

Bertrand

LE GAL

Maître de conférences

ENSEIRB

bertrand.legal@enseirb.fr

<http://www.enseirb.fr/~legal/>

Laboratoire IMS

bertrand.legal@ims-bordeaux.fr

Université de Bordeaux 1

351, cours de la Libération

33405 Talence - France

Filière Electronique

2^{ème} année

2007 / 2008

“Introduction au langage UML”



1

“ Retour sur le flot
de Conception ”



Les causes courantes des échecs de projets

- *Mauvaise compréhension* des besoins des utilisateurs finaux.
- Incapacité à gérer les *modifications des exigences* des clients au cours du développement.
- Les modules composant le projet ne fonctionnent pas ensemble (*mauvaises interfaces*)
- *Code source* du logiciel *difficile à maintenir* ou à *faire évoluer*
- Logiciel de *mauvaise qualité* (beaucoup d'anomalies de conception / exécution)

L'idéal pour la gestion d'un projet !

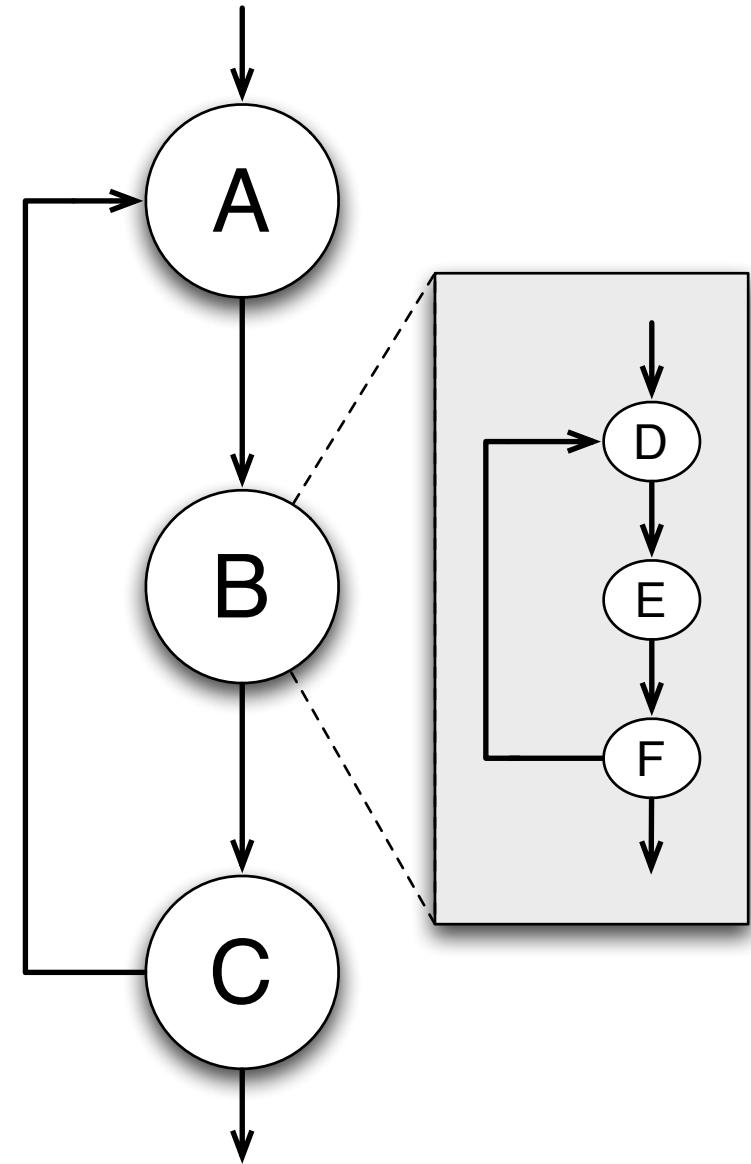
1. *Bien comprendre les besoins*, les demandes et les exigences des utilisateurs finaux.
2. *Bonne communication avec le client* pour valider certains choix et vérification de l'étape (1).
3. *Tester et valider chaque phase* de la conception pour ne pas découvrir des problèmes plus tard.
4. *Traiter au plus tôt les problèmes*.
5. *Maîtriser la complexité* et augmenter la réutilisation.
6. Définir une *architecture robuste*.
7. Faciliter le *travail en équipe*.

Les 7 bonnes pratiques

1. Développement *itératif*,
2. Développement à base de *composants* centré sur l'architecture,
3. Pilotage du *développement par les risques*,
4. Gestion des *exigences*,
5. Maîtrise des modifications,
6. *Evaluation continue de la qualité*,
7. Modélisation visuelles.

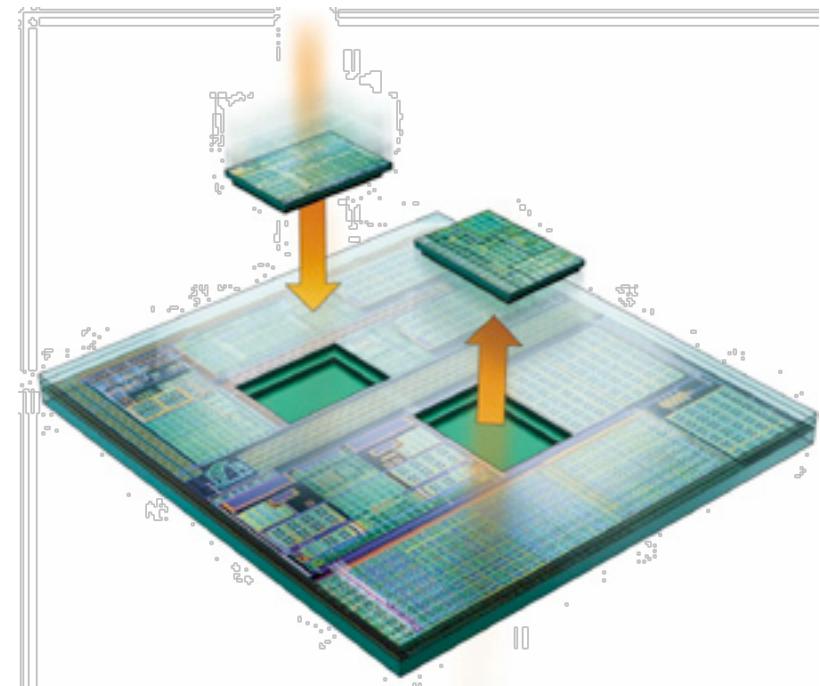
1 - Le développement itératif

- Cette méthode est basée sur de petites étapes (élémentaires)
- Chaque étape produit des retours et des adaptations si nécessaire
- Autres dénominations
 - ◆ Développement en spiral
 - ◆ Développement évolutif



2. Développement basé sur les composants

- Validation de l'architecture logicielle lors des premières itérations :
 - ◆ Développement basé sur des IP du commerce,
 - ◆ Réutilisation préférée au redéveloppement,
- Tester l'architecture au plus tôt même si elle n'est pas fonctionnellement complète.



Méthodologie valable dans le cadre du développement de systèmes logiciels et matériels

3. Pilotage par les risques

- Analyser les risques potentiels au plus tôt (début de la conception) :
 - ◆ Les regrouper et les hiérarchiser,
 - ◆ Travailler en priorité sur les risques critiques,
- Cela concerne les risques techniques, ainsi que :
 - ◆ Les risques liés aux clients,
 - ◆ Les risques liés au domaine applicatif,
 - ◆ Les risques liés à l'organisation du projet,
- Sémantique
 - ◆ *Un risque est un événement redouté dont l'occurrence est plus ou moins prévisible et qui aura des répercussions sur le projet.*
 - ◆ *Un problème est un risque qui s'est révélé.*

4. Les exigences du client

■ Sémantique

- ◆ *Une exigence est une condition à laquelle le système doit satisfaire ou une capacité dont il doit faire preuve.*

■ Les exigences peuvent être :

◆ Fonctionnelles

- Elles décrivent ce que le système doit savoir faire,

◆ Non fonctionnelles

- Qualité des services,
- Temps de réponse, temps de traitement,
- Sécurité au fonctionnement,
- IHM adaptée aux utilisateurs.

5. Les demandes de changement

■ Sémantique

- ◆ *Une demande de changement est une requête visant à modifier un artefact ou un processus.*
- Il faut dans ce cas indiquer dans la documentation, la nature du changement, *sa cause* et *ses impacts* sur la solution proposée.
- Il existe 2 types de changements :
 1. La demande d'*évolution*
 - Nouvelle caractéristique du système ou modifications d'une existante
 2. Le rapport d'*anomalie*

6. Evaluation constante de la qualité

- Il a été prouvé que l'identification des erreurs de conception au plus tôt permet de “gagner” de l'argent (ou d'en perdre moins)
 - ◆ Ratio 10 à 100 !
- Les problèmes de qualité sont souvent les causes principales des dépassements de délais,
- Les phases de test, de vérification et de correction consomment environ 70% du temps total d'un projet !
- 60% des problèmes existent déjà lors de la conception de l'application.

Pourquoi utiliser la modélisation visuelle ?

- On doit pouvoir mémoriser nos pensées et les communiquer en utilisant des langages visuels et schématiques.
 - ◆ Réseaux de Pétri, Grafcet,
 - ◆ Schémas blocs (décomposition fonctionnelle),
 - ◆ UML, etc.
- Avantages de ces approches
 - ◆ Les *langages visuels* sont naturels et *faciles à comprendre* (non relatifs à un langage de communication),
 - ◆ On estime que 50% du cerveau est impliqué dans le processus visuel.

2

“Introduction à
UML”



Introduction à UML

- UML est un langage standard conçu pour permettre aux concepteurs d'élaborer les plans des logiciels qu'ils doivent développer.
- On se sert du langage UML pour:
 - ◆ *Visualiser* => *Schématiser le travail à réaliser,*
 - ◆ *Spécifier* => *Exprimer formellement les contraintes,*
 - ◆ *Construire* => *Commencer à développer une solution,*
 - ◆ *Documenter* => *Expliquer les choix réalisés (voir les problèmes),*
 - ◆ *Communiquer* (travailler en équipe),

UML est un langage

- Ce langage possède un *vocabulaire et des règles* qui permettent de *combiner les mots* afin de communiquer.
- La modélisation permet de comprendre le système à concevoir de manière formelle,
 - ◆ Tous les modèles sont liés les uns avec les autres afin de représenter le système et les interactions entre les blocs élémentaires.
- Le vocabulaire et les règles d'écriture régissent la manière de construire et de lire les modèles correctement mis en forme.
 - ◆ *Aucune décision d'implémentation n'est prise lors de la phase de spécification et d'analyse du cahier des charges.*

UML, un langage pour visualiser

- Pour certains développeurs, il y existe un lien direct entre une idée et son code d'implémentation.
 - ◆ Utilisation d'un *modèle de représentation mental*,
- Problème dans le partage de ces modèles avec d'autres personnes,
 - ◆ *Risques d'incompréhension*, d'incertitudes...
 - ◆ Chaque organisation et/ou personne possède son propre langage (difficulté d'intégration),
 - ◆ Risque d'oubli des décisions prises avec le temps !
- *UML permet de résoudre les problématiques du partage et de mémorisation de l'information de manière visuelle.*

UML, un langage pour visualiser

- UML est bien plus qu'un assemblage de symboles graphiques !
 - ◆ Chaque symbole graphique possède sa sémantique propre (signification universelle),
 - ◆ Un modèle peut être écrit par un concepteur et compris pas un autre sans ambiguïté à la lecture,
 - ◆ Une automatisation par outil peut ainsi être mise en place pour traiter certaines informations => Génération de code source,
- UML est un *méta-langage de modélisation* permettant d'unifier les modèles utilisés dans les méthodes de développement (raffinement).

UML, un langage pour spécifier

- Dans le cadre d'UML, *spécifier signifie construire sans ambiguïté*,
- UML offre la possibilité de spécifier le comportement de l'application de manière formelle,
 - ◆ Réflexion sur des *modèles formels*,
 - ◆ *Approche indépendante du langage* objet d'implémentation (C++, Java, etc.),
 - ◆ *Raffinement progressif des modèles* (on construit une solution de manière itérative en plusieurs passes successives),
- UML n'est pas une méthode mais une notation qui laisse la liberté de conception (choix dans les solutions à mettre en oeuvre).

