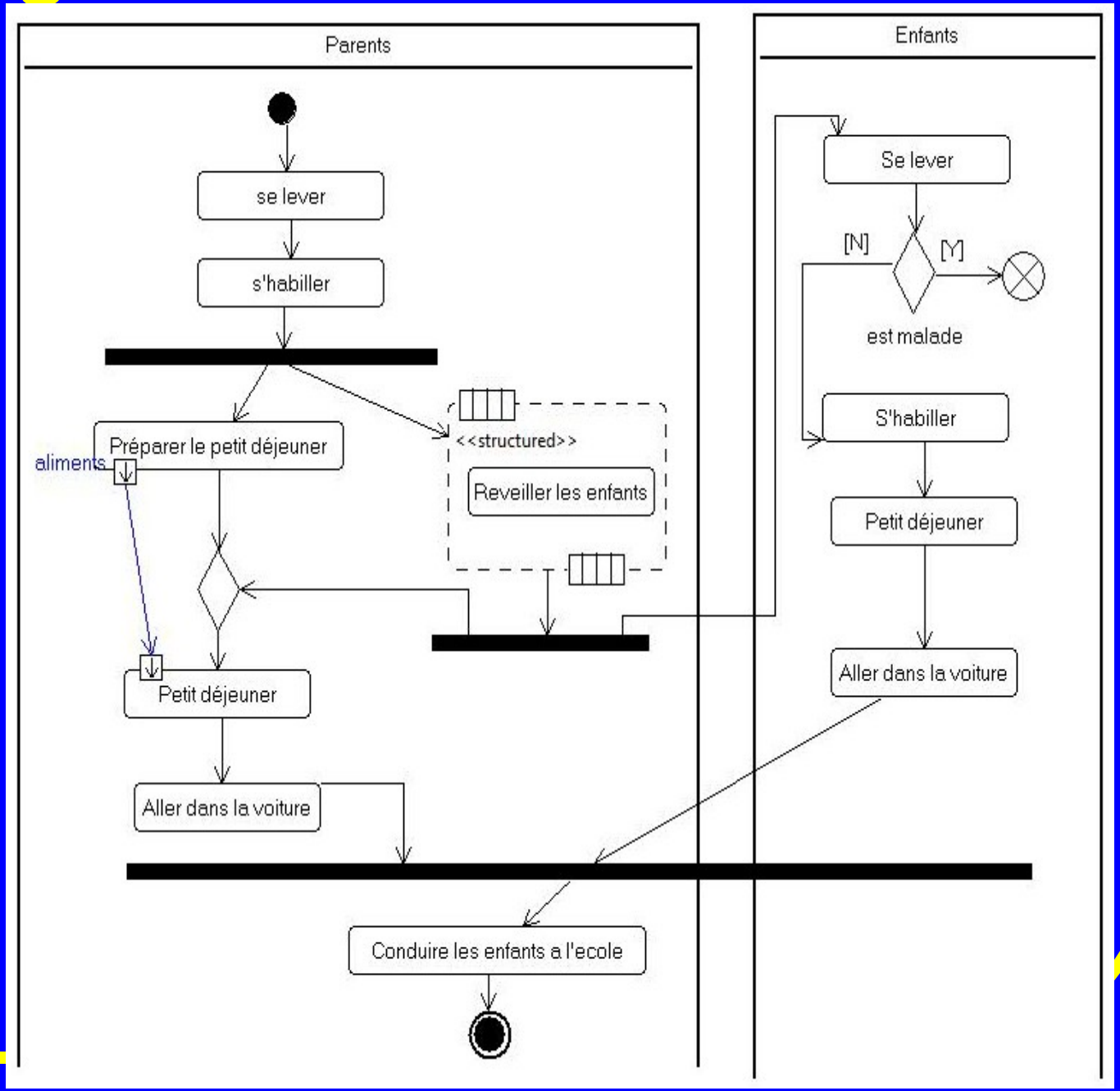
- Plan
  - Utilisation
  - Diagramme d'activités
  - Critères d'évaluation
  - QCM

Diagrammes d'activité

- **Utilisation 1/1**
  - **Modélise:**
    - **Les cas d'utilisation**
    - **Action entre objets**
    - **Workflow**
    - **Processus d'entreprise ...**
  - **Ressemble le plus aux organigrammes**
  - **Ils sont simples à comprendre**
  - **En général, ils sont appréciés par tous les niveaux hiérarchiques**

Diagrammes d'activité

- **Diagramme D'activités 1/5**

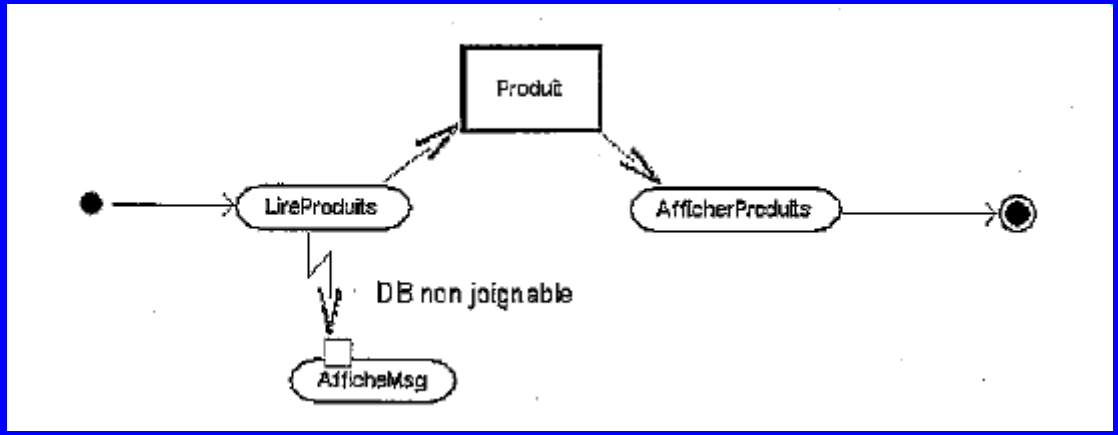


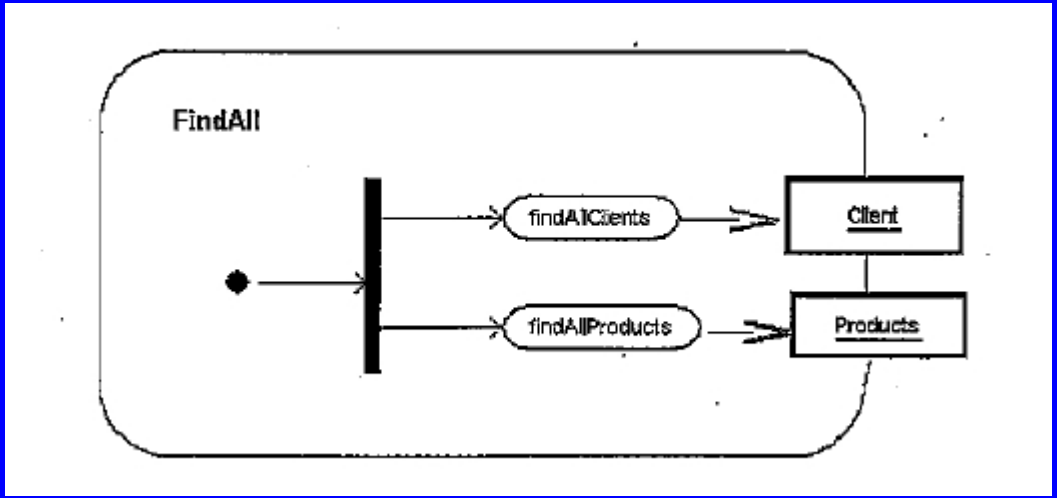
Diagrammes d'activité

- **Diagramme d'activités 2/5**
  - **Qu'est-il possible de spécifier de plus ?**
    - **Des flux d'objets**
    - **Des paramètres**
    - **Des actions subsidiaire**
    - **Des gestions d'exceptions**
    - **Des événements**

Diagrammes d'activité

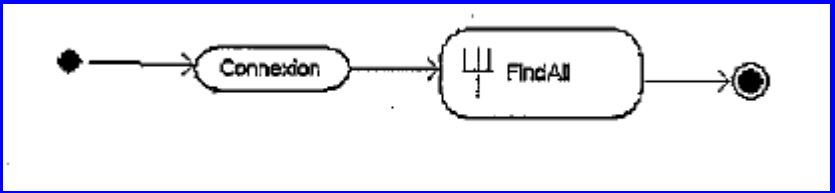
- **Diagramme d'activités 3/5**

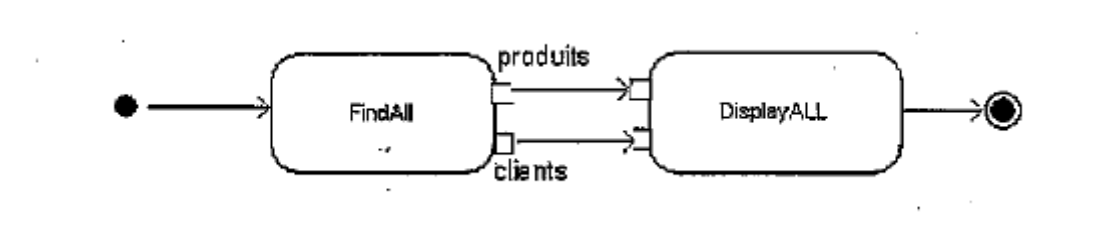




Diagrammes d'activité

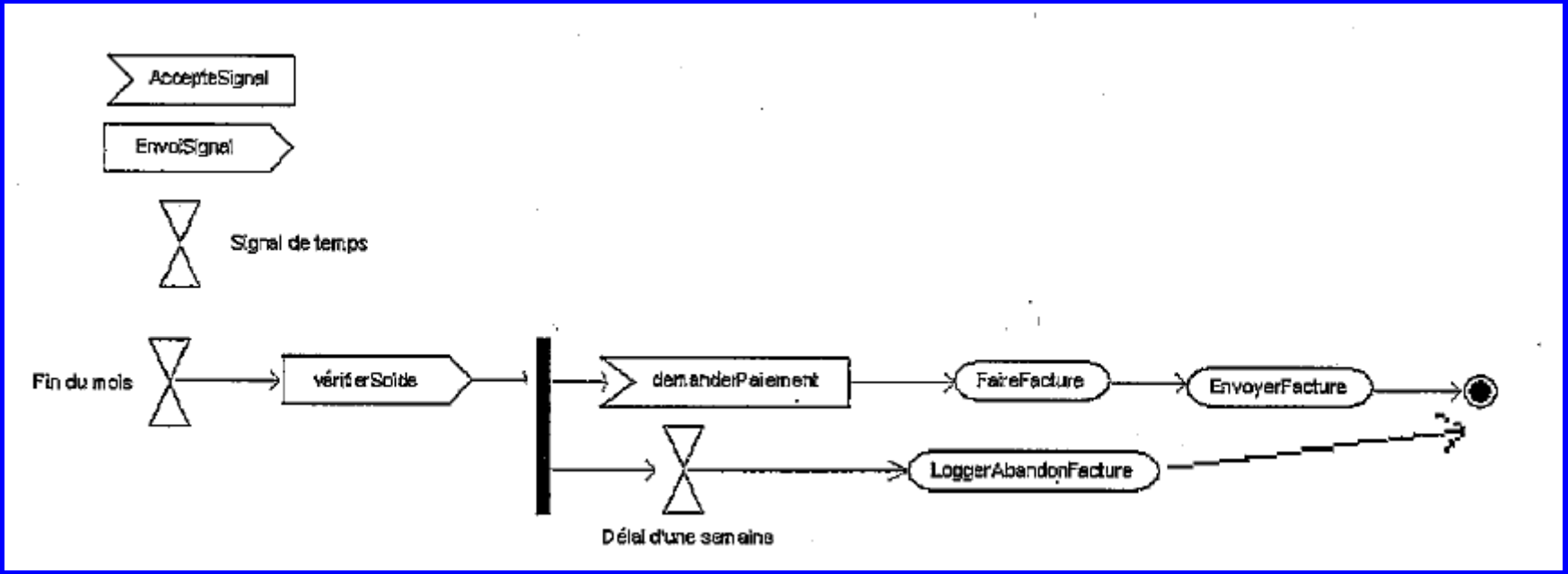
- **Diagramme d'activités 4/5**





Diagrammes d'activité

- **Diagramme d'activités 5/5**



Diagrammes d'activité

- **Critères d'évaluation 1/1**
  - **Les participants ont-ils été correctement identifiés ?**
  - **Les principaux uses cases ont-ils été modélisés ?**
  - **Les données et actions principales sont-elles identifiées ?**
  - **Sont-ils suffisamment complets ?**
  - **Sont-ils logiquement cohérents ?**

Diagrammes d'activité

- QCM 1/1
  - Que peux représenter un diagramme d'activité ?
  - Qu'est-il intéressant de visualiser sur un diagramme d'activité ?
  - Comment se concrétise dans le code les actions et les données des actions ?
  - Tous les uses cases doivent-ils donner lieu à un diagramme d'activité ?