UML, un langage pour documenter

- Lors du développement d'un produit, on réalise des choix lors de l'élaboration de la solution :
 - ◆ Architecture d'implémentation,
 - ◆ Besoins identifiés,
 - ◆ Code source du projet,
 - ◆ Planning de développement,
 - ◆ Les plans et les jeux de test, etc.
- UML permet de réaliser la définition et la documentation de ces tâches en fonction des contraintes.

Les domaines d'utilisation de UML

- UML est employé dans l'ensemble des secteurs du développement informatique et électronique :
 - ◆ Systèmes d'information,
 - ◆ Télécommunication, défense,
 - ◆ Transport, aéronautique, aérospatial,
 - ◆ Domaines scientifiques,
- Mais pas seulement : on peut modéliser des comportements mécaniques, humain, etc.
 - ◆ Structure d'un système de santé / judiciaire, etc.

Les briques de base d'UML

■ Dans le langage UML, il existe 3 types de briques de base pour modéliser tous systèmes :

1. Les *éléments*

✓ Ce sont les abstractions essentielles au modèle.

2. Les *relations*

✓ Les relations expriment les liens existants entre les différents éléments.

3. Les *diagrammes*

✓ Les diagrammes comprennent des ensembles d'éléments dignes d'intérêt.

Il existe 4 types d'éléments dans UML

- Ce sont les éléments de base qui vont permettre la définition par la suite des modèles,
- Nous allons avoir différents types de modèles pour représenter les différentes caractéristiques du système,
 1. Les éléments *structurels*,
 2. Les éléments *comportementaux*,
 3. Les éléments de *regroupement*,
 4. Les éléments *d'annotation*.

Les éléments structurels

- Les éléments structurels sont les éléments les plus statiques d'UML, il en existe 7 types distincts :
 1. Les classes,
 2. Les interfaces,
 3. Les collaborations,
 4. Les cas d'utilisation,
 5. Les classes actives,
 6. Les composants,
 7. Les noeuds,
- A partir de là, on peut exprimer visuellement les relations structurelles de tous types de systèmes .

Les éléments comportementaux

- Les *éléments comportementaux* représentent les *parties dynamiques* des modèles,
- Ils permettent de définir le comportement du modèle dans le temps et l'espace,
- Il en existe 2 types fondamentaux de diagrammes pour exprimer les relations comportementales :
 1. Les interactions,
 2. Les automates à états finis.

Les éléments de regroupement

- Les *éléments de regroupement* représentent les *parties organisationnelles* des modèles UML,
- Ce sont les boites dans lesquelles un modèle peut être décomposé,
- Il existe un seul type d'élément de regroupement : “le package”

Les éléments d'annotation

- Les *modèles d'annotation* représentent les *parties explicatives* des modèles UML.
- Ce sont les commentaires qui vont être ajoutés aux modèles afin :
 - ◆ D'expliquer un choix de conception,
 - ◆ Décrire le fonctionnement du système ou d'une sous-partie,
 - ◆ Faire une remarque sur un élément du système,
 - ◆ Détailler une hypothèse d'utilisation,
 - ◆ Spécifier une contrainte d'implémentation (temps d'exécution maximum, consommation en énergie, etc.).

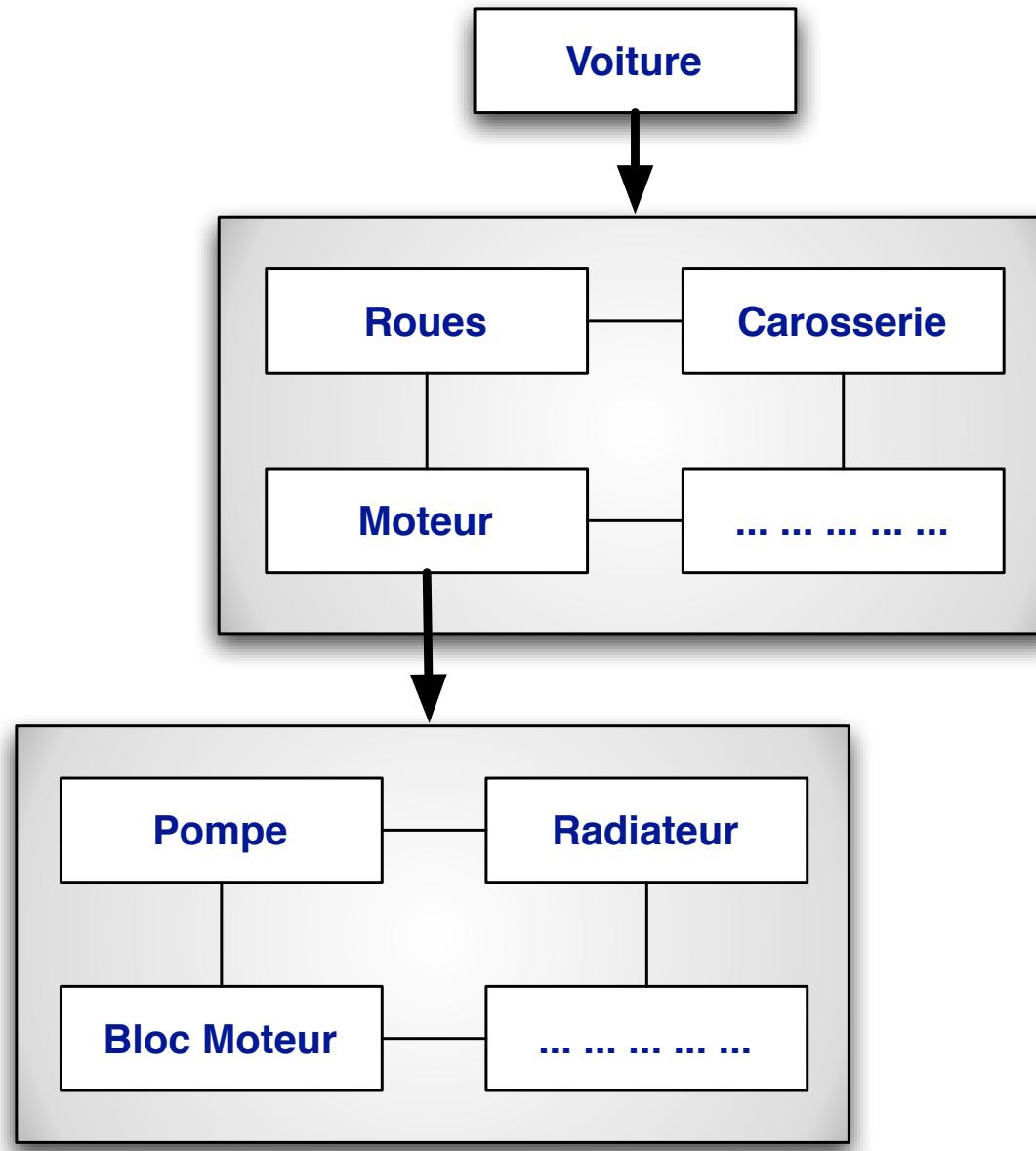
Les relations dans UML

- Afin d'exprimer les liens interconnectant les différents éléments des modèles, *4 relations de base* ont été définies :
 1. La *dépendance*,
 2. La *généralisation*,
 3. L'*association*,
 4. La *réalisation*,

Les diagrammes en UML

- Un *système* est un ensemble de sous-systèmes organisés pour atteindre un objectif et décrit à l'aide de modèles.
- Un *sous-système* est un regroupement d'éléments , dont certains spécifient le comportement proposé par les autres éléments contenus.
- Un *modèle* est une abstraction complète et cohérente d'un système permettant d'en faciliter la compréhension.
 - Modélisation des parties structurelles (statique)
 - Modélisation du comportement (dynamique)

Exemple de décomposition hiérarchique



Les diagrammes avec UML

- Un *diagramme* est une *représentation visuelle* de l'ensemble des éléments qui constituent le système.
 - ◆ Ils servent à visualiser un système sous différents angles (utilisateur, administrateur par ex.)
- Dans les systèmes complexes, *un diagramme* ne fourni qu'une *vue partielle* du système.
 - ◆ L'ensemble des diagrammes aboutés permet d'obtenir une vue globale du système à concevoir.
 - ◆ Chaque diagramme va permettre de modéliser ou spécifier une vue (spécificité) du système à concevoir.

Les diagrammes dans UML : 9 types

■ Les diagrammes “*statisques*”

1. Diagramme de classes,
2. Diagramme d'objets,
3. Diagramme de composants,
4. Diagramme de déploiement,

■ Les diagrammes “*dynamiques*”

5. Diagramme de cas d'utilisation,
6. Diagramme de séquences,
7. Diagramme de collaboration,
8. Diagramme d'états-transitions,
9. Diagramme d'activités.

Les diagrammes structurels

■ Les *diagrammes de classes*

- ◆ Ils représentent l'ensemble des classes, des interfaces et des collaborations ainsi que leurs relations.
- ◆ Les diagrammes les plus utilisés dans une approche de développement objets.

■ Les *diagrammes d'objets*

- ◆ Ils représentent un ensemble d'objets (instances de classes) avec leurs relations,
- ◆ Similaire aux diagrammes de classes, mais avec des détails réels et des prototypes.

Les diagrammes structurels

■ Les *diagrammes de composants*

- ◆ Ils sont apparentés aux diagrammes de classes.
- ◆ Un composant correspond généralement à une ou plusieurs classe, interfaces ou collaboration.

■ Les *diagrammes de déploiements*

- ◆ Ils sont apparentés aux diagrammes de composants.
- ◆ Un noeud englobe généralement un ou plusieurs composants.

Les diagrammes comportementaux

■ Les *diagrammes de cas d'utilisation*

- ◆ Organisent les comportements du système.
- ◆ Représentation des acteurs et de leurs relations avec l'application.

■ Les *diagrammes de séquences*

- ◆ Centrés sur l'ordre chronologique des messages.
- ◆ Il modélisent chronologiquement les messages échangés par les objets du système (sous-système).

✓ **Ce sont les 2 modèles les plus employés (modélisation dynamique)**

Les diagrammes comportementaux

■ Les *diagrammes de collaboration*

- ◆ Centrés sur l'organisation structurelle des objets qui communiquent (envoie & réception des messages),
- ◆ Mise en évidence des liens existants entre les objets,

■ Les *diagrammes états-transitions*

- ◆ Centré sur le changement d'état du système en fonction des événements,
- ◆ Modélisation sous forme d'un automate à états-finis (état, action, transition),

■ Les *diagrammes d'activités*

- ◆ Centrés sur le flot de contrôle d'une activité sur une autre.

3

“ Diagramme de
cas d'utilisation ”



Introduction aux cas d'utilisation

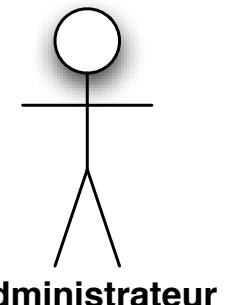
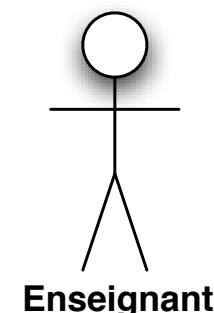
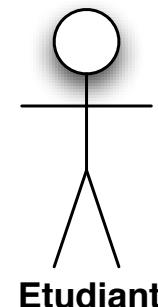
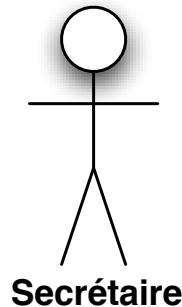
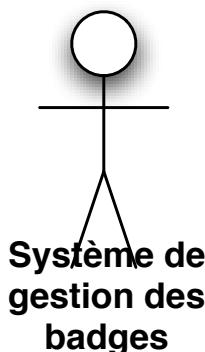
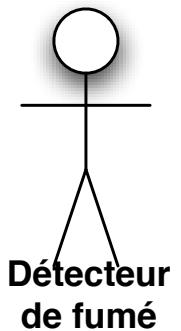
- Les cas d'utilisation servent à modéliser le *comportement dynamique* d'un système (sous système).
- Les comportements différents d'un système sont représenté sous forme de cas d'utilisation,
 - ◆ On ne se préoccupe pas de l'implémentation du système, juste des services qu'il doit rendre aux utilisateurs,
 - ◆ Données directement extraites du cahier des charges,
- Un cas d'utilisation décrit un *ensemble de séquences* dans lequel chaque séquence représente les interactions entre l'extérieur du système (acteurs) et le système lui même.

Introduction

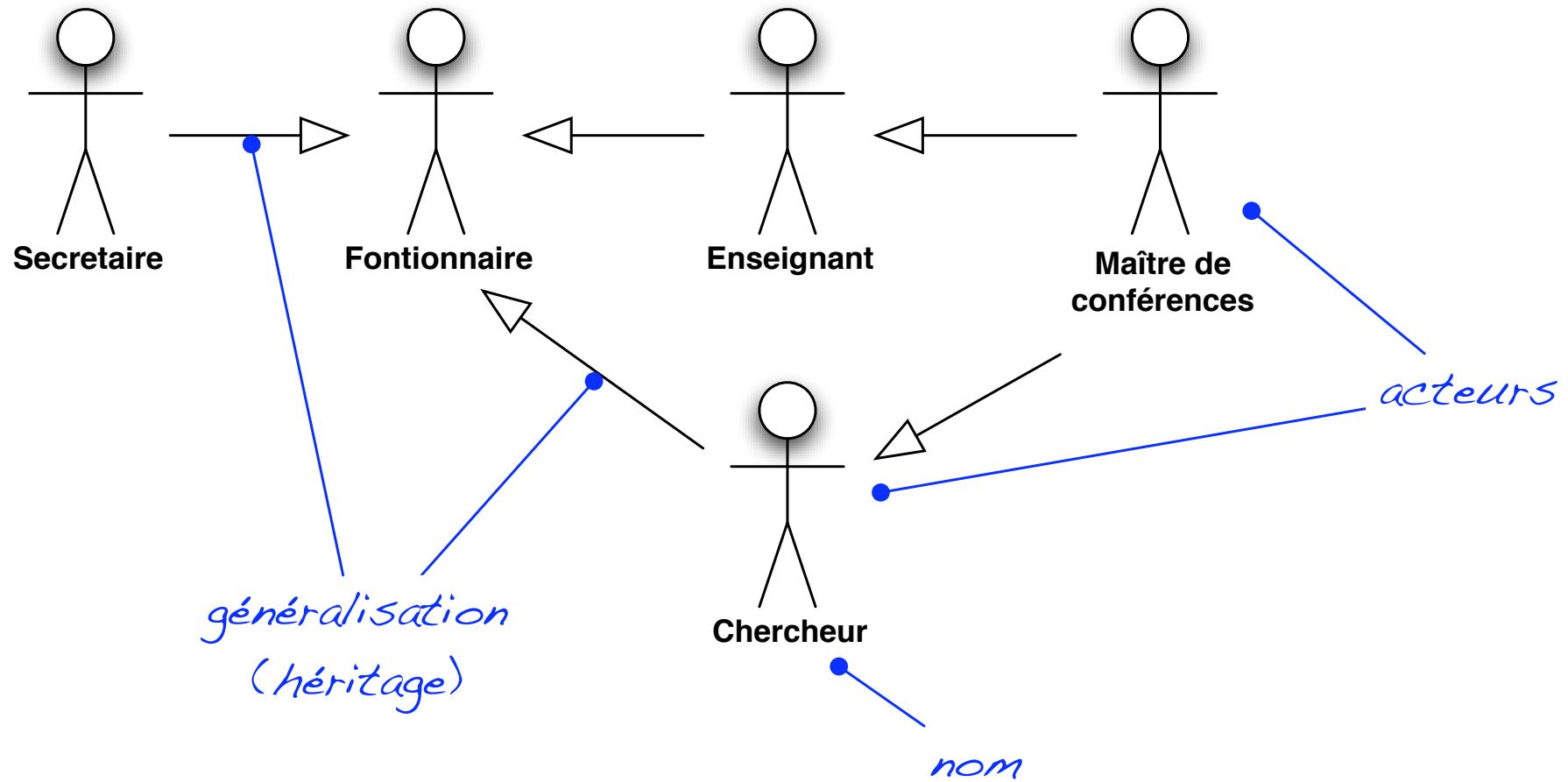
- L'expression des cas d'utilisation permet de déterminer les points suivants :
 - ◆ Quels sont les services que doit implémenter le système ?
 - ◆ A qui ces services doivent être rendus ?
- Les cas d'utilisation se composent :
 - ◆ *D'acteurs externes* au système (personnes, capteurs, autres applications ...),
 - ◆ *Du système* en lui même et des services qu'il doit rendre,
 - ◆ *Des relations* qui relient les acteurs aux services qui leurs sont rendus,

Cas d'utilisation, les acteurs

“ Un acteur est une personne ou une chose qui va rentrer en interaction avec le système à concevoir ! “

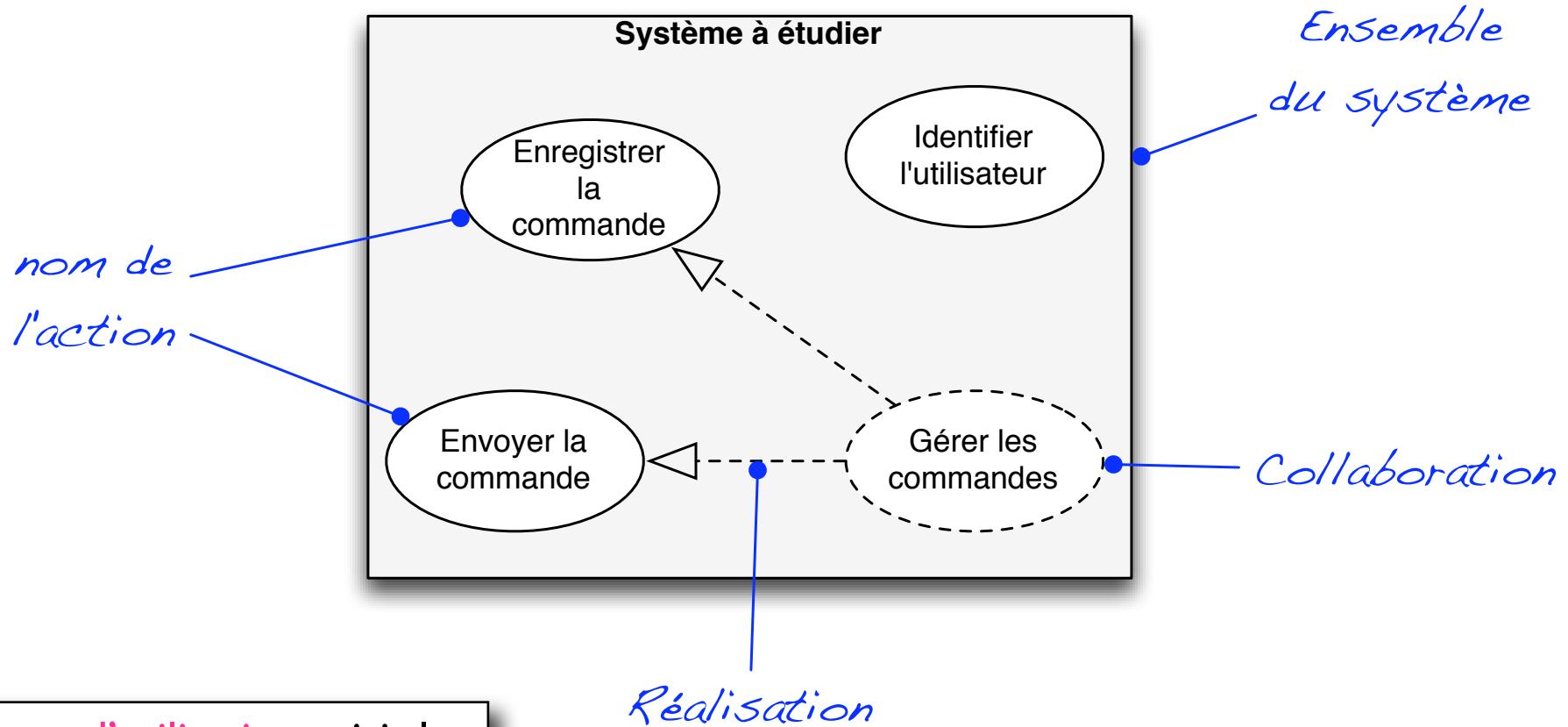


Cas d'utilisation, relations entre acteurs



Une **secrétaire** est une **fonctionnaire**, tout comme les **enseignants** et les **chercheurs**. Un **maître de conférence** est un **fonctionnaire** qui fait un travail d'**enseignant** et de **chercheur** à la fois.

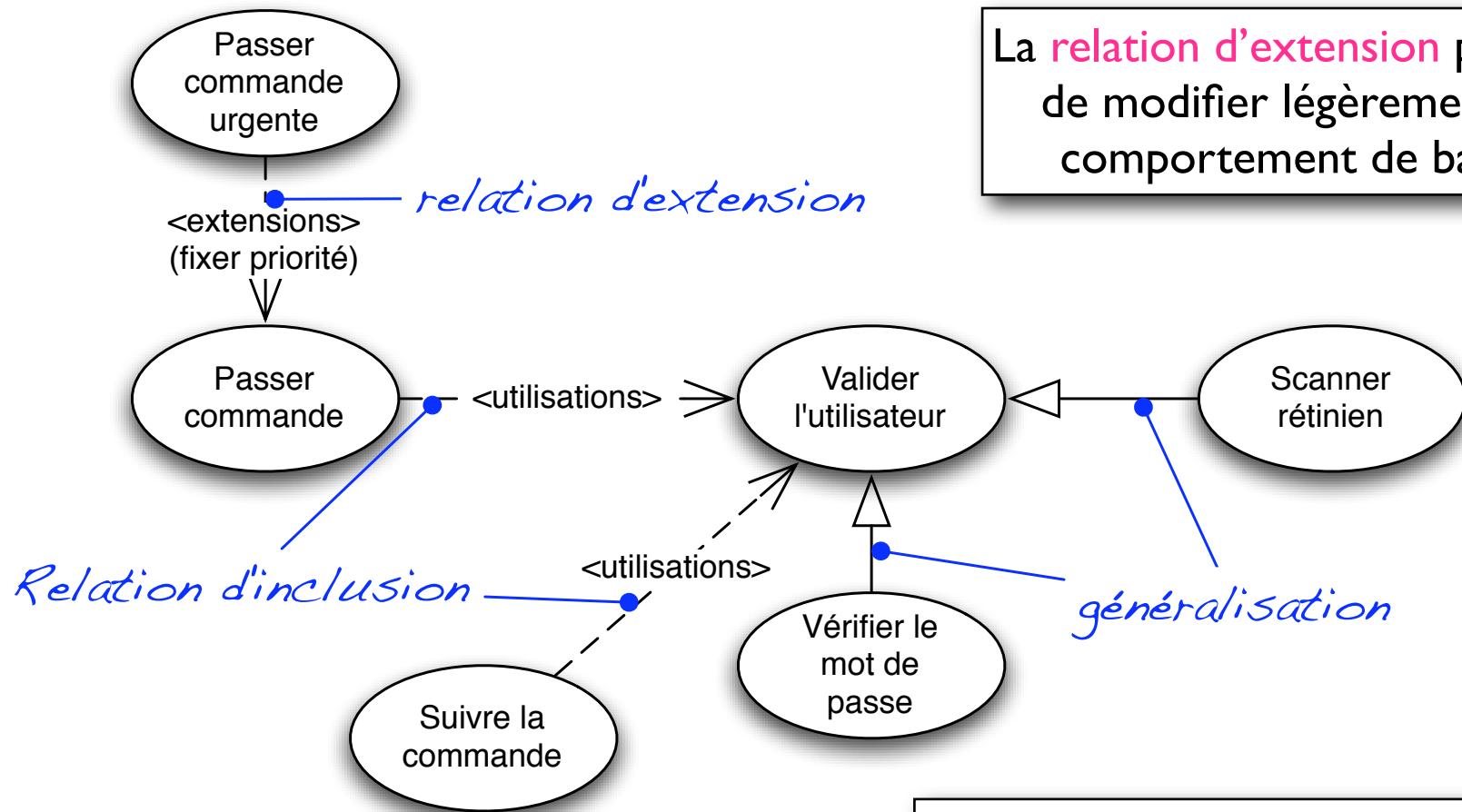
Cas d'utilisation, les actions



Un **cas d'utilisation** saisit le comportement attendu du système sans que l'on précise comment cela est réalisé.