Diagramme de classes

Diagramme de classes

- Plan 1/2
  - Introduction
  - Exemple de classes
  - Héritage
  - Encapsulation
  - Dépendance
  - Association
  - Interface
  - Contraintes

Diagramme de classes

- Plan 2/2
  - Template
  - Règles de design
  - Critères d'évaluation
  - QCM

Diagramme de classes

- **Introduction**

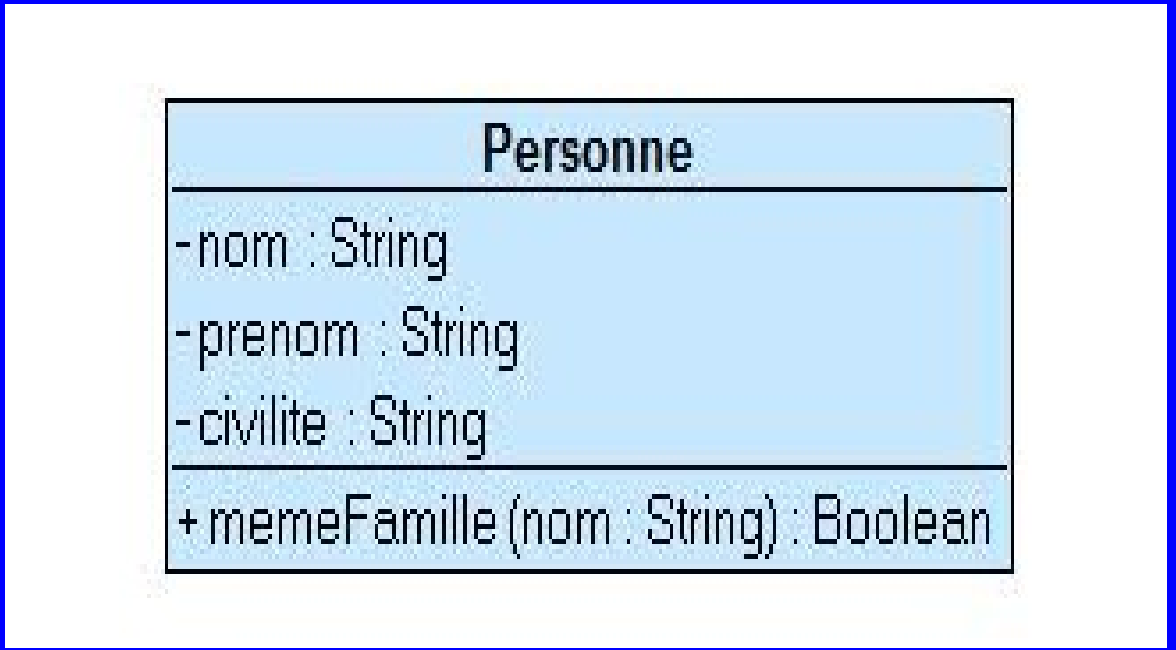


Diagramme de classes

- **Introduction 1/2**
  - **Diagramme le plus courant**
    - **Peut contenir toutes les caractéristiques des classes**
    - **Peut être partiel**
  - **Comment trouver les classes ?**
    - **Classes « Métiers »**
      - **Extraire des cas d'utilisation les concepts, entités, manipulées par le système**
      - **Regarder les opérations effectuées sur ces objets avec le regard des acteurs**
      - **Les classes peuvent être les noms communs des cas d'utilisation, les verbes les méthodes**

Diagramme de classes

- **Introduction 2/2**
  - **Comment trouver les classes ?**
    - **Classes « techniques »**
      - **En réalisant les diagrammes séquences des use case**
        - **Précise les premiers diagrammes de classes**
    - **Doivent refléter les diagrammes d'activités**
      - **Les flux de données se reflètent dans les attributs des classes et les paramètres des méthodes**
      - **Les activités se reflètent dans les méthodes**

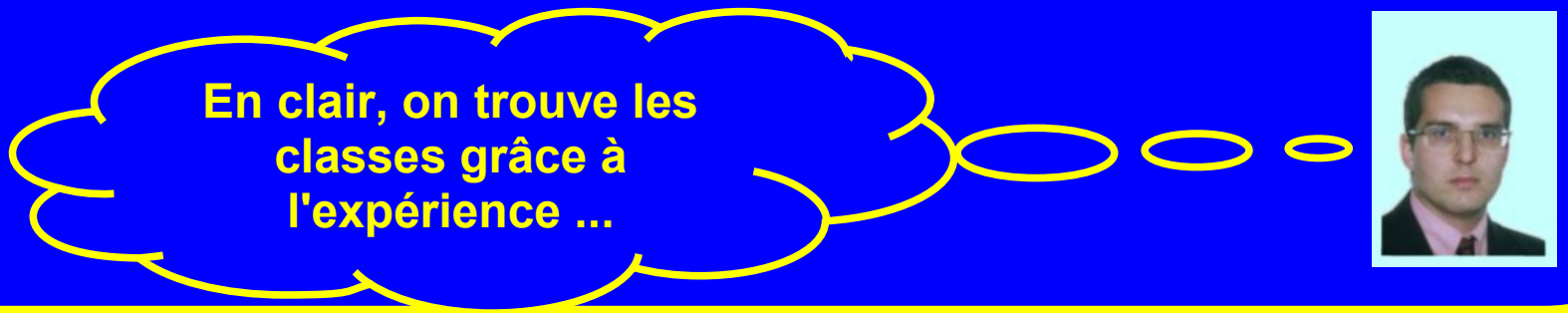


Diagramme de classes

- **Pas définition de ce qu'est une classe**
  - **Une entité associant des données et les opérations faites sur ces données …**
    - **Accent mis sur les données membres**
  - **Une modélisation du monde réel …**
    - **Du monde du projet**
    - **Accent mis sur la modélisation**
  - **Un nouveau type**
    - **Accent mis sur la programmation**

La plus intéressante de mon point de vue

Diagramme de classes

- **Exemple de classes 1/1**

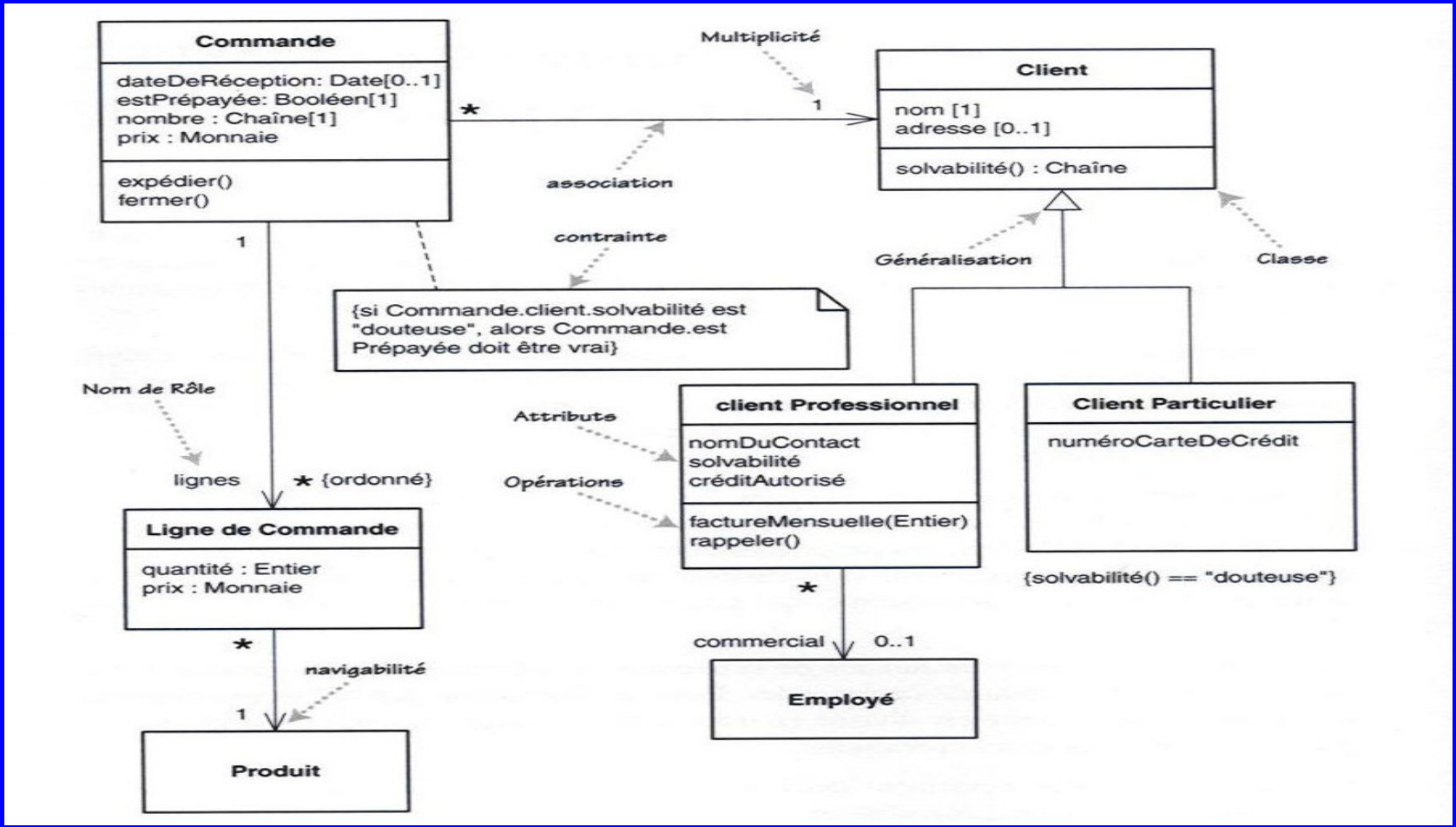


Diagramme de classes

- **Héritage 1/3**
  - **Les deux notions de la POO :**
    - **Polymorphisme**
    - **Encapsulation**
  - **L'héritage est, entre autres, un moyen pour réaliser le polymorphisme**

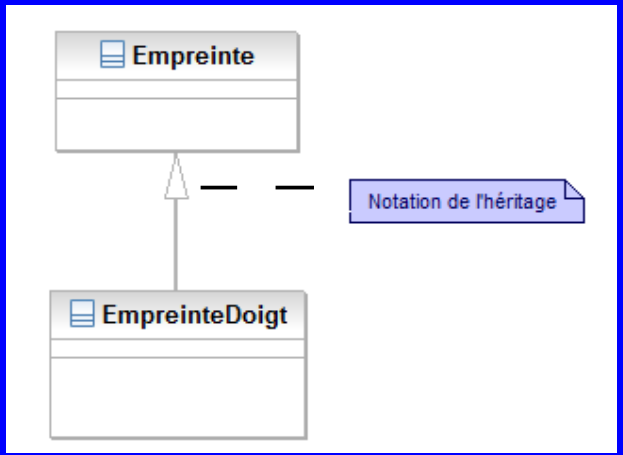


Diagramme de classes

- **Héritage 2/3**
  - **La classe fille hérite des propriétés et méthodes**
    - **Mais n'a pas vu sur tout**
  - **Les classes mères et filles sont liées par une relation « est une sorte de »**
    - **Mais :**
      - **PoissonRouge est une sorte de Poisson**
      - **Totor est une sorte de poissonRouge**
      - **Est-ce vrai ?**
  - **Les héritages multiples et dynamiques sont dangereux**

Diagramme de classes

- **Héritage 3/3**
  - **L 'héritage est une relation très forte entre deux classes**
    - **À utiliser à bon escient**
  - **Ne pas utiliser par « confort »**
  - **Ne pas utiliser si cela cache une taxinomie**

Diagramme de classes

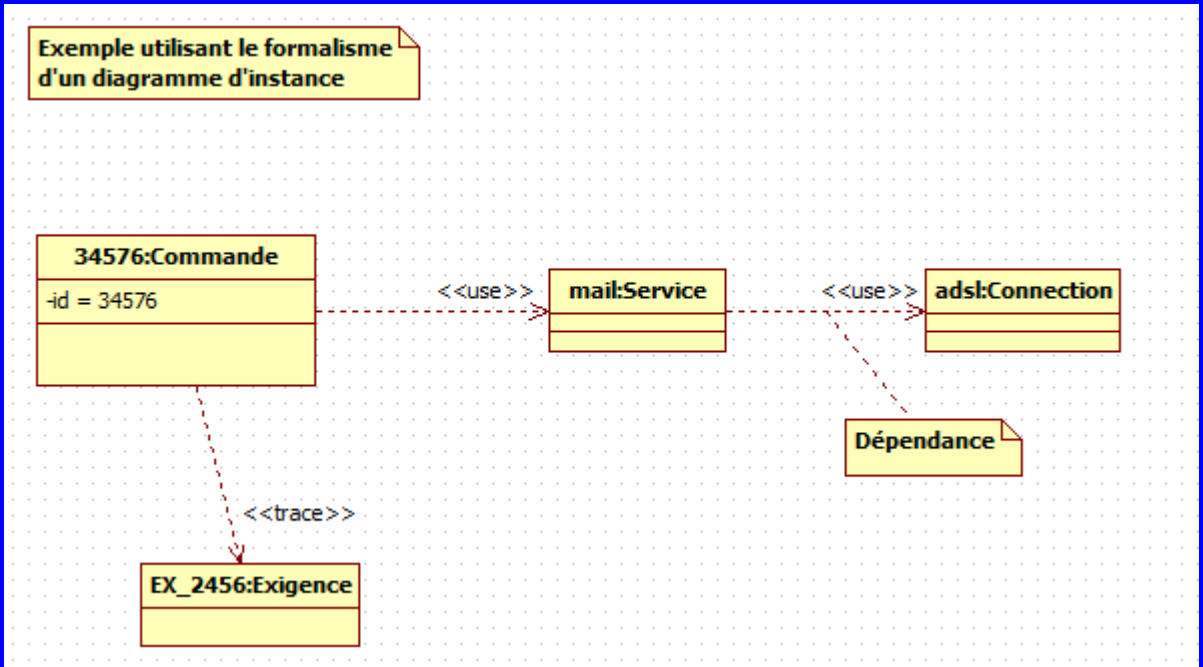
- **Encapsulation 1/2**
  - **Deuxième notion fondamentale de la POO**
  - **Synonyme de visibilité**
  - **Permet de définir une « interface à tous les niveaux »**

Diagramme de classes

- **Encapsulation 2/2**
  - **Attributs/méthodes publics : +**
  - **Attributs/méthodes protégés : #**
  - **Attributs/méthodes privés : -**
  - **Attributs/méthodes packages : ~**
  - **Suivant les langages il existe d'autre mécanismes**
  - **Par défaut les attributs sont privés et les méthodes publiques**

Diagramme de classes

- **Dépendance 1/2**



  - **Assez peu utilisé et pourtant ...**

Diagramme de classes

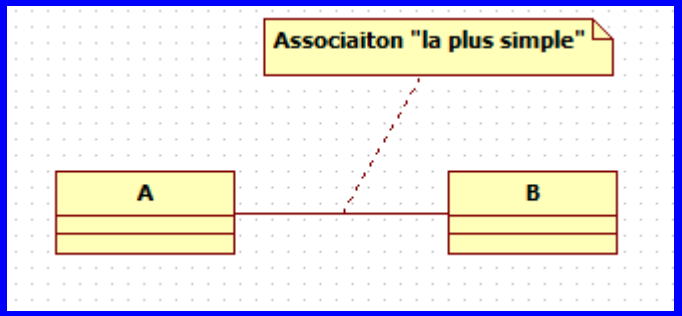
- **Dépendance 2/2**

C'est peu utilisé, pourtant il est très important pour faire évoluer un logiciel d'avoir une vue sur qui dépend de qui

Il y a peut-être des outils pour cela ou cela intervient plus pour les packages ?