La **collaboration** permet de raffiner la vue implémentation du système en factorisant les points communs.

Cas d'utilisation, Relations entre actions



La **relation d'inclusion** est faite pour éviter la répétition des actions en factorisant les services rendus. Cela revient à déléguer une partie des actions.

La **relation d'extension** permet de modifier légèrement le comportement de base.

La **relation de généralisation** modélise un héritage du comportement de base afin de l'affiner.

Afin de dessiner un cas d'utilisation...

- Identifier l'ensemble des acteurs possibles qui vont interagir avec l'application,
 - ◆ Identifier tous les acteurs,
 - ◆ Les factoriser sous forme de groupes,
- Prendre en considération les besoins de tous les utilisateurs
 - ◆ Identifier tous les services que doit rendre le système,
 - ◆ Le regrouper les besoins communs à l'aide des relations d'inclusion, de généralisation et d'extension,

Exemple pédagogique : l'ENSEIRB

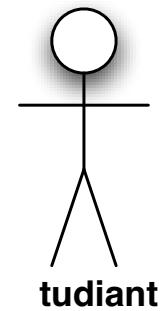
- L'ENSEIRB désire automatiser le processus d'inscription des étudiants, en voici le cahier des charges :
 - ◆ Le *responsable des filières* établit au début de chaque semestre le *programme des cours*
 - ◆ *Chaque étudiant* doit sélectionner *4 cours obligatoires* et *2 cours optionnels* par semestre.
 - ◆ Lorsqu'un étudiant s'inscrit, le *système de facturation* est averti et génère immédiatement la facture.
 - ◆ *Les enseignants* utilisent ces informations pour *consulter leur service*.
 - ◆ *Les étudiants* peuvent consulter *leur emploi* du temps en fonction des cours qu'ils ont choisis.

Exemple, Identification des acteurs

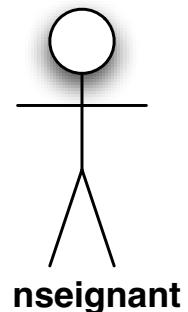
- 4 acteurs existent dans notre cahier des charges :

1. L'étudiant,
2. L'enseignant,
3. Le système de facturation,
4. Le directeur des filières,

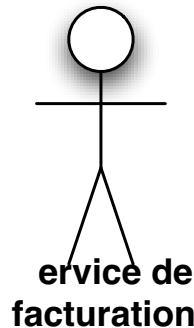
- Ce sont les seules personnes qui interagissent avec le système (selon le cahier des charges).



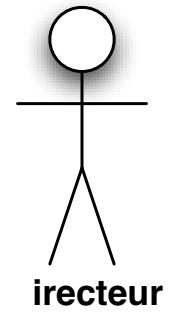
étudiant



enseignant



service de facturation

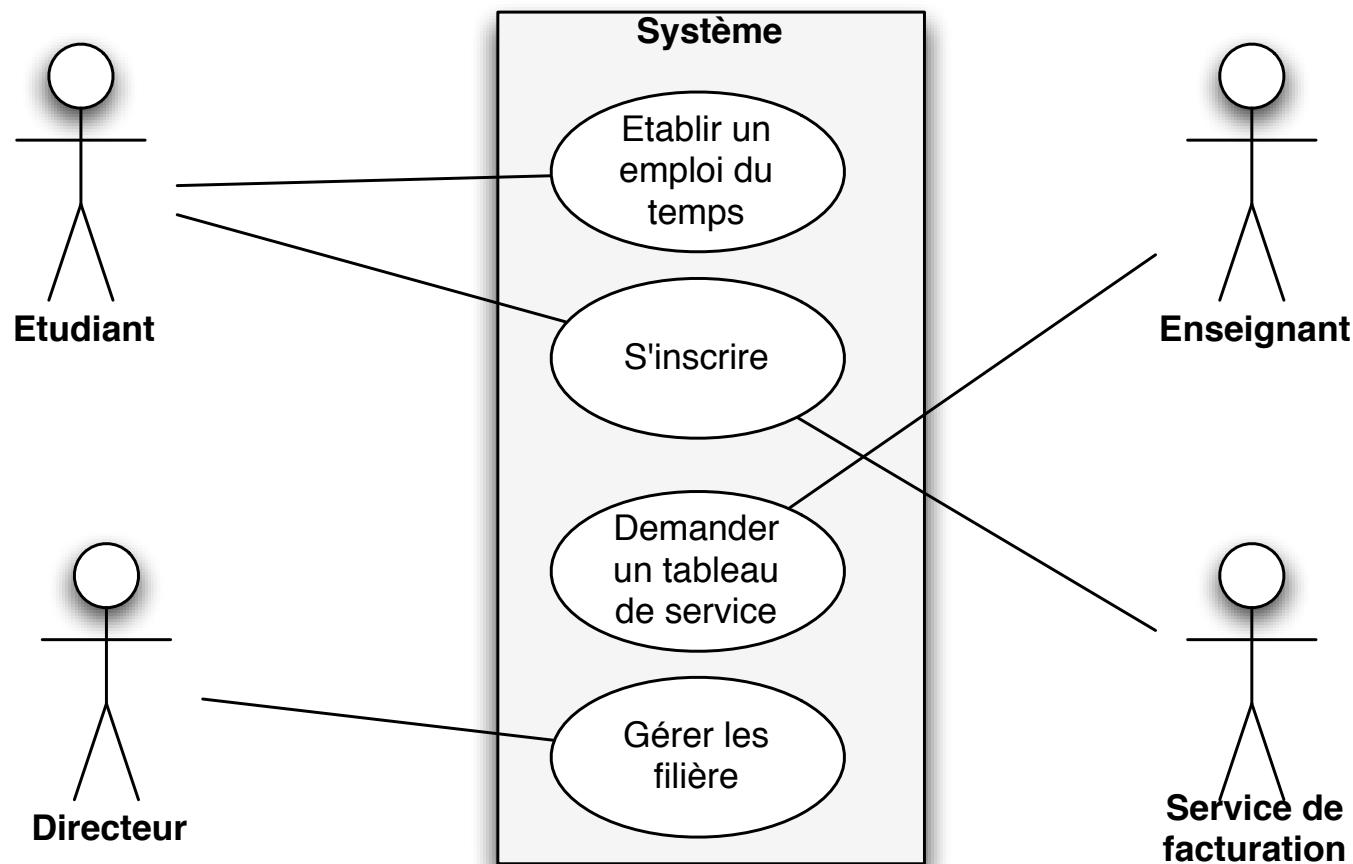


directeur

Exemple, Identification des fonctions

- Un cas d'utilisation est un motif de comportement intrinsèque au système,
 - ◆ Chaque cas d'utilisation est une séquence de transactions connectées, effectuées par un dialogue entre un acteur et le système.
- Identification des besoins propres aux acteurs
 - ◆ *Chef de service* - maintenir le programme des études.
 - ◆ *Enseignant* - demander un tableau de service.
 - ◆ *Etudiant* - établir un emploi du temps.
 - ◆ *Système de facturation* - recevoir les informations de facturation du système d'inscription.

Les cas d'utilisations, solution



Bien identifier les cas d'utilisation

■ Définition d'un cas d'utilisation

- ◆ Ensemble d'actions produisant un résultat observable pour un acteur particulier,

■ Vérifier que chaque cas d'utilisation candidat,

- ◆ Fournit une valeur ajoutée à la description,
- ◆ Qu'il existe un événement externe le produisant,
- ◆ On peut le décrire succinctement pour l'expliquer,

■ Attention au niveau de granularité du modèle !

- ◆ Ne pas descendre trop bas,
- ◆ Limiter le nombre de niveau (<20)

Documenter les cas d'utilisation

■ Pour chaque cas d'utilisation, il faut :

- ◆ Exprimer ce que fourni l'utilisateur au système,
- ◆ Ce que l'utilisateur reçoit en retour après exécution,
- ◆ Il faut aussi penser à détailler les hypothèses réalisées lors de la description du cas d'utilisation (pré-requis et les post-conditions)

■ L'expression des transactions entre le système et l'utilisateur sera détaillé par d'autres modèles plus adaptés :

- ◆ Diagrammes de séquences,
- ◆ Diagrammes d'activité, etc.

Scénario d'un cas d'utilisation

■ Demande d'un tableau de service (enseignant)

1. L'enseignant se présente devant un terminal,
 2. Le système affiche un message d'accueil,
 3. L'enseignant demande un relevé d'heures,
 4. Le système lui demande de s'authentifier,
 5. L'enseignant s'authentifie (login et mot de passe),
 6. Le système affiche les informations à l'écran,
 7. L'enseignant précise qu'il a terminé sa consultation et qu'il désire se déconnecter,
 8. Le système effectue la requête et retourne à l'écran d'accueil.
- Description simplifiée

Exemple de scénario d'un cas d'utilisation

■ Demande d'un tableau de service (enseignant)

◆ Pré-conditions

(avant que l'opération ne puisse avoir lieu)

- L'enseignant doit être inscrit à l'université,
- L'enseignant doit connaître ses identifiants,

◆ Post-condition

(si l'opération s'est bien déroulée)

- L'enseignant a pris connaissance des heures de cours qu'il a données.

4

“ Diagrammes de
classes ”



ÉCOLE NATIONALE SUPÉRIEURE
ÉLECTRONIQUE, INFORMATIQUE & RADIOPROGRAMMATION
BORDEAUX

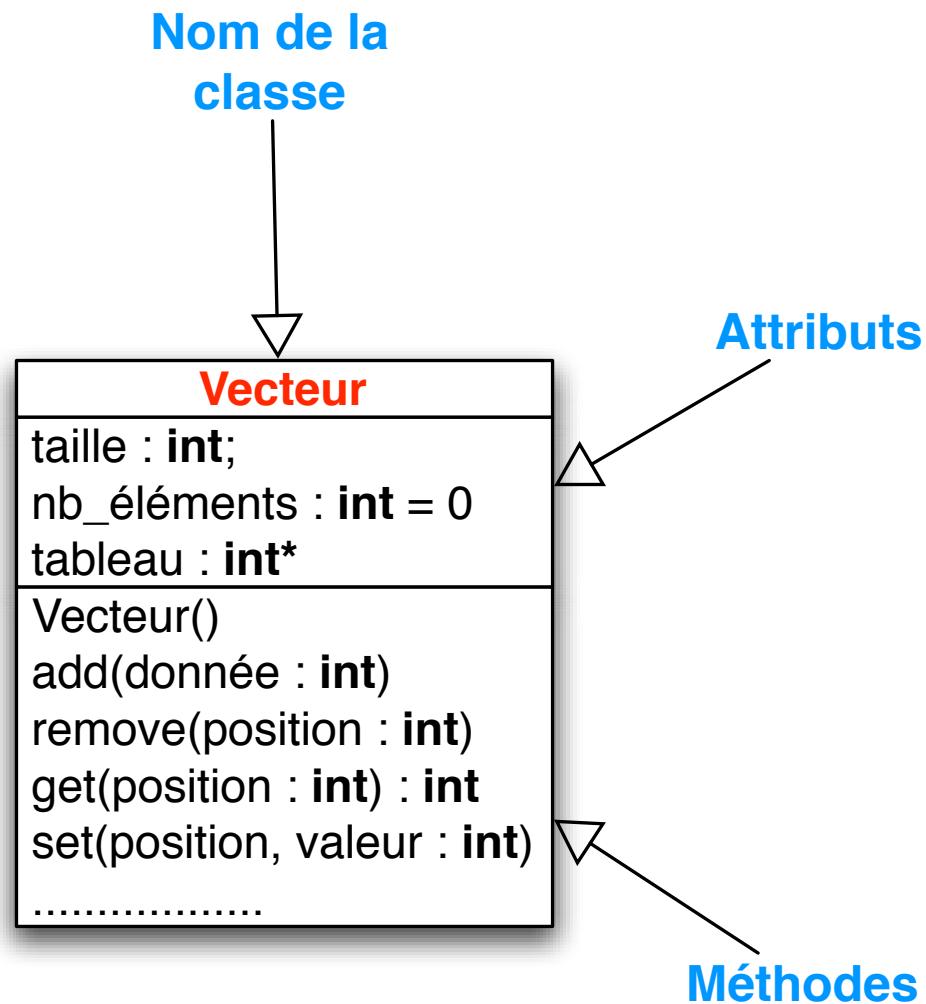
Les diagrammes de classes

- Un diagramme de classe se compose des éléments suivants,
 - ◆ Des classes,
 - ◆ Des interfaces (classes virtuelles),
 - ◆ Des collaborations,
 - ◆ Des relations (dépendances, associations, généralisation),
- Le diagramme de classes peut être défini à différents niveaux d'abstraction
 - ◆ Point de *vue conceptuel* (entités fournissant des services),
 - ◆ Point de *vue des spécifications* (raffinement fonctionnel des entités),
 - ◆ Point de *vue implémentation* (les classes que l'on doit mettre en oeuvre).

Définition d'une classe

- Une classe est la description élémentaire d'un ensemble d'objets qui partagent des attributs et des méthodes en communs.
- La définition des attributs et des méthodes n'est *pas obligatoire* :
 - ◆ Les attributs et les méthodes sont trop nombreux pour être représentés dans la majorité des cas...
 - ◆ On indique alors les responsabilités de la classe sous forme de texte (ce qu'elle doit implémenter comme service).
 - ◆ On peut dès la définition d'une classe préciser les droits d'accès aux méthodes et aux attributs (+, -, #)

Exemple de modélisation d'une classe



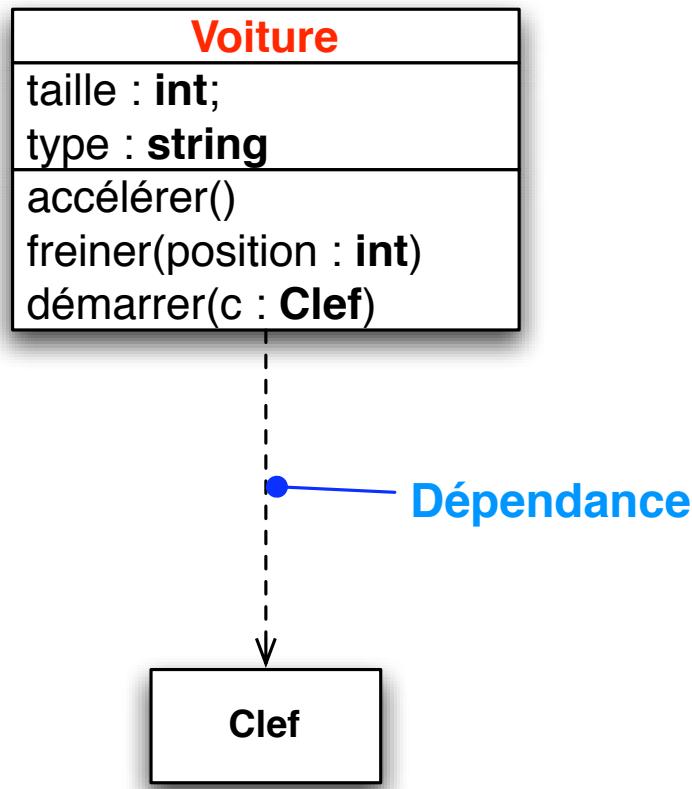
Introduction aux relations

- En UML, la manière dont les éléments sont interconnectés entre eux (logiquement ou structurellement) est modélisé sous forme de relation,
 - ◆ Les roues, le moteur, les portes, la carrosserie font partie du vocabulaire de construction d'une voiture mais ils n'existent pas de manière indépendantes.
- Il existe différents types de relation entre les éléments :
 - ◆ La dépendance,
 - ◆ L'utilisation,
 - ◆ La généralisation,
 - ◆ L'association,
 - ◆ L'agrégation.

Les relations : la dépendance

- La *dépendance* établit une *relation d'utilisation* entre 2 entités d'un même diagramme.
- La plus part du temps il s'agit d'une *dépendance d'utilisation*,
 - ◆ Argument d'une méthode par exemple,
 - ◆ On parle alors de “*relation d'utilisation*”,
- Cela permet d'identifier implications possibles des modifications à apporter dans une entité
 - ◆ Changement du comportement d'une des classe par exemple.

Les relations : la dépendance

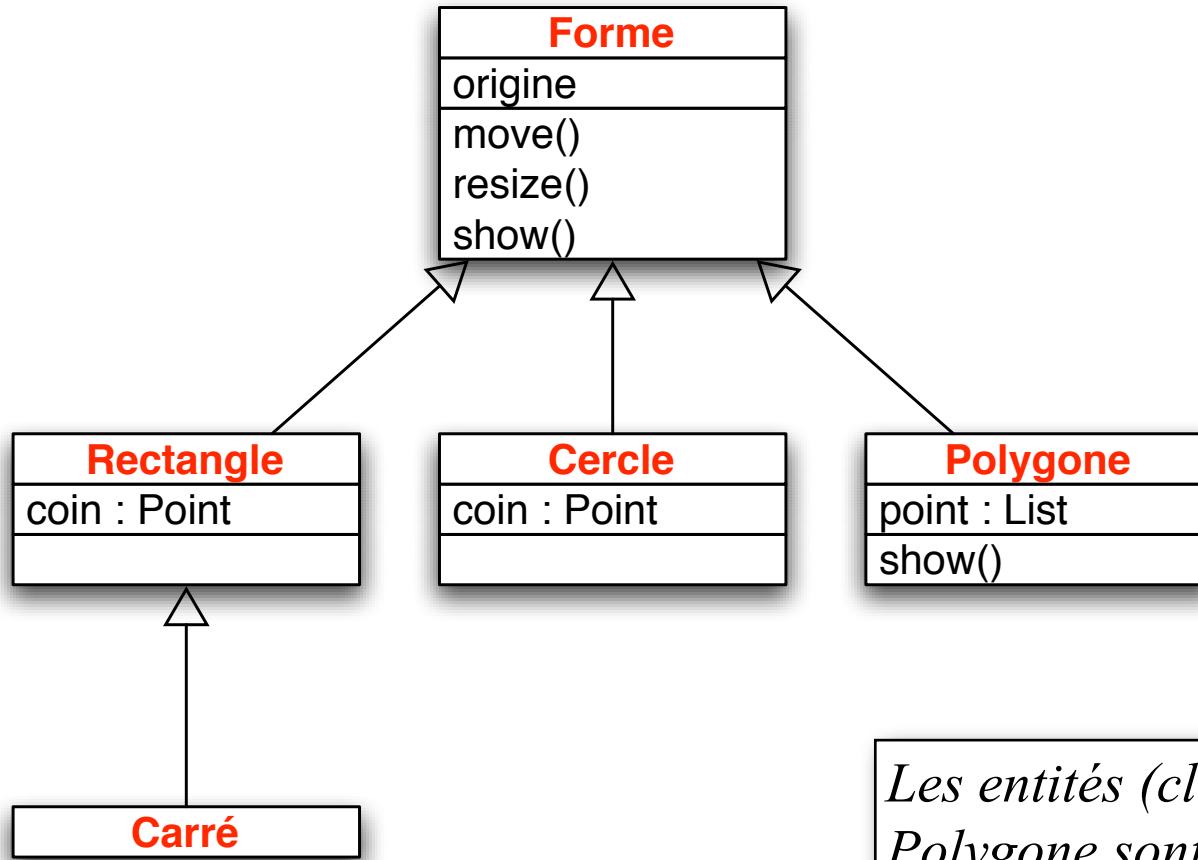


La classe “Voiture” dépend pour son utilisation de la classe “Clef”

Les relations : la généralisation

- La *généralisation* modélise la *notion d'héritage* qui existe dans les langages objets
 - ◆ La généralisation correspond à la notion “*est une sorte de*”,
 - ◆ Modélisation des relations parents / enfants,
- Cela implique que les entités issues d'une généralisation sont utilisables partout où leur classe mère peut l'être (mais pas l'inverse)
 - ◆ Généralisation (classe, classe) ou (classe, interface),
- Cette relation est modélisée par une *flèche pointant sur la classe mère*,

Les relations : la généralisation

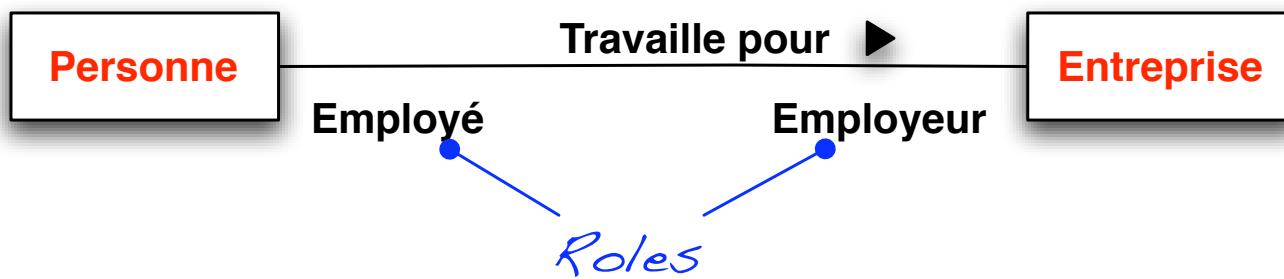
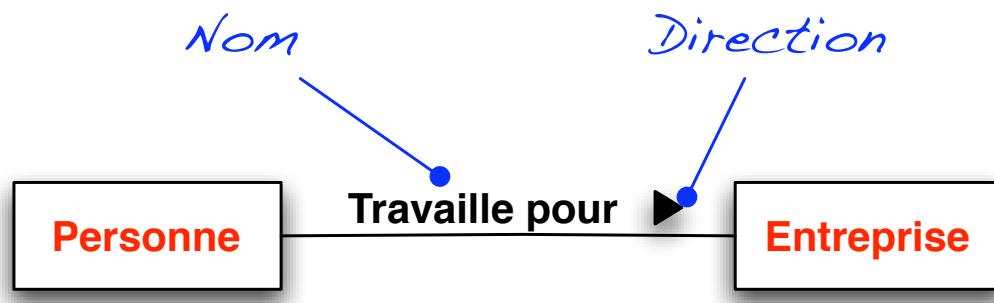
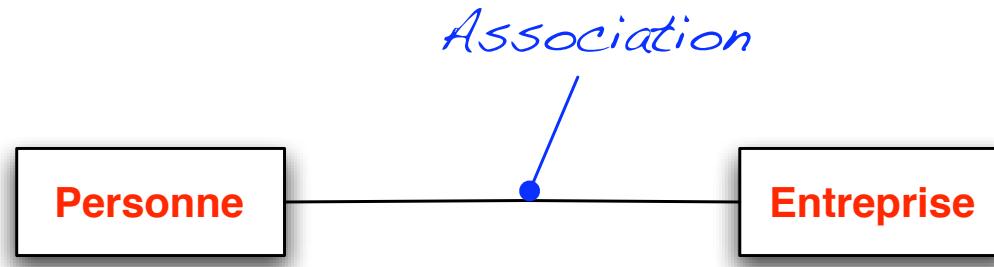


Les entités (classes) Rectangle, Cercle et Polygone sont des classe filles qui héritent de la classe mère Forme. De la même manière, l'entité Carré hérite de l'entité Rectangle.

Les relations : l'association

- Les *relations d'association* permettent de modéliser un *lien structurel* entre 2 éléments du modèle.
 - ◆ Si le lien joint 2 classes alors cela signifie qu'il existe une relation de navigabilité entre ces 2 classes,
- 4 décorations permettent de spécifier plus finement le lien entre les objets :
 - ◆ *Le nom* : décrire la nature de la relation entre les objets,
 - ◆ *La direction* : lever l'ambiguïté sur le nom,
 - ◆ *Les rôles* : indiquer le rôle spécifique de chacune des classes dans l'association,
 - ◆ *La cardinalité* : spécifie le nombre d'éléments affectés.