Diagramme de classes

- **Association 1/6**
  - Représente un lien entre deux classes



  - Peut avoir un nom

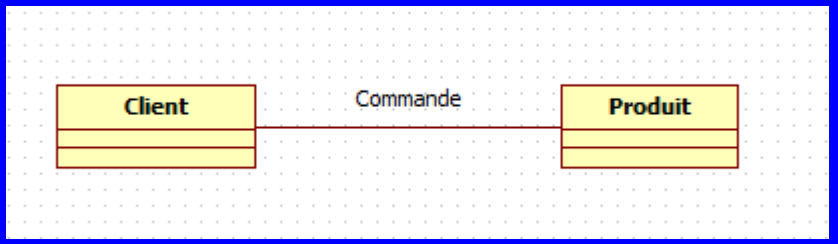
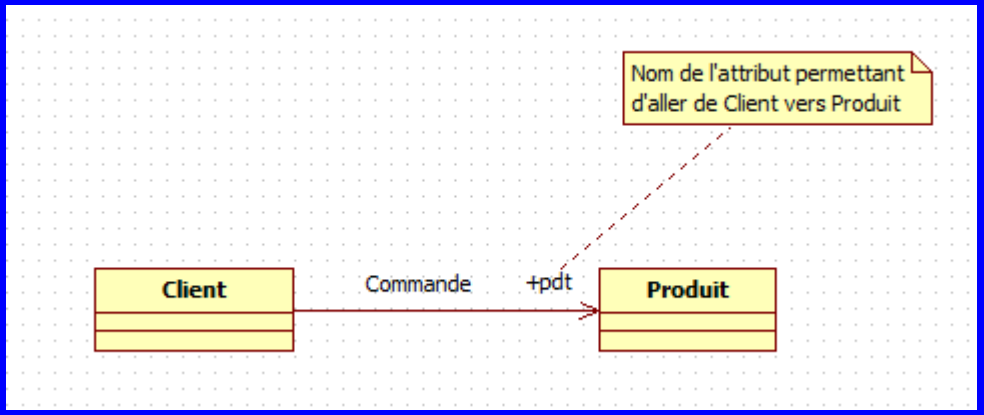


Diagramme de classes

- **Association 2/6**
  - **Peut avoir une « navigabilité »**
    - **Si une association est navigable, il y a un attribut/méthode permettant d'aller de l'une à l'autre**



  - **Peut avoir une « non navigabilité »**

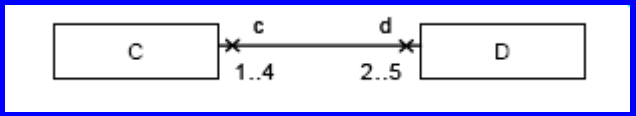
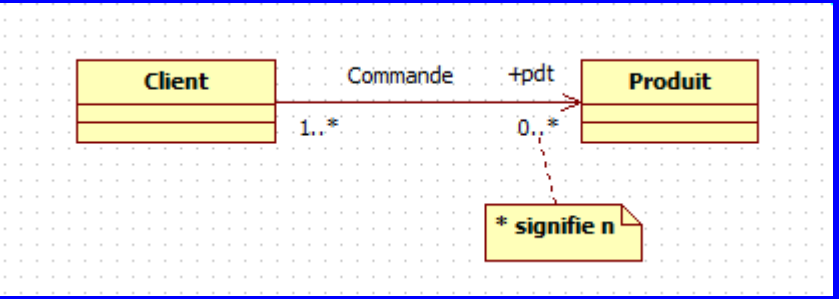


Diagramme de classes

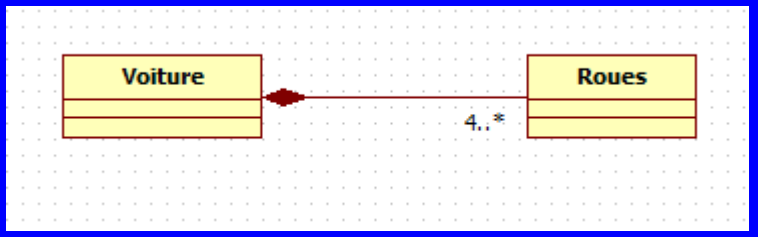
- **Association 3/6**
  - **Peut avoir une numéricité**



  - **Deux associations particulières :**
    - **L'agrégation qui est fortement déconseillée**
    - **La composition qui est une forme forte d'association**

Diagramme de classes

- **Association 4/6**
  - **Composition**



  - **La numéricité du coté du diamant est zéro ou un**
  - **Une instance de roue n'appartient qu'à une instance de voiture**
  - **La voiture est responsable de l'instanciation/destruction des roues (cycle de vie)**

Diagramme de classes

- **Association 5/6**
  - **Classe d'association**
    - **Lorsque l'association porte des informations qui n'appartiennent à aucune des deux classes**

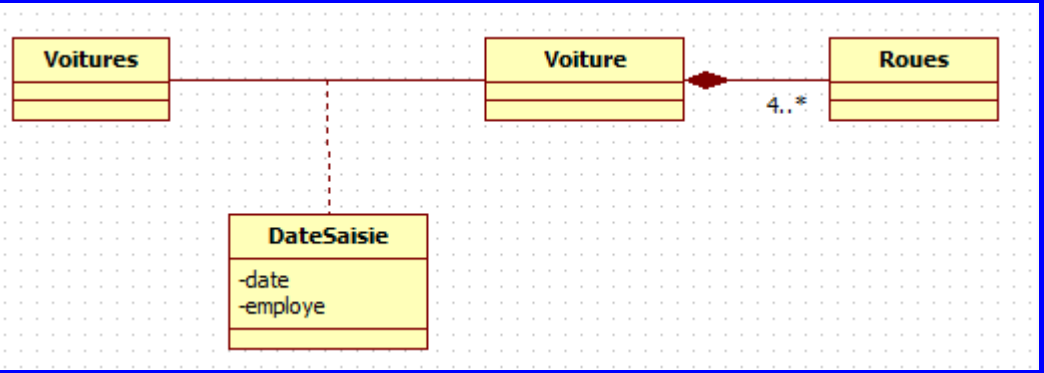


Diagramme de classes

- **Association 6/6**
  - **Association qualifiée**
    - **Une clé permet de trouver les membres de l'association**
    - **La numéricité est alors 1**

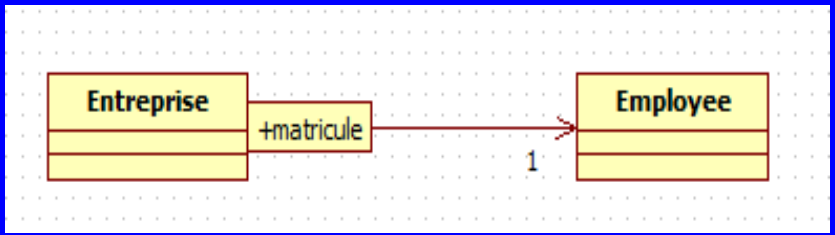


Diagramme de classes

- Interface 1/4
  - Définit une spécification, un contrat
    - Sans implémentation
    - Donc avec une documentation
  - Permet de fournir une API sans donner le code
  - Concept central dans les architecture COM, J2EE, moins en JEE
  - Java permet de les définir au niveau des packages
    - Le C++ aussi, mais de façon plus compliquée

Diagramme de classes

- **Interface 2/4**

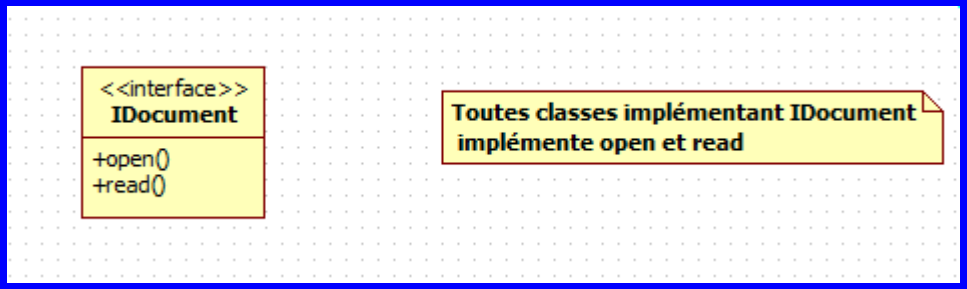
C'est vraiment un concept très important, vieux comme l'informatique, mais bien formalisé en UML

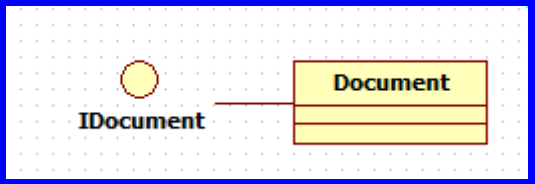
Cela mérite d'être travaillé alors

Super, je vais pouvoir définir des interfaces stables et choisir les implémentations que je veux

Diagramme de classes

- **Interface 3/4**





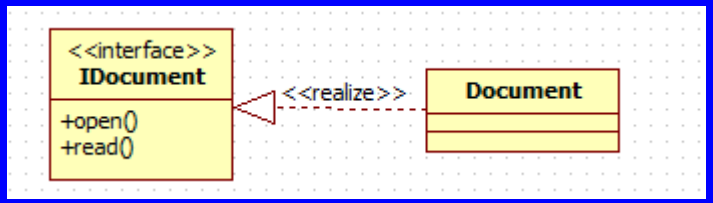


Diagramme de classes

- **Interface 4/4**

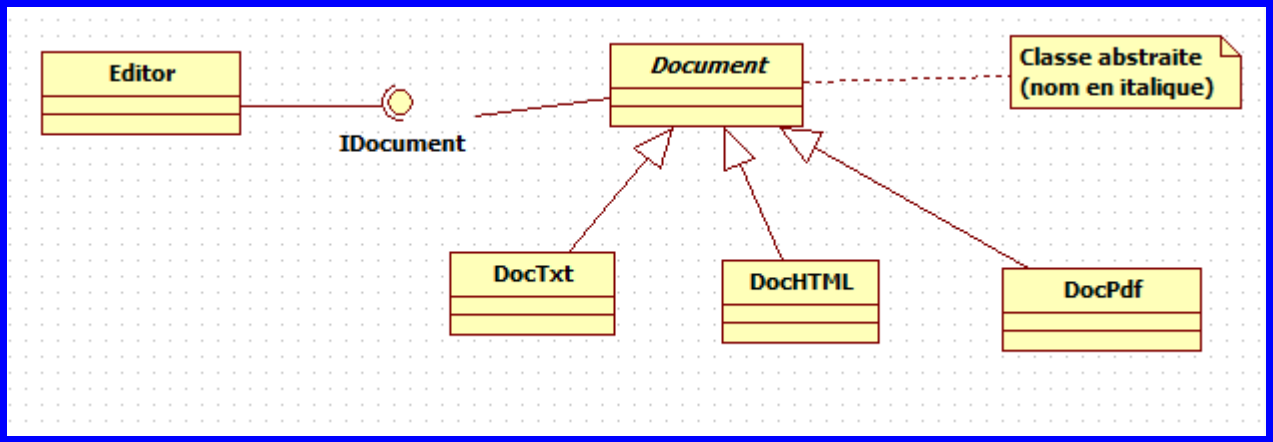


Diagramme de classes

- **Interface 4b/4**

Ce schéma est très important

Il permet l'évolutivité, le principe ouvert/fermé, la gestion des responsabilités, et la délégation ?

Et si on jouait à l'évolution ?

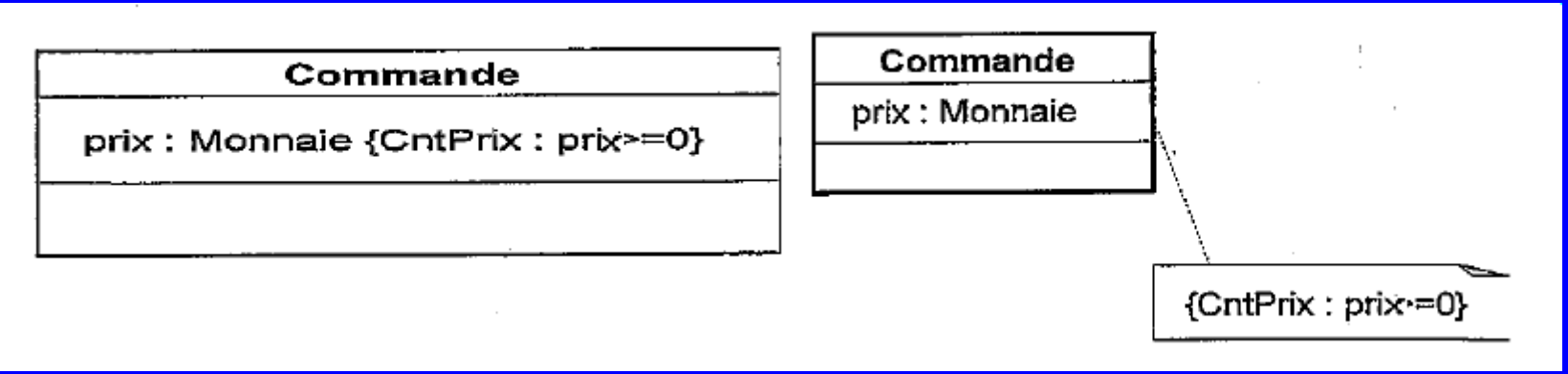
La délégation non

Diagramme de classes

- **Contraintes 1/3**
  - **Permettent d'indiquer des contraintes sur une classe de façon souple**
    - **Contraintes sur les valeurs d'attributs**
    - **Contraintes sur les associations**
    - **Contraintes sur les listes ...**
  - **Les contraintes peuvent être exprimées en langage naturel ou en OCL**
    - **En OCL, elles peuvent être traitées automatiquement**

Diagramme de classes

- **Contraintes 2/3**
  - **Syntaxe : { <nom contrainte> : <texte>}**



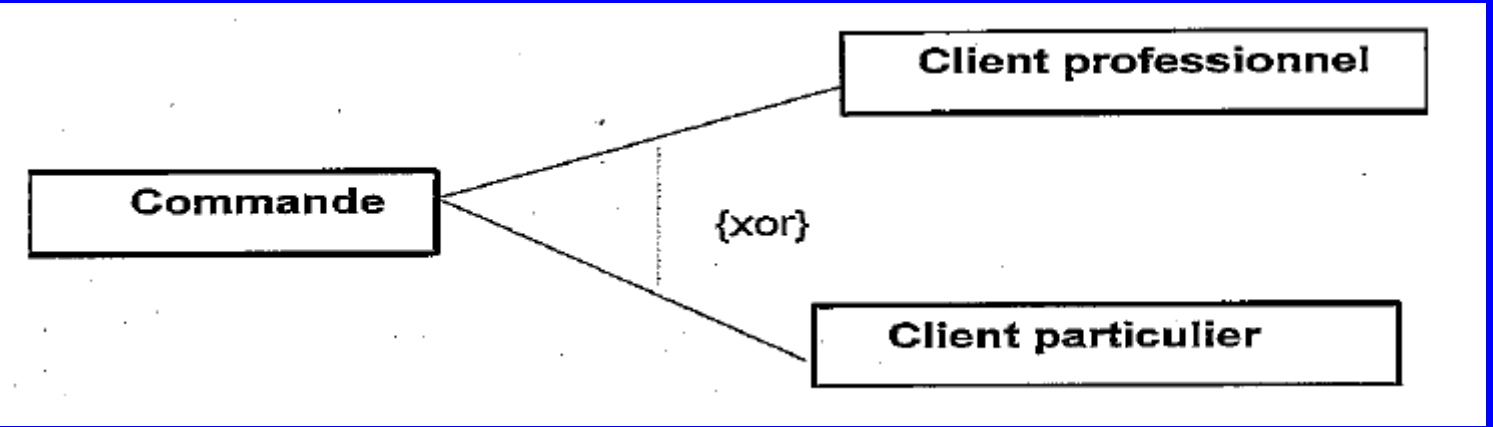


Diagramme de classes

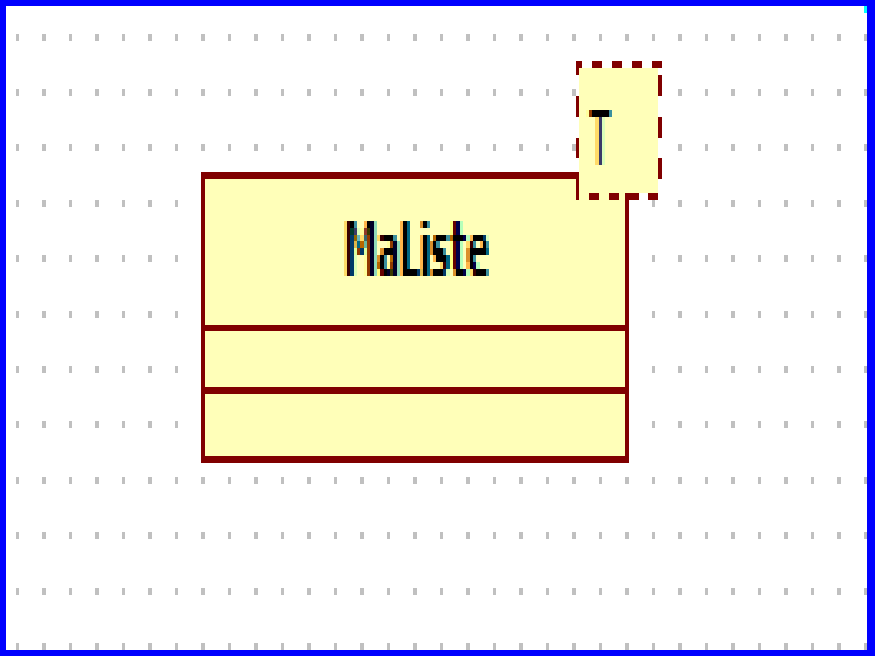
- **Contraintes 3/3**

Cela mériterait d'être plus utilisé, et cela le devient

Les compilateurs OCL n'ont pas « pris », mais ça arrive

Le DBC ce n'est pas des contraintes ?

Diagramme de classes



- **Template 1/1**
  - **Ou classe paramétrable**
  - **Est vraiment utile pour les listes**
    - **Conservation du type des éléments de la liste**
  - **Peut être redondant avec le concept d'interface**

Personnellement je n'aime pas les templates

Cela n'empêche pas de l'enseigner

Diagramme de classes

- **Règles de design 1/2**
  - **Chaque classe doit avoir une responsabilité**
  - **L'héritage et l'encapsulation doivent être utilisés correctement**
  - **Isoler ce qui peut changer, ce qui dépend des clients**
  - **Séparer la modélisation de la visualisation**
  - **Définir des interfaces stables**

Diagramme de classes

- **Règles de design 2/2**
  - **Les principes de la POO ont-ils été appliqués ?**
    - **« Cohésion forte, couplage faible »**
    - **« Principe ouvert/fermé »**
    - **« Déléguer chaque fois que cela est nécessaire »**

Diagramme de classes

- Critères d'évaluation 1/2
  - Les relations « est une sorte de » sont-elles correctes ?
  - Les noms des classes sont-ils signifiants ?
  - Les cas d'utilisation futurs ont-ils été utilisés ?
  - Y a-t-il beaucoup de dépendances entre les classes ?
  - L'héritage est-il trop/pas utilisé ?
  - Les interfaces sont-elles clairement définies ?
  - La traçabilité avec les CU est-elle assurée ?

Diagramme de classes

- **Critères d'évaluation 2/2**

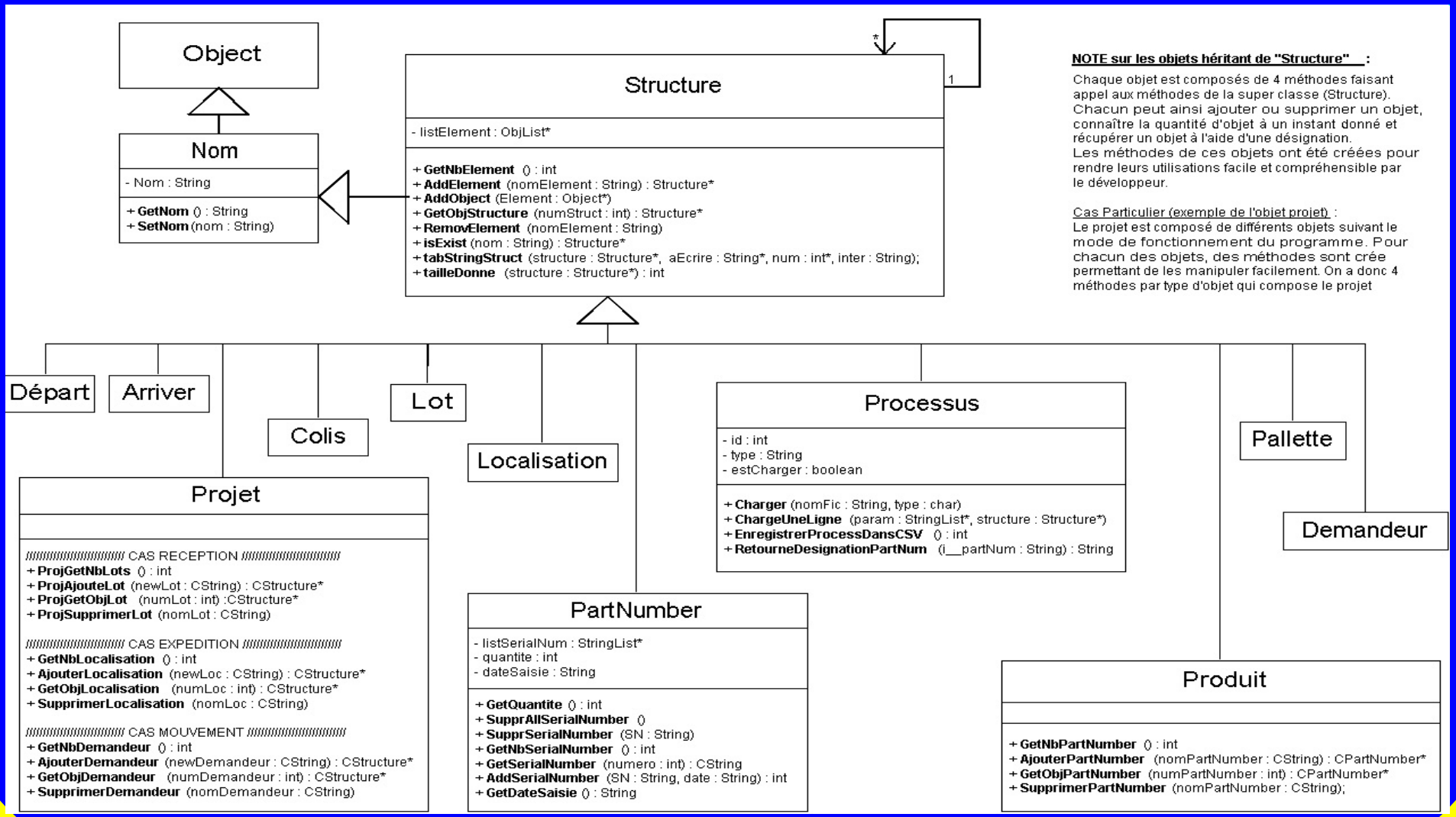


Diagramme de classes

- QCM 1/1
  - Quelle est la différence entre une composition et une association simple ?
  - Quelles sont les deux caractéristiques de la POO ?
  - Citer deux principes de POO ?
  - Quelle phrase doit lier une classe mère et une classe fille ?
  - Dans quel cas ne pas utiliser l'héritage ?

Diagrammes de séquences et d'états

Diagrammes comportementaux

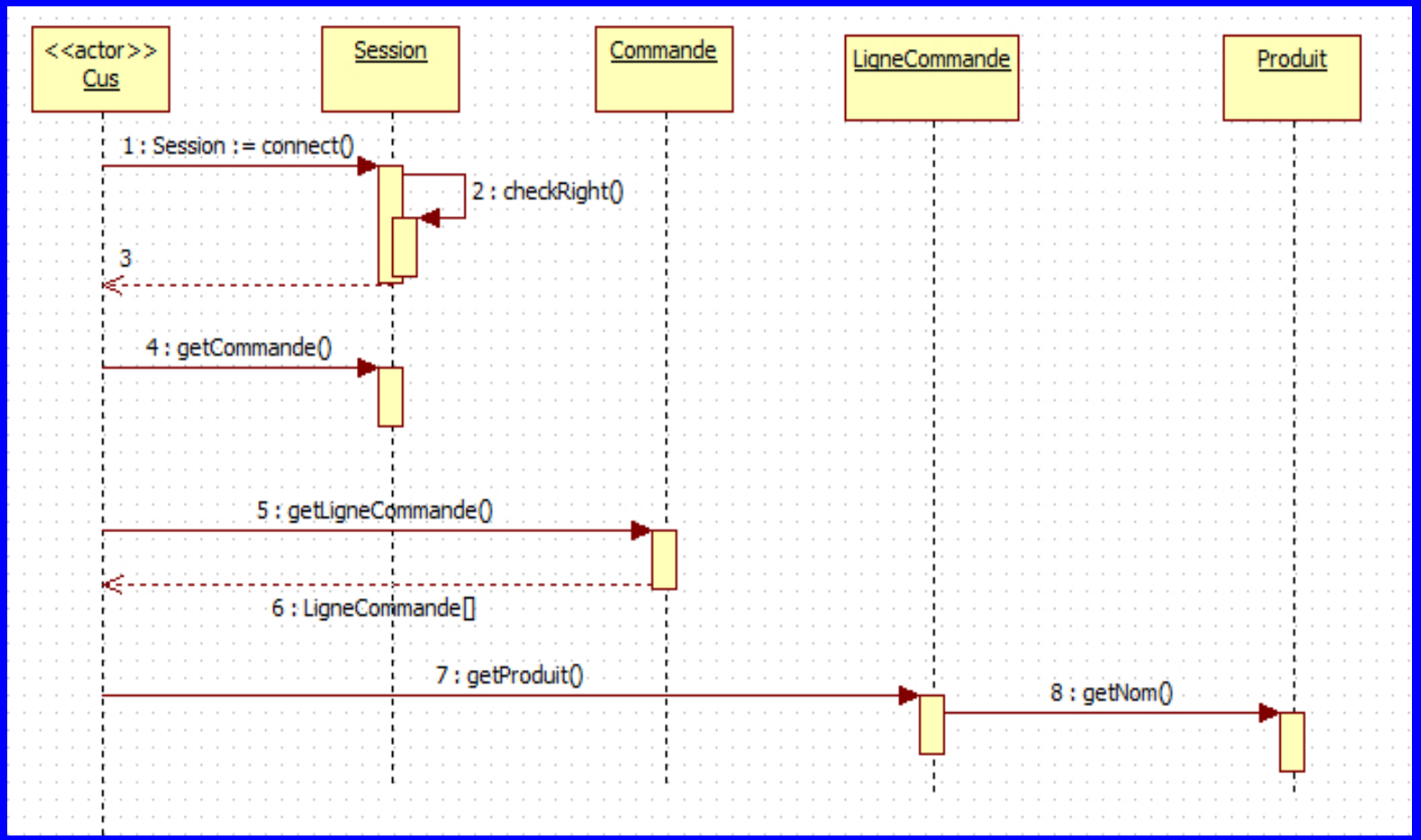
- Plan
  - A quoi servent-ils ?
  - Diagramme de séquences
  - Diagramme de machines états
  - Critères d'évaluation
  - QCM

Diagrammes comportementaux

- À quoi servent-ils ? 1/1
  - Le schéma de classes fournit « vue statique »
    - Il montre les classes et leurs liens
    - Il ne fournit aucune information aucune information sur les interactions, les différents appellent de méthodes
  - C'est le rôle des diagrammes comportementaux
  - UML ne possède pas d'organigramme en tant que tel
    - Rien ne vous empêche d'en faire ...