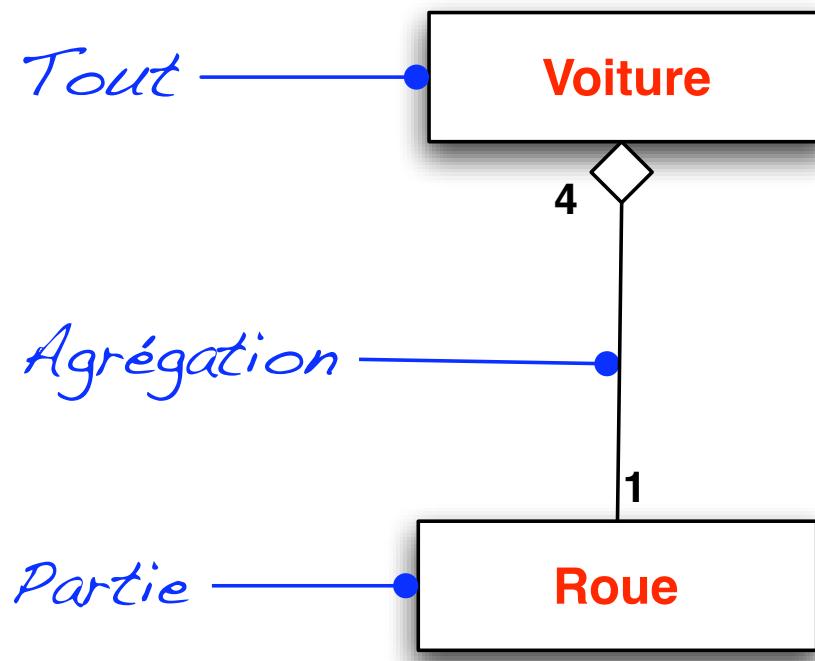
Les relations : l'association



Les relations : l'agrégation

- Dans l'association, les entités se trouvent au même niveau hiérarchique.
- Dans l'agrégation, il existe une relation hiérarchique entre les entités,
- *L'agrégation* répond à la question “*se compose de*” et modélise une *notion de possession*,
 - ◆ Notion de “*tout*” et de “*parties*”,
- Se note comme une association avec *un losange du côté du tout*.

Les relations : l'agrégation



Conclusion sur les relations

- Les relations doivent être spécifiées pour toutes les classes qui interagissent entre elles,
- Les relations sont des *liens statiques* entre les différentes entités (classes),
 - ◆ Les relations apparaissent quasi-exclusivement dans les diagrammes de classes,
- Il existe 2 autres types de relations utilisés dans la modélisation dynamique :
 - ◆ Les “*liens*” : flux permettant le transfert de messages entre objets.
 - ◆ Les “*transitions*” : utilisés dans la modélisation des machines à états finis.

Spécification des commentaires (Notes)

- Lors des phases de modélisation et de spécification, il est nécessaire de *documenter* ses modèles :
 - ◆ Contraintes *matérielles*,
 - ◆ Contraintes de *performance*,
 - ◆ *Choix techniques* réalisés,
 - ◆ *Références* à d'autres documents,
 - ◆ Explications techniques, etc.
- Dans le langage UML, il existe une sémantique pour les commentaires.

Modélisation des commentaires

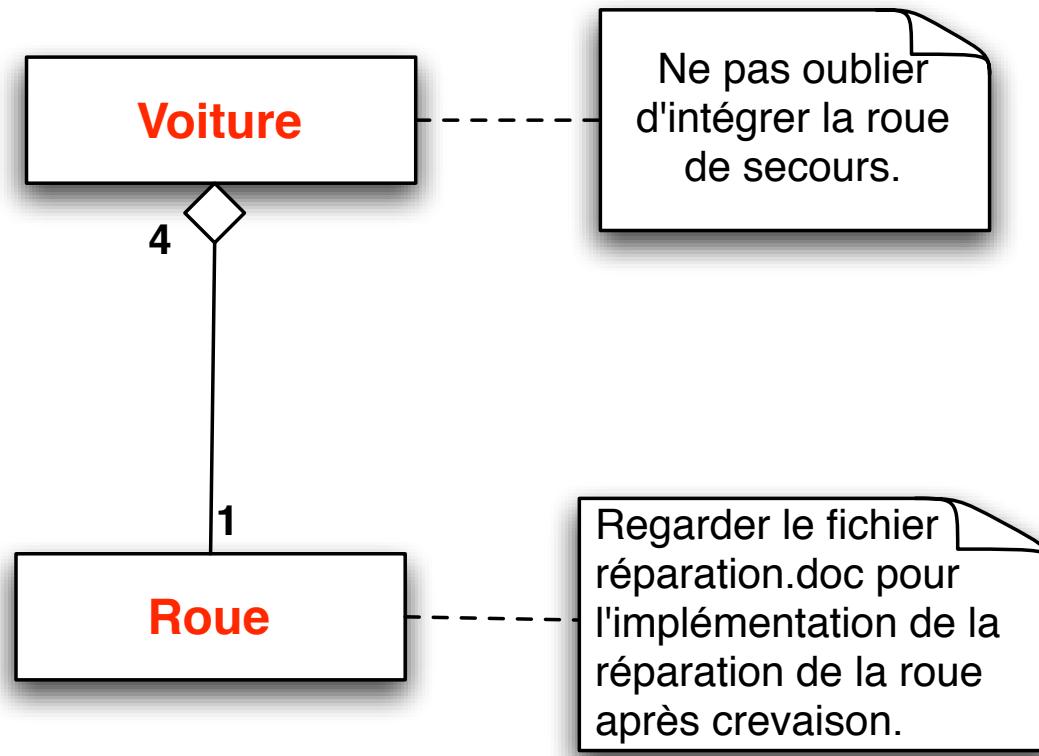


Diagramme de classes, exemple

- Nous allons maintenant essayer de modéliser une entreprise à l'aide d'un diagramme de classes :
 - ◆ Une *entreprise* est *composée* de plusieurs *services*, dans ces *services officie* un *certain nombre de personnes*.
 - ◆ Dans les locaux de l'entreprise se *trouvent* des *bureaux utilisés* par les *employés*, chaque bureau *porte* un *numéro* et *possède* un *numéro de téléphone*.
 - ◆ Les *employés sont reconnus* à l'aide de leur *nom* et de leur *numéro personnel*. Chaque *employé possède* un *statut* dans l'entreprise.

Diagramme de classes - Exemple (1)

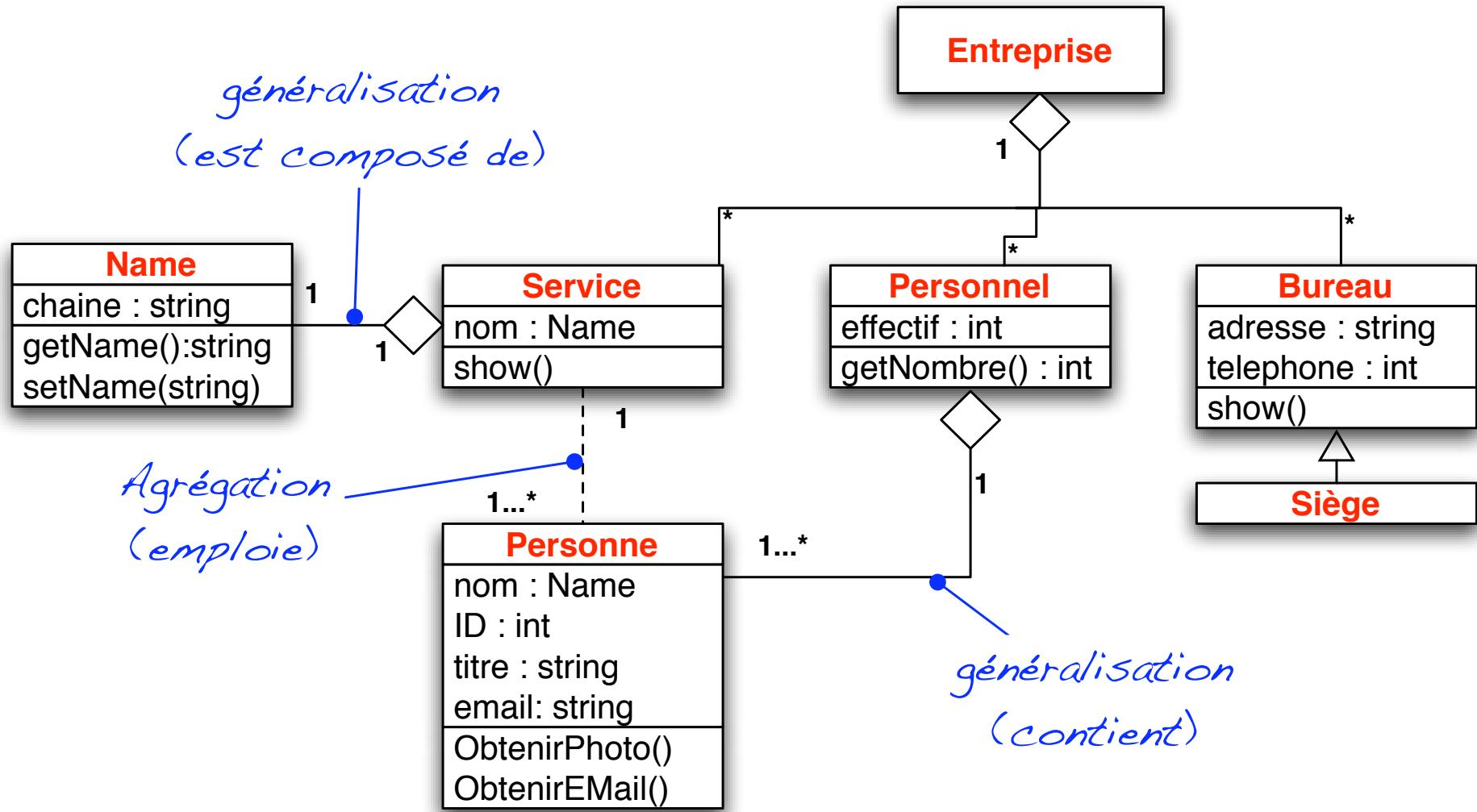


Diagramme de classes - Exemple (2)

- Nous allons maintenant nous attacher à essayer de modéliser une université :
 - ◆ L'*université* est *composée* d'*étudiants* qui *suivent* des *cours* dans les *filières* de leur choix.
 - ◆ Ces *cours* sont *dispensés* par des *enseignants* qui peuvent être *responsable* des filières.
 - ◆ Les *cours* sont *dispensés* dans *au moins* une *filière* et peuvent être *mutualisés* avec d'autres.
- A partir de là, réalisez le diagrammes de classes modélisant l'université.

Correction de l'exemple (1)

Université

Filière

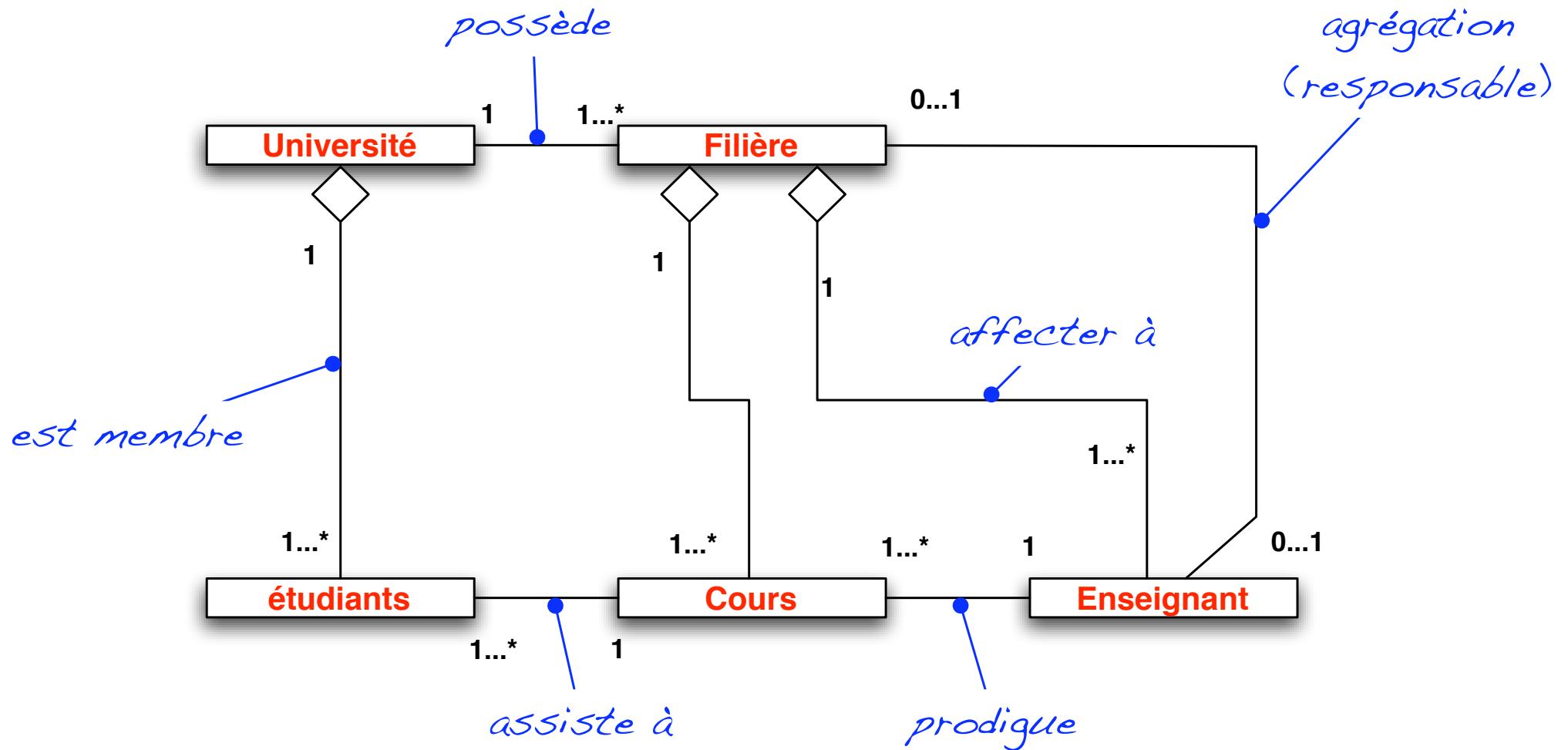
étudiants

Cours

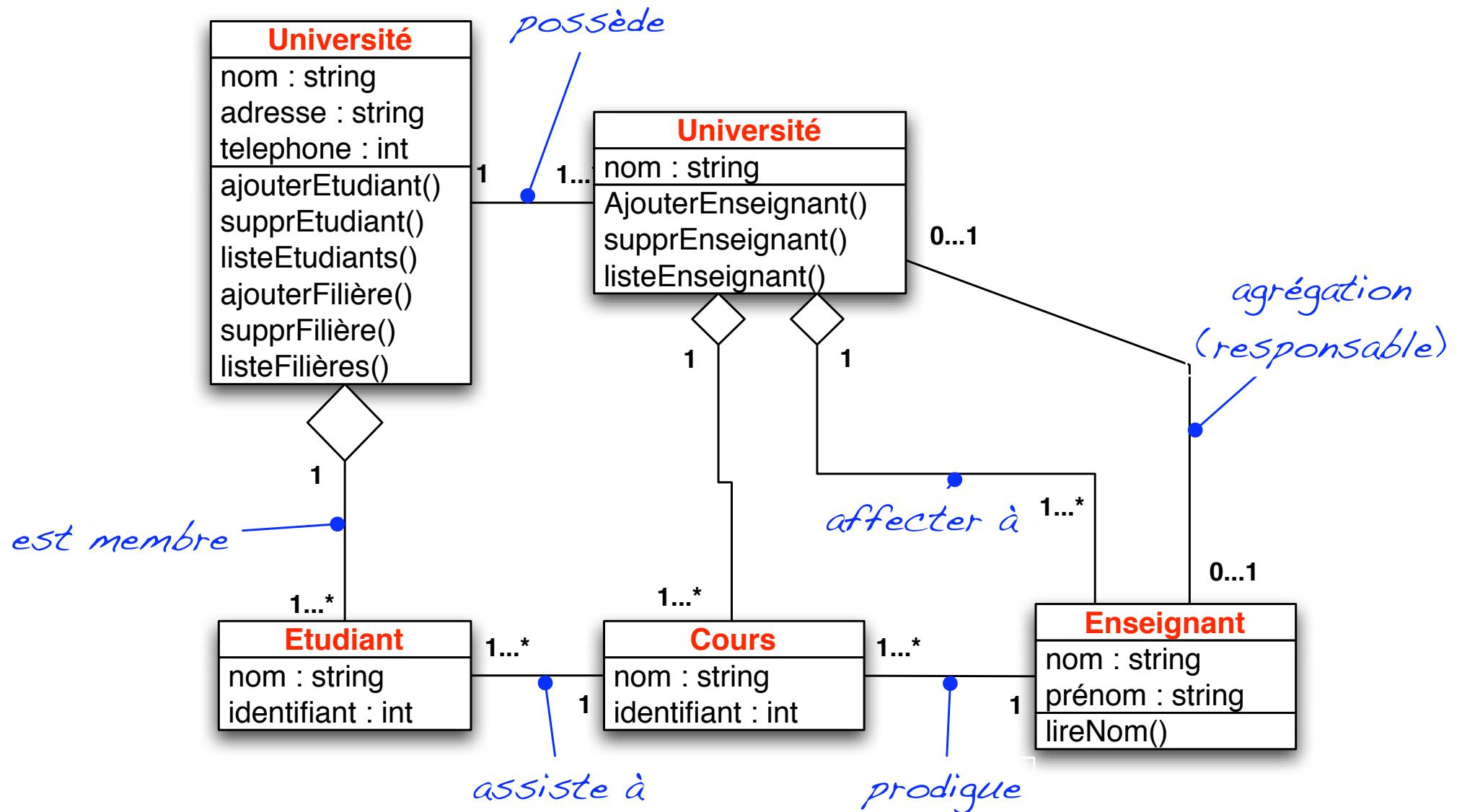
Enseignant

*Voici les 5 entités de base qui constituent
l'université que nous devons modéliser*

Correction de l'exemple (2)



Correction de l'exemple (3)

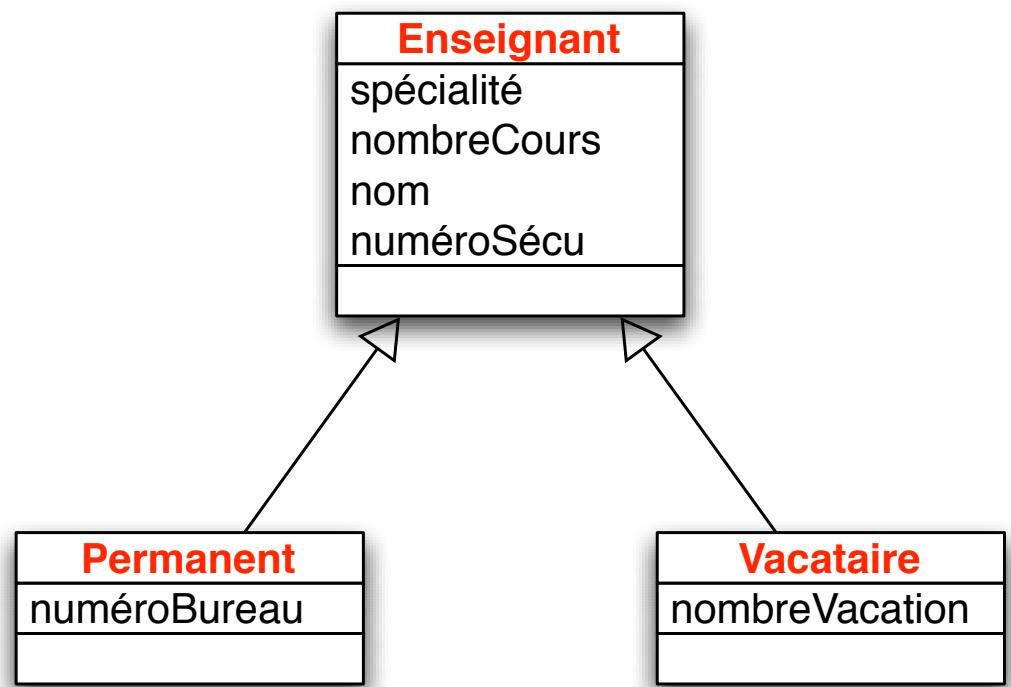


Exemple de travail itératif sur le modèle (1)

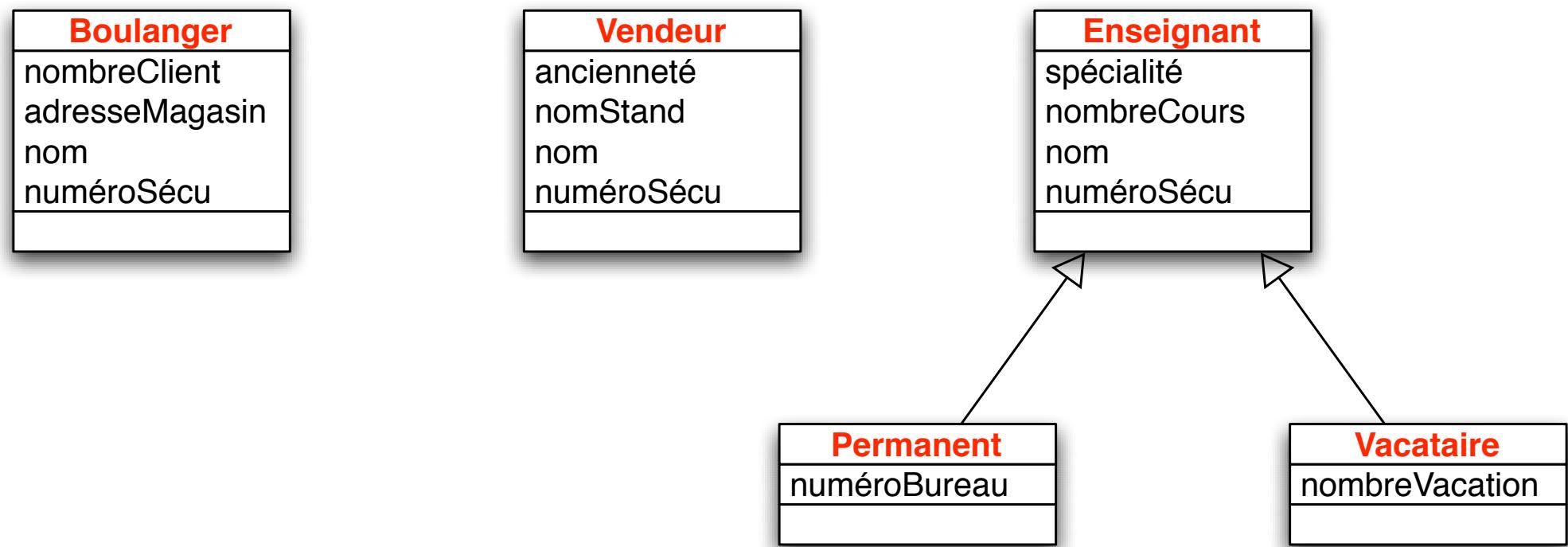
Permanent
numéroBureau
spécialité
nombreCours
nom
numéroSécu