Diagrammes comportementaux

- **Diagrammes de séquences 1/7**

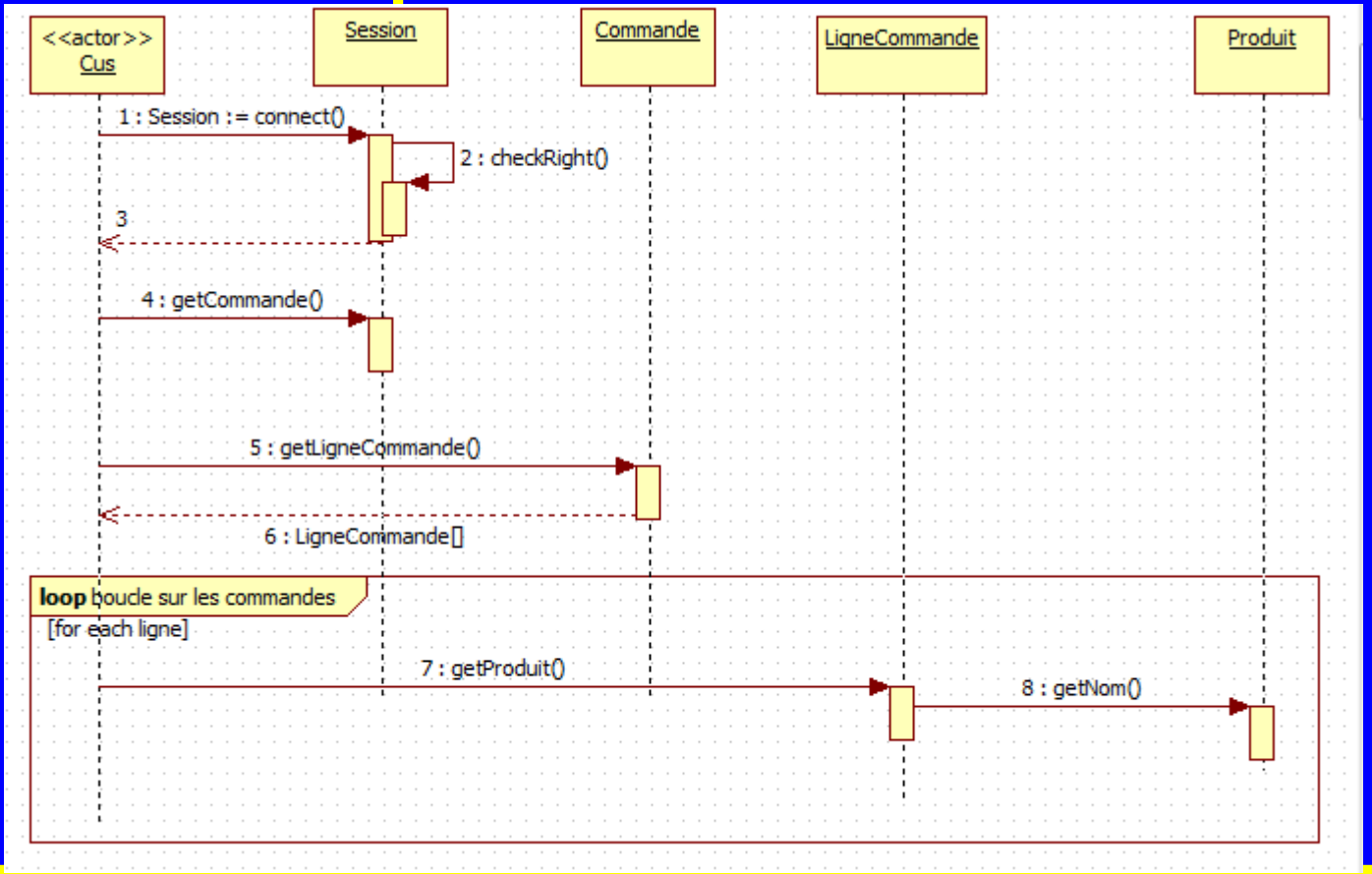


Diagrammes comportementaux

- **Diagramme de séquences 2/7**
  - **Que nous faut-il de plus ?**
    - **Tests**
    - **Boucle**
    - **« Séquence réutilisable »**
    - **Création/destruction d'objet**
    - **Éléments de gestion des threads**
    - **Contrainte temporelle**

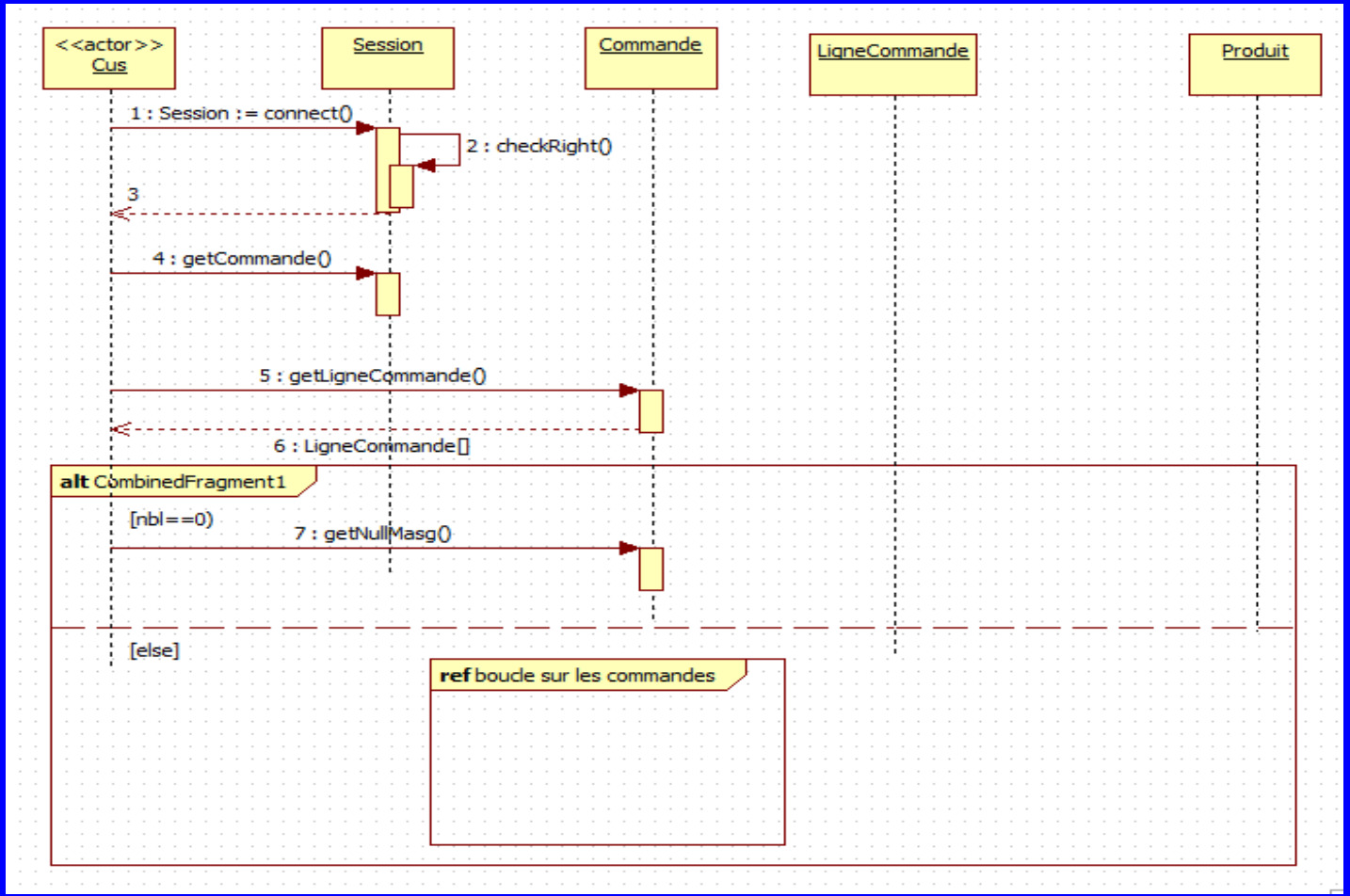
Diagrammes comportementaux

- **Diagramme de séquences 3/7**
  - **Détaillent les premiers schémas de classes**



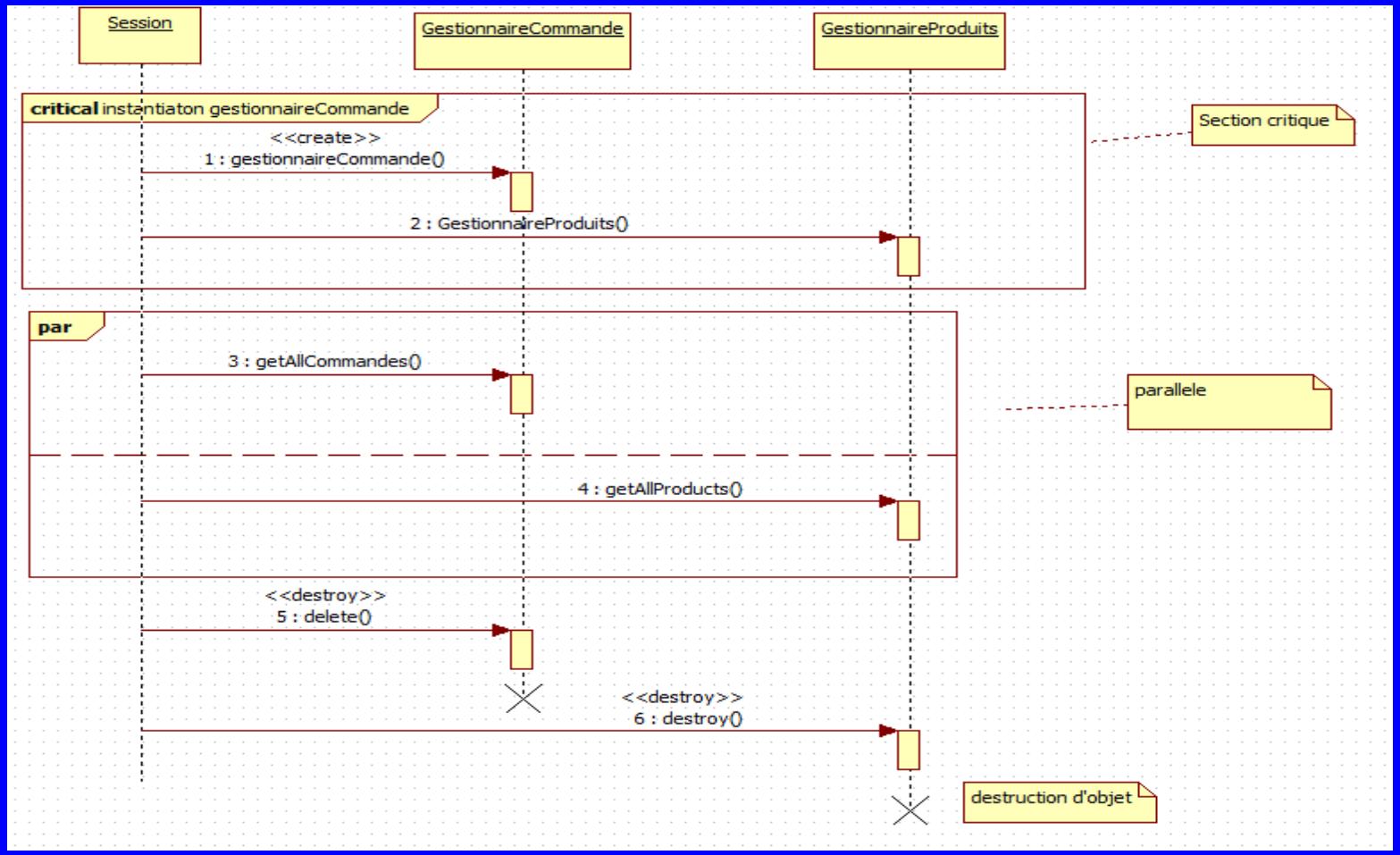
Diagrammes comportementaux

- **Diagramme de séquences 4/7**



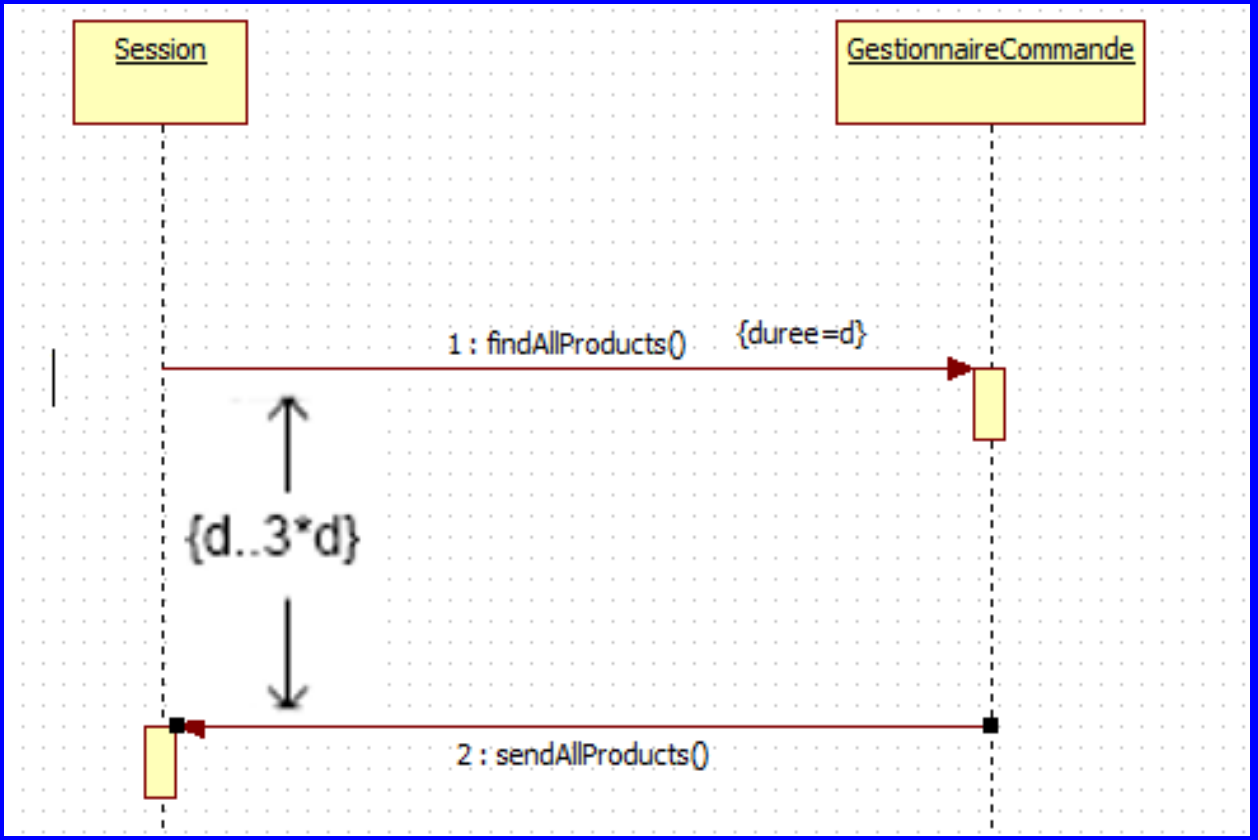
Diagrammes comportementaux

- **Diagramme de séquences 5/7**



Diagrammes comportementaux

- **Diagramme de séquences 6/7**



Diagrammes comportementaux

- **Diagramme de séquences 7/7**

Les diagrammes séquences sont très riches et permettent de modéliser beaucoup de chose

Pfff c'est compliqué

Oui mais on est pas obligé de tout utiliser, commençons par ce qu'on a compris

Diagrammes comportementaux

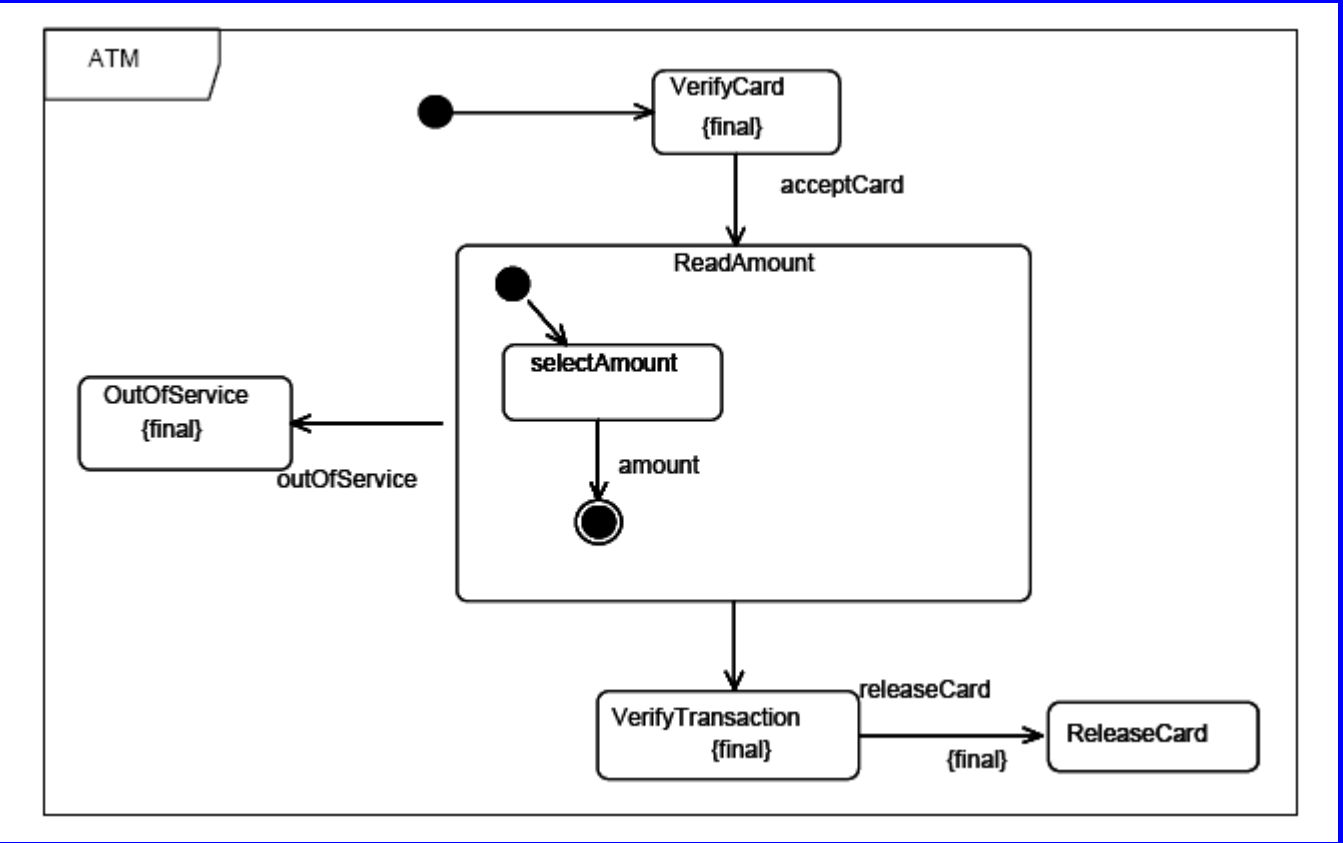
- Diagramme de machine d'états 1/4
  - Permet de décrire les objets modélisés par des états et recevant des messages
    - Messages qui peuvent être des événements ou des appels de méthodes
  - Bien adapté pour les IHM, les contrôleurs
  - Un état est défini par ses transitions possibles
    - Un état peut être vu comme un ensemble de valeurs des attributs d'une instance
  - Ne fait intervenir qu'une classe/système
    - Dans la majorité des cas

Diagrammes comportementaux

- **Diagramme de machine d'états 2/4**
  - **« Un classique » est d'avoir deux états : initialisé, non initialisé**
    - **Cela évite de mettre du code dans les constructeurs**
  - **Seul schéma de comportement « fermé »s**

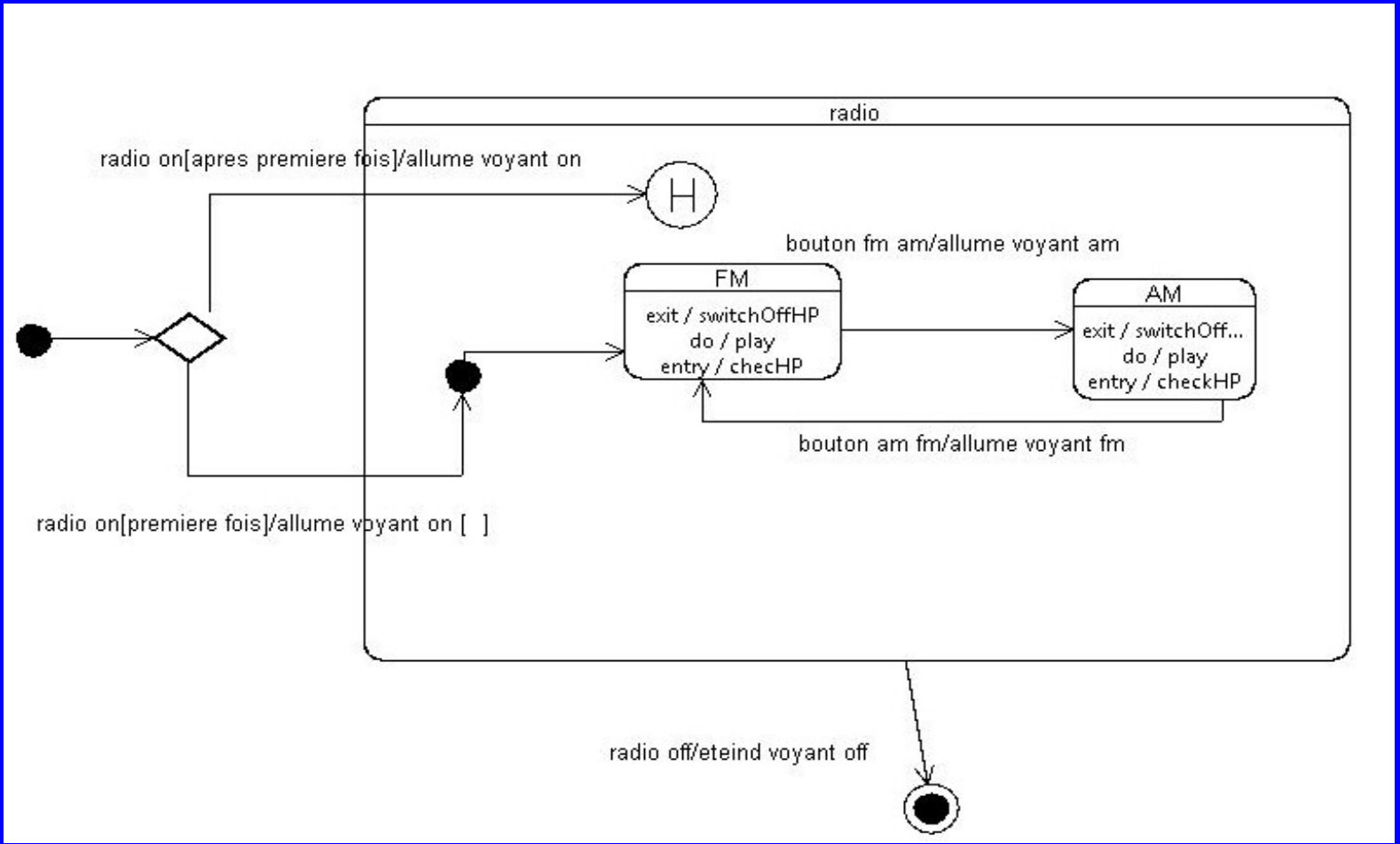
Diagrammes comportementaux

- **Diagramme de machine d'états 3/4**
  - **Exemple de machine d'états**



Diagrammes comportementaux

- **Diagramme de machine d'états 4/4**



Diagrammes comportementaux

- **Critères d'évaluation**
  - **Les types de diagrammes sont-ils bien choisis ?**
  - **Y a-t-il une cohérence de noms, de nomenclature, typographie entre l'ensemble des diagrammes**
  - **Les principaux CU ont-ils été mis en diagramme séquence**
  - **Les principales contraintes sont-elle identifiées ?**
  - **La traçabilité est-elle assurée ?**

Diagrammes comportementaux II

- QCM
  - Quel diagramme ne fait intervenir qu'une classe à la fois ?
  - Dans quel diagramme sont utilisés les cadres d'interaction ?
  - UML permet-il de décrire le code d'une méthode ?
  - Que représente un diagramme séquence ?

Diagramme de packages

Diagramme de packages

- Plan
  - Simple mais crucial
  - Règles
  - Diagramme
  - Relation entre packages
  - Critères d'évaluation
  - QCM

Diagramme de packages

- **Simple, mais crucial 1/2**
  - **Ce sont des diagrammes simples, mais très importants**
  - **Un package permet de ranger n'importe quel éléments UML**
    - **Y compris des packages qui deviennent alors des sous packages**
  - **Regrouper des éléments est un acte structurant fort**
    - **À ne pas négliger**

Diagramme de packages

- Simple, mais crucial 2/4
  - Les regroupements peuvent être techniques ou fonctionnels
  - Un package définit un espace de nommage
  - Concrètement ils modélisent les sources des libg, jar, dll et autres
    - Ce n'est pas tout à fait cela
    - Cette conception des package aide à la traçabilité
  - Ils doivent donner une bonne vue
    - De l'arborescence des sources
    - Du build du produit

Diagramme de packages

- **Simple, mais crucial 3/4**

Ils font le lien entre le build management et le monde des « sources »

Ils doivent respecter le pattern couche

Pattern, vous avez dit pattern ?

Diagramme de packages

- **Simple, mais crucial 4/4**
  - **Il ne doit pas y avoir d dépendances circulaires**
  - **C'est mieux s'ils ont vue interne et une vue externe**

On retrouve ce concept important

Interfaces ou encapsulation ?

Diagramme de packages

- **Règles 1/1**
  - **Un élément ne peut appartenir qu'à un et un seul package**
  - **Quand le package est détruit, ses éléments sont détruits également**

Diagramme de packages

- **Diagrammes 1/1**

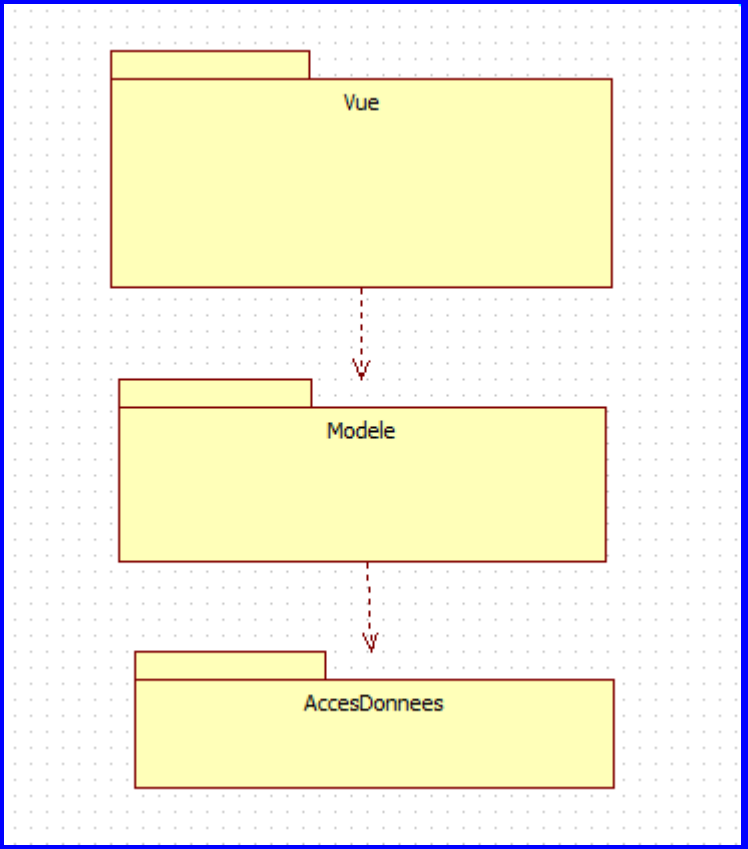


Diagramme de packages

- **Relation entre package 1/1**

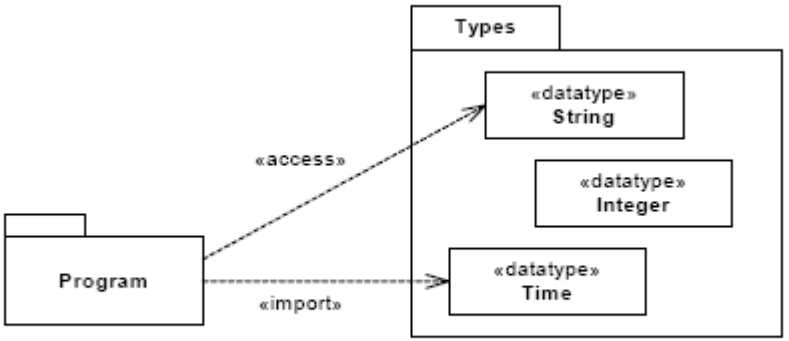


Figure 7.39 - Example of element import

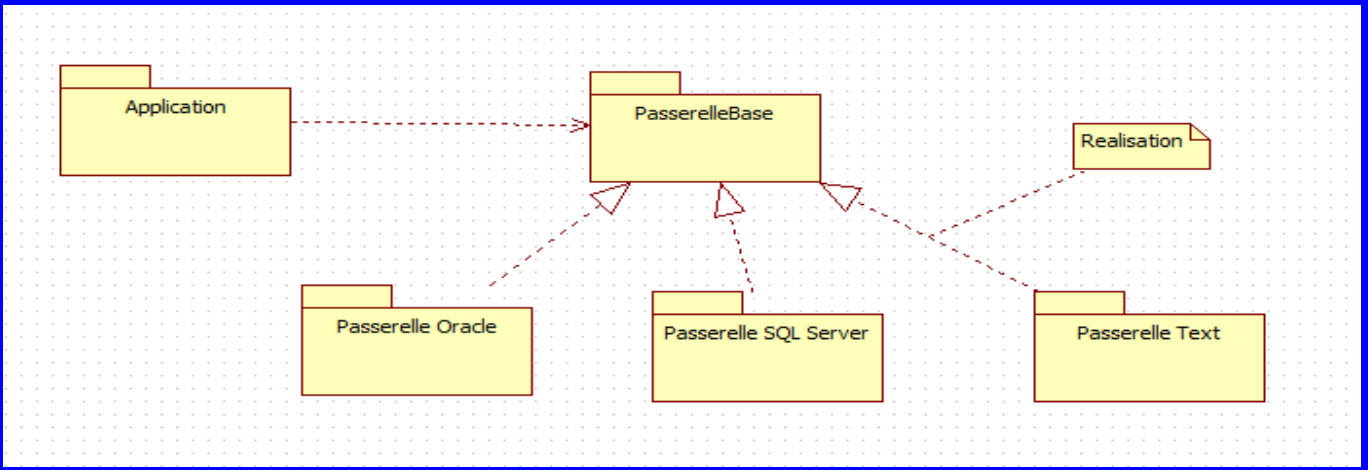


Diagramme de packages

- Critères d'évaluation
  - Le responsable du build a-t-il revu les diagrammes ?
  - Les dépendances sont-elles justifiées, minimales ? (pas de dépendance circulaire)
  - Les packages ont-ils une structure en couche ?
  - L'arborescence des sources est-elle visible ?
  - Y a-t-il une bonne traçabilité avec les diagrammes de classes ?
  - Les noms sont-ils signifiants ?

Diagramme de packages

- QCM
  - Que modélisent en première approche les packages ?
  - Les packages font le liens entre quels « mondes » ?
  - Quel pattern doit respecter les diagrammes de package ?
  - Comment est-il intéressant de modéliser les packages ?

Design de déploiement

Diagramme de déploiement

- Plan
  - Loin du développement
  - Définition
  - Diagrammes
  - Critères d'évaluation
  - QCM

Diagramme de déploiement

- Loin du développement 1/2
  - Ils définissent l'architecture matérielle sur laquelle est déployée l'application
  - Ils définissent les environnement
    - Au moins 3 :
      - Production
      - Intégration
      - Développement
  - Ils définissent les plates-formes d'exécution
    - Nombre, OS, Base de données, Serveur d'application
  - Ce n'est pas le monde du développement

Diagramme de déploiement

- **Loin du développement 2/2**
  - **Mais si on réalise un programme c'est pour qu'il soit utilisé et donc déployé**
  - **Quand on développe, il est impératif de tenir compte des contraintes opérationnelles**

Diagramme de déploiement

- **Définitions 1/1**
  - **Artéfact**
    - **Représente tout élément qui va être installé et utile au bon fonctionnement du système**
    - **Ce peut être :jar, dl, modèle de données, fichiers, scripts, exécutable, fichiers de configuration etc.**
  - **Nœud**
    - **Element sur lequel un artéfact est déployé : machine, serveur d'application, base de donnée etc.**
  - **Chemin de communication**
    - **Connexion entre les nœuds, définissent le réseau**

Diagramme de déploiement

- **Diagrammes 1/2**

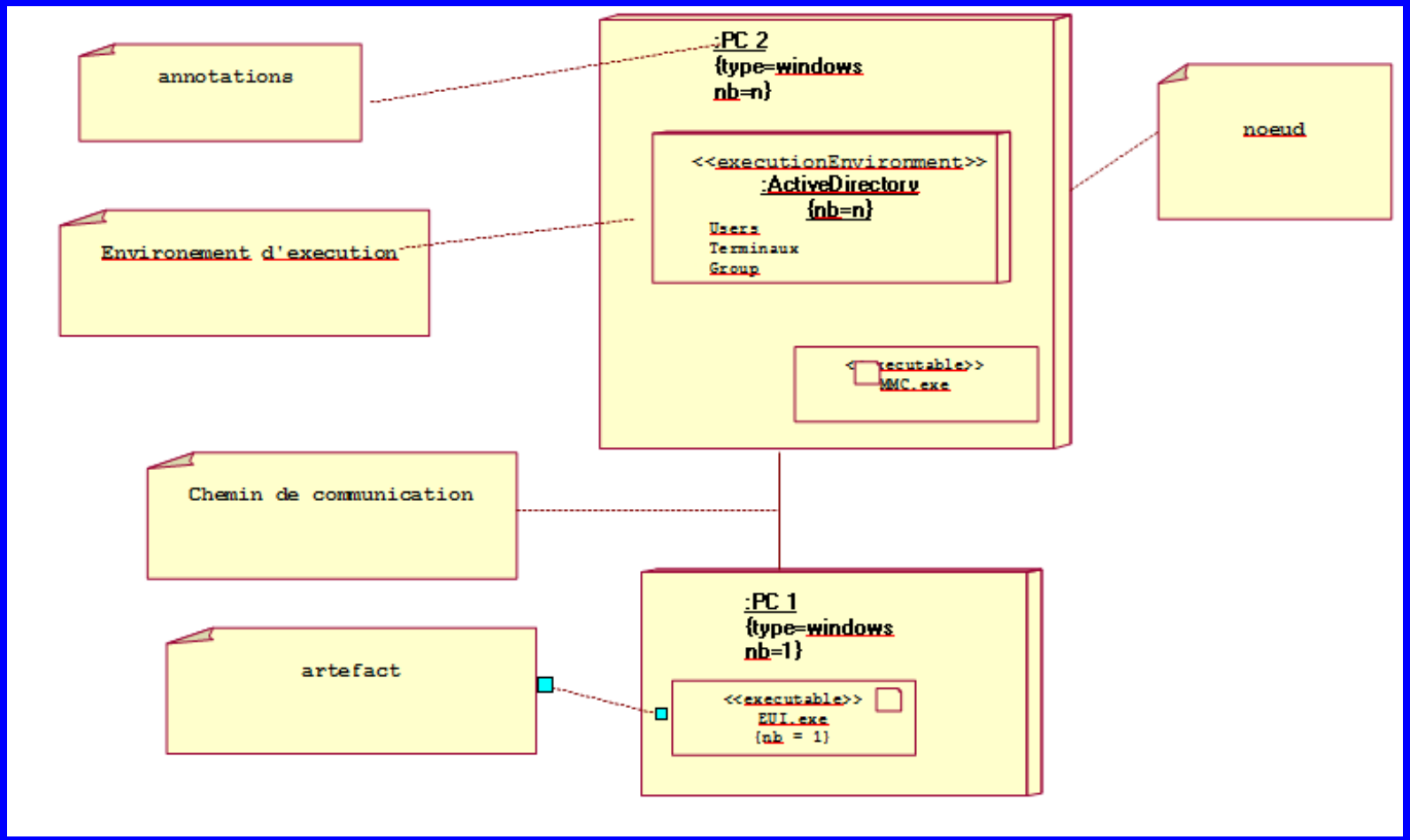


Diagramme de déploiement

- **Diagrammes 2/2**

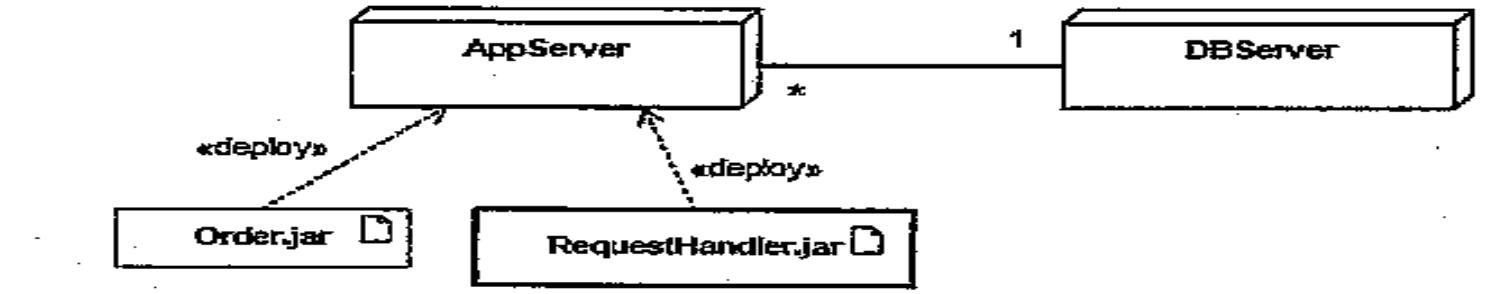


Figure 10.17 – Communication path between two Node types with deployed Artifacts.

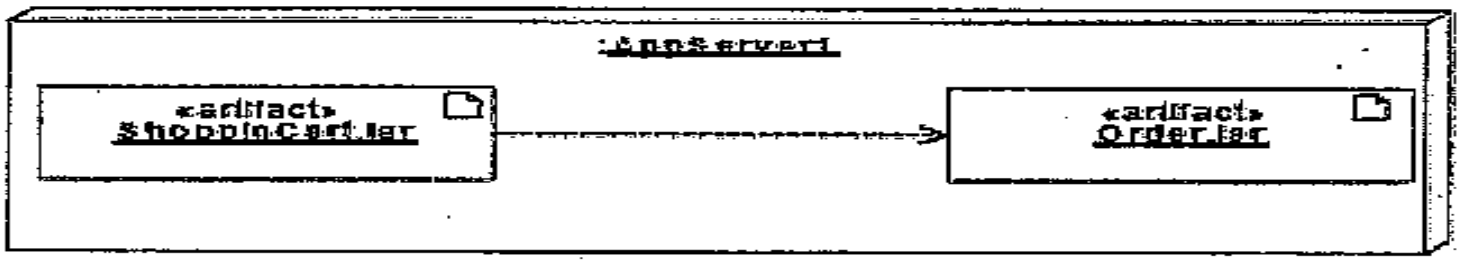


Figure 10.18 – A set of deployed component artifacts on a Node

Diagramme de déploiement

- Critères d'évaluation 1/1
  - Les responsables de l'installation ont-ils revu les diagrammes
  - Les administrateurs ont-ils été consultés ?
  - Les différentes configuration ont-elles été analysées ?
  - Les diagrammes ont-ils été fait à temps ?
  - Les diagrammes permettent-ils d'automatiser le déploiement ?

Diagramme de déploiement

- QCM
  - Qu'est-ce qu'un artéfact ?
  - Quelles personnes doivent revoir les diagrammes ?
  - À quel moment doivent-ils être réalisés ?
  - À quoi sert un chemin de communication ?

Diagramme de composant

Diagramme de composant

- Plan
  - Introduction
  - Un exemple : les plug ins
  - Objectifs
  - Composant simple
  - Composant fractionné
  - Des définitions
  - Complexité

Diagramme de composant

- **Introduction 1/3**
  - **Depuis les débuts de l'informatique, les informaticiens rêvent :**
    - **De ne pas développer perpétuellement la roue**
    - **D'utiliser des codes sans erreur, efficaces, complets**
    - **De construire un logiciel en utilisant des « composants » sur étagères**
      - **Comme un lego ou un puzzle**

Diagramme de composant

- **Introduction 2/3**

  - **Le rêve échoue toujours ...**
    - **Les open sources rendent le rêve de plus en plus réaliste**
    - **Ajax et le web aussi**

• Introduction 2/3a

- Jboss
- Hibernate
- Spring
- SLF4J
- JBPM
- Mule ESB
- Talend
- Jasper Soft
- Zabbix
- OCS , Puppet, Chef

- PostGres
- JFaces
- Eclipse RCP
- OpenLDAP
- JFX ...

Diagramme de composant

- **Introduction 3/3**
  - **Des « architectures » ont mis en avant le concept de composant :**
    - **Corba**
    - **COM/DCOM**
    - **EJB / JEE**
    - **Active X**
    - **Web Service**
  - **Un points clé : les Interfaces**
    - **Un composant se définit par ses interfaces**

Diagramme de composant

- Un exemple : les plugs in
  - Un plug in est un binaire
    - Dll, jar etc.
  - Un plug in implémente des interfaces reconnues par le logiciel hôte
  - Un plug in consomme les interfaces du logiciel hôte
  - Le plug In s'intègre dans le logiciel hôte
  - Le logiciel hôte devient la somme de ses composants

Diagramme de composant

- Objectifs 1/2
  - Un diagramme de composant découpe un système en sous parties autonomes et communiquant via des interfaces
    - « Divise un système compliqué en sous systèmes manipulables »
  - Les sous parties peuvent être logiques/fonctionnelles ou techniques
    - C'est un problème de ce diagramme

Diagramme de composant

- Objectifs 2/2
  - Les objectifs principaux sont
    - De communiquer et de présenter le système
    - De structurer le problème
      - En le découpant pour le rendre adressable
    - D'identifier les parties indépendantes
      - Les parties livrables indépendement les unes des autres
    - De mettre en place une architecture de plug in
      - Éventuellement ...

Diagramme de composant

- **Composant simple 1/2**

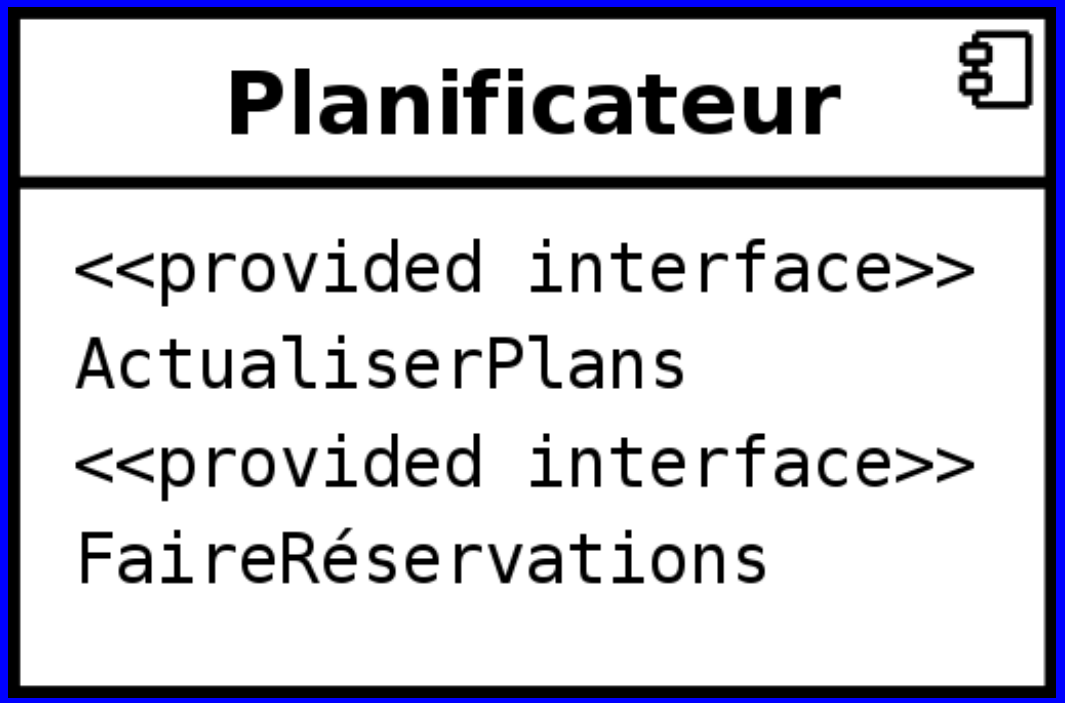
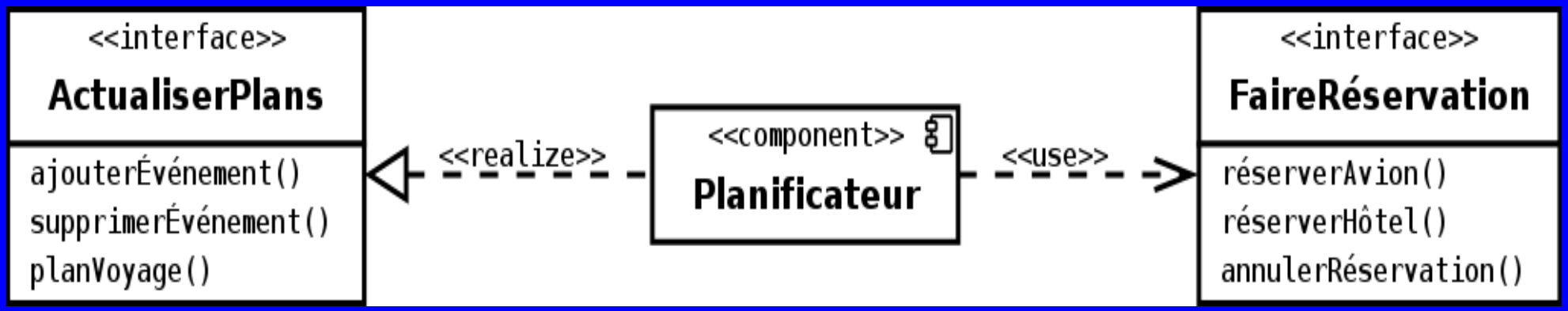


Diagramme de composant

- **Composant simple 2/2**



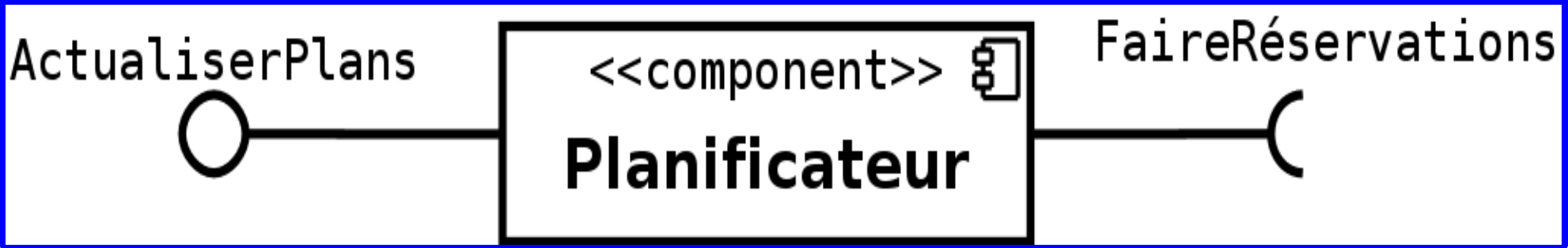


Diagramme de composant

- **Composant fractionné**

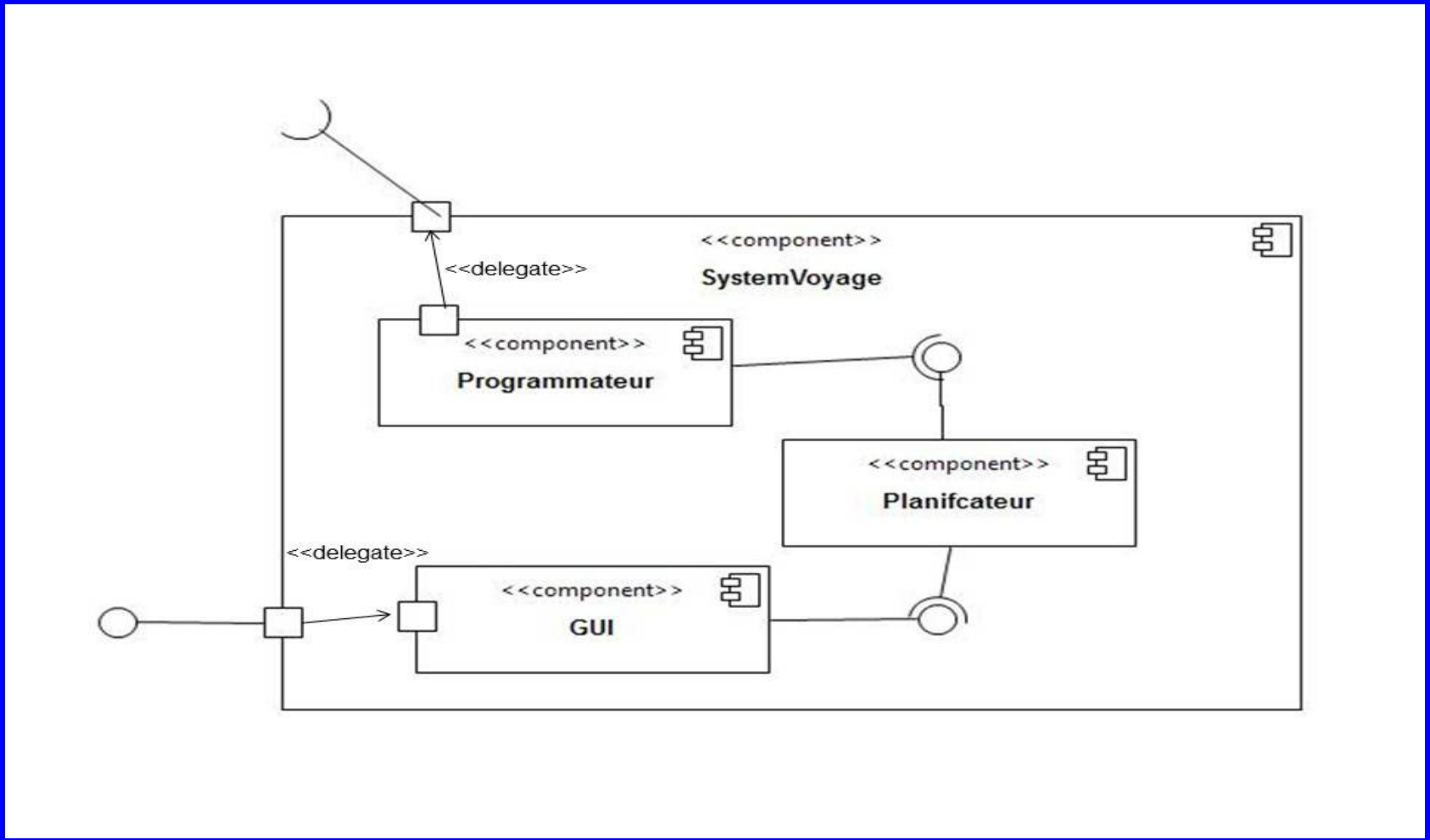


Diagramme de composant

- Des définitions 1/1
  - Les composants représentent la vue du logiciel que veut voir l'utilisateur en terme d'éléments interchangeables
  - Éléments offrant et consommant des interfaces
    - Éléments interchangeables bien sur
  - La définition UML 1.x ... qui avait le mérite d'être simple et compréhensible
    - Éléments « binaires » déployés
    - En UML 2 ce sont des artéfacts

Diagramme de composant

- Complexité 1/2
  - Un somme de composants est difficile à rendre cohérente
    - « Eclipse »
    - « L'enfer des Dlls »
  - Le bon niveau de granularité est difficile à trouver
    - Trop petit, ils fournissent peu de services et il y en a beaucoup
    - Trop grand ils sont un « élément monolithique »

Diagramme de composant

- **Complexité 2/2**
  - **Faire des éléments interchangeables, réutilisables est en soi un problème très compliqué ...**

Diagramme de composant

- **Critères d'évaluation 1/1**
  - **Les interfaces des composants sont-elles identifiées ?**
  - **Les composants respectent-ils la définition d'un composant ?**
  - **Les composants externes sont-ils identifiées ?**
  - **La granularité des composants est-elle correcte ?**
  - **La traçabilité avec les diagrammes de déploiement a-t-elle été réalisée ?**

Diagramme de composant

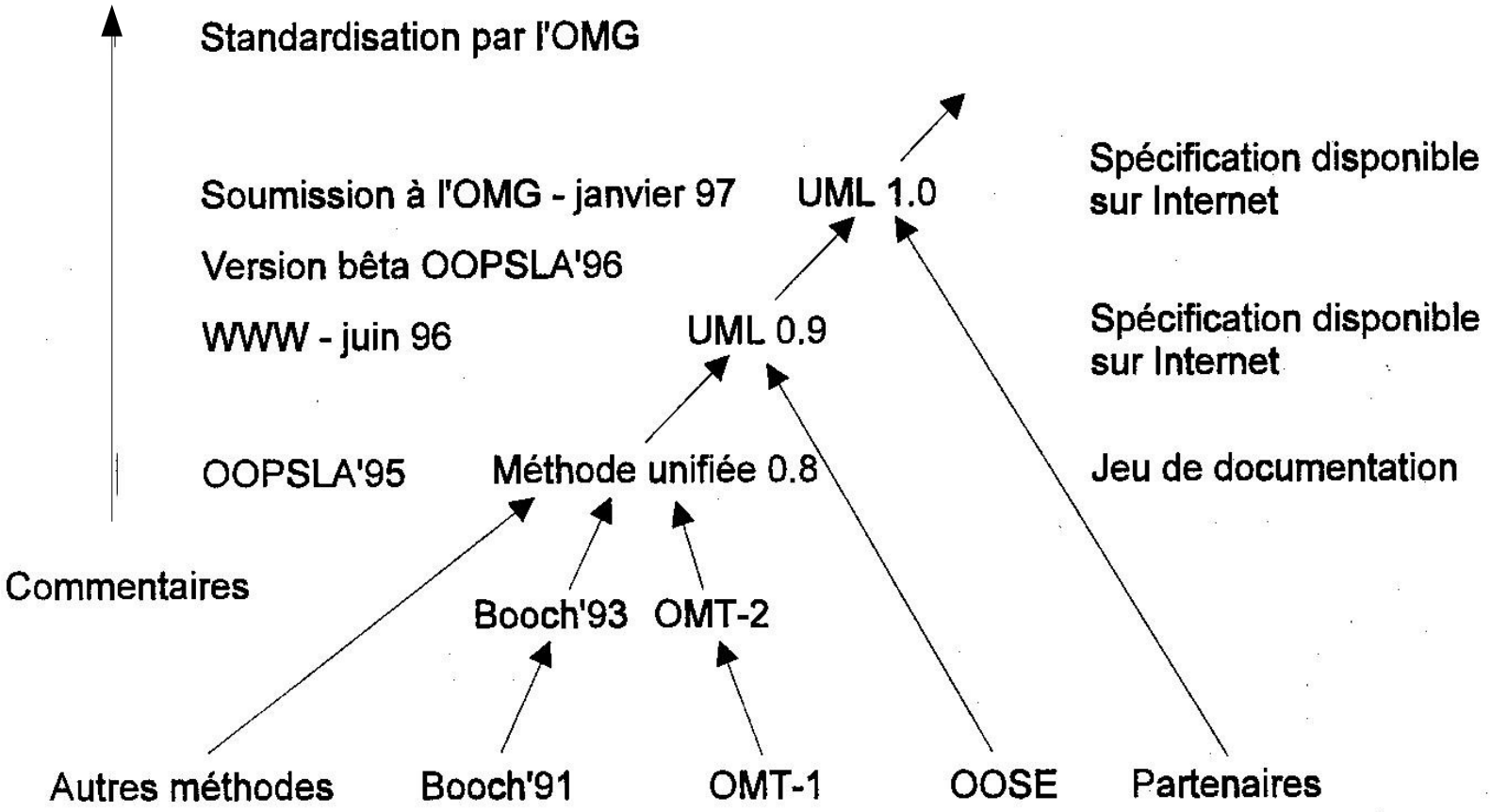
- QCM
  - Par quoi est définit un composant ?
  - Que représente un composant ?
  - Que faut-il faire avant de réaliser un composant ?
  - Quels sont les deux types de découpage qu'il est possible de faire ?

Conclusion

Conclusion

- **Historique 1/3**
  - **UML est promue par l'OMG**
    - **http://www.omg.org**
    - **Consortium d'entreprises**
  - **Norme ISO ISO/IEC 19501 pour UML 1.4.2**
  - **UML 1.0 : 1997**
  - **UML 1.5 : 2003**
  - **UML 2.0 : 2004-2005**
  - **MDA : ...**

Conclusion


Standardisation par l'OMG
Soumission à l'OMG - janvier 97
Version bêta OOPSLA'96
WWW - juin 96
OOPSLA'95
Commentaires
UML 1.0
UML 0.9
Méthode unifiée 0.8
Booch'93
OMT-2
Autres méthodes
Booch'91
OMT-1
OOSE
Partenaires
Spécification disponible sur Internet
Spécification disponible sur Internet
Jeu de documentation

Conclusion

- **Historique 3/3**
  - **UML unifie différentes méthodes : Booch, OMT, OOSE**
  - **Les acteurs :**
    - **Les trois amigos :**
      - **Grady Booch (Booch)**
      - **Ivar Jacobson (Object Oriented Software Engineering)**
      - **Jim Rumbaugh (Object Modeling Techique)**
    - **Les comités de l'OMG**

Conclusion

- Différents niveaux d'utilisation 1/2
  - UML est utilisable partiellement et de différentes façons
  - Le croquis au tableau pour partager une idée
    - Permet l'échange et la créativité
    - Les schémas sont volatils
    - Seule la communication compte
  - Documents de conception
    - Les schémas sont Intégrés dans une méthodologie
    - Ils sont conservés
    - Création d'un référentiel UML

Conclusion

- Différents niveaux d'utilisation 2/2
  - Ils doivent respecter un formalisme
  - Ils ne sont pas forcément de l'UML « pur »
    - Ils modélisent la pratique de la structure
  - Génération de code
    - Pas forcément du MDA
    - Quasiment de l'UML strict
      - Strict si MDA

Conclusion

- **Points importants 1/2**
  - **Traçabilité entre modèles**
  - **Vocabulaire et définitions des mots**
    - **Mots du langage usuel**
    - **Mots métiers**
    - **Mots techniques**
  - **Gestion des versions**
  - **Capture des raisons des choix**

Conclusion

- **Points importants 2/2**
  - **Maintenance des modèles et mise à jour ...**
  - **Réalisation d'un dictionnaire ...**
  - **UML est compliqué**
    - **Donc source d'erreurs s'il est mal utilisé**
    - **Il faut rester modeste**

Conclusion

Voilà c'est la dernière page

Si vous utilisez UML lors de vos projets, je suis intéressé d'avoir un retour sur le cours

Et surtout amusez vous en utilisant UML !