Vacataire
nombreVacation
nombreCours
spécialité
nom
numéroSécu

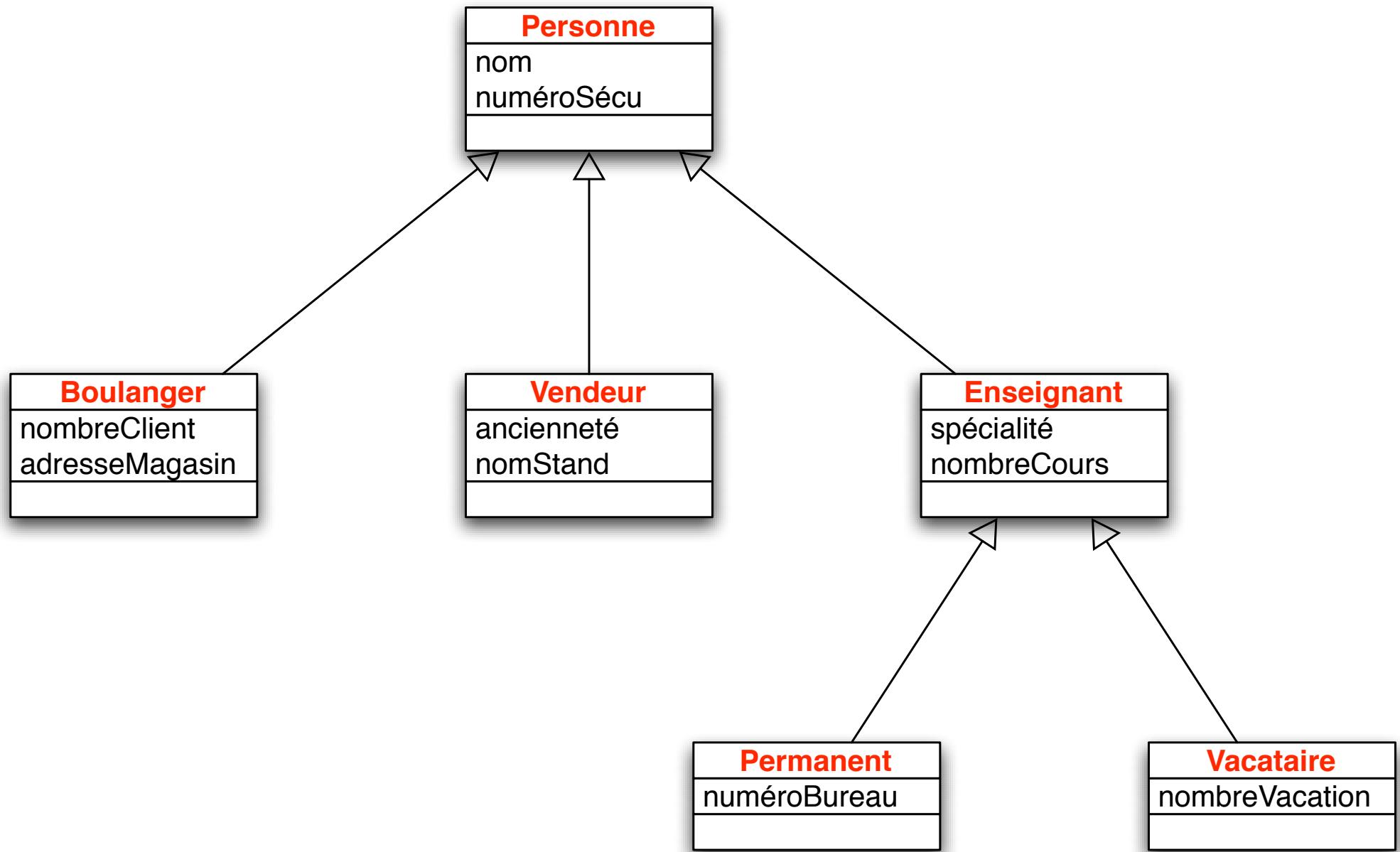
Exemple de travail itératif sur le modèle (2)



Exemple de travail itératif sur le modèle (3)



Exemple de travail itératif sur le modèle (4)



5

“ Implémentation
des diagrammes
de classes ”



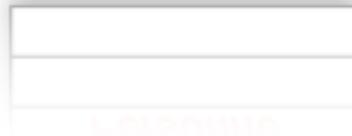
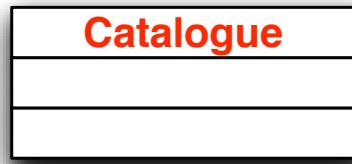
Et après les modèles que fait'on ?

- Les diagrammes permettent de spécifier de manière claire, les classes et les relations existantes entre elles.
 - ◆ Génération de code source à partir du modèle,
 - ◆ Les modèles ne sont pas spécifiques à un langage (choix à la génération => 1 modèle plusieurs implémentations possibles suivant les contraintes),
- A partir de tout diagramme UML, il est possible de remonter vers le cahier des charges
 - ◆ Les éléments du modèle ont une sémantique propre et unique.

Et après les modèles que fait'on ?

- Une fois que l'on a réalisé l'ensemble des diagrammes de classe, on va traduire les modèles dans un langage objet :
 - ◆ Indépendance du modèle vis-à-vis du langage utilisé pour l'implémentation.
 - ➡ Nous utiliserons ici du C++
- Chaque type d'entité et de relation possède une implémentation particulière,
 - ◆ Pas d'ambiguïté possible.

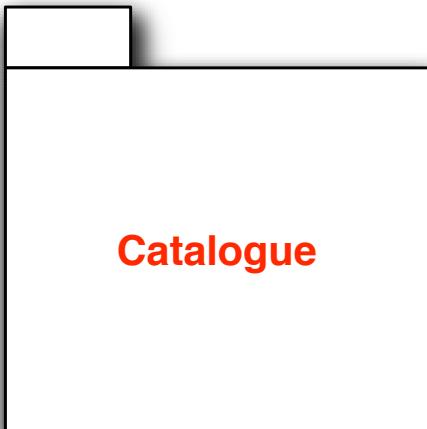
Transformation du modèle au code source



```
class Catalogue{  
    // ... .... ....  
};
```

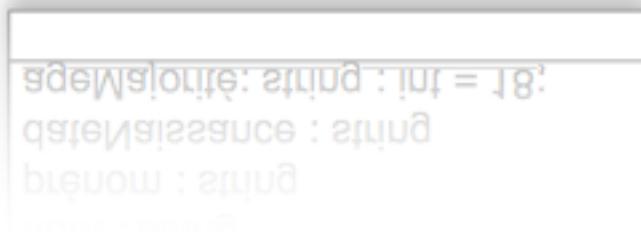
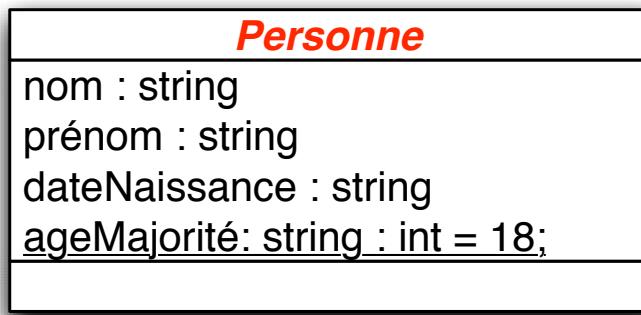
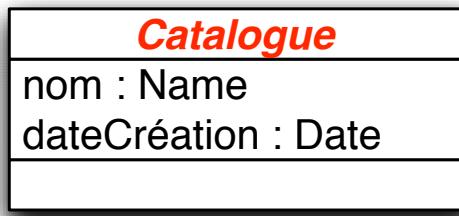
```
class Personne {  
    // ... .... ....  
};
```

Transformation du modèle au code source



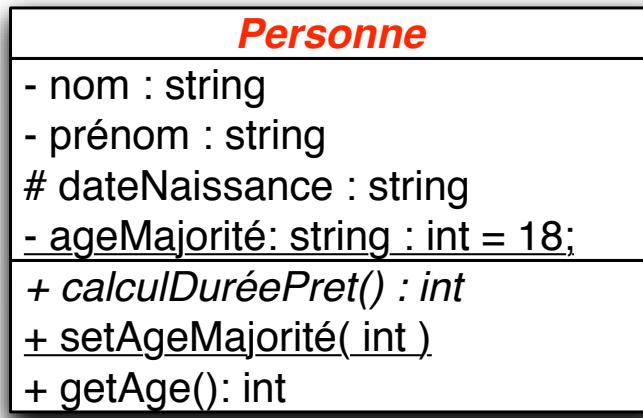
```
namespace Catalogue
{
    // ...
};
```

Transformation du modèle au code source



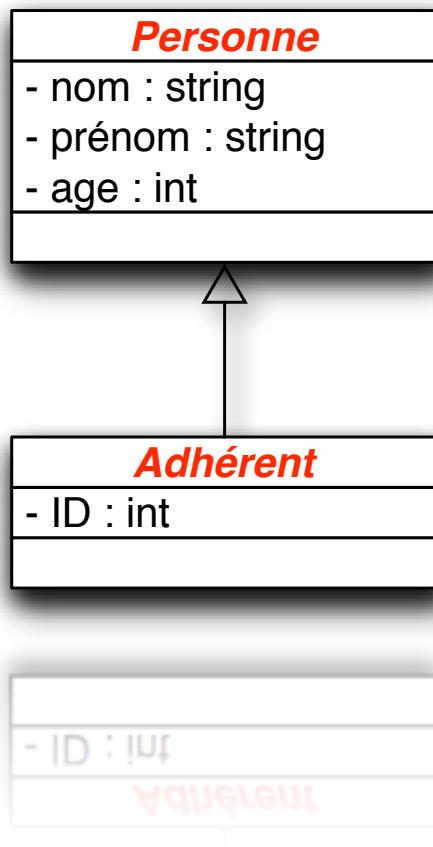
```
class Catalogue {  
private:  
    string nom;  
    DateTime dateCreation;  
    // ... ....  
}  
  
class Personne {  
private:  
    string nom;  
    string prénom;  
protected:  
    DateTime dateNaissance;  
private:  
    static int ageMajorite = 18;  
}
```

Transformation du modèle au code source



```
class Personne {  
    private:  
        string nom;  
        string prénom;  
        static int ageMajorite = 18;  
    protected:  
        DateTime dateNaissance;  
    public:  
        int CalculerDureePrest();  
        static void SetAgeMajorite(int  
            aMaj) {  
            // ... ....  
        }  
        public int GetAge() {  
            // ... ....  
        }  
}
```

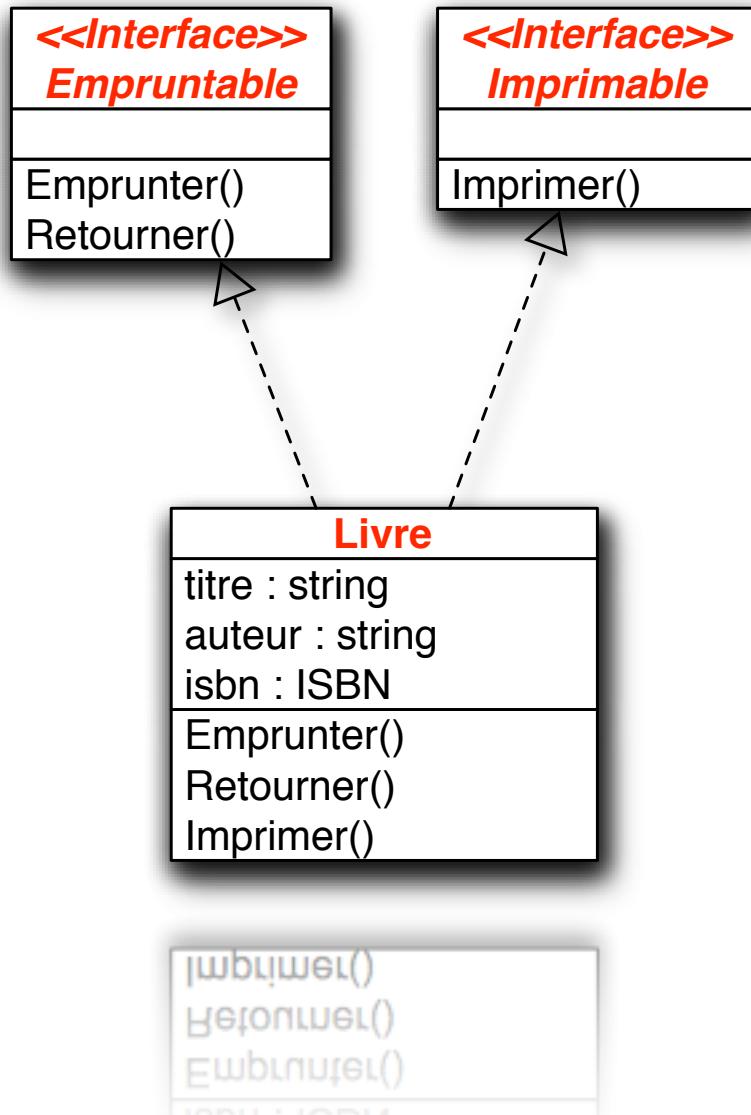
Implémentation d'une généralisation (héritage)



```
class Personne {
    // ...
}

class Adhérent : Personne {
    private int iD;
}
```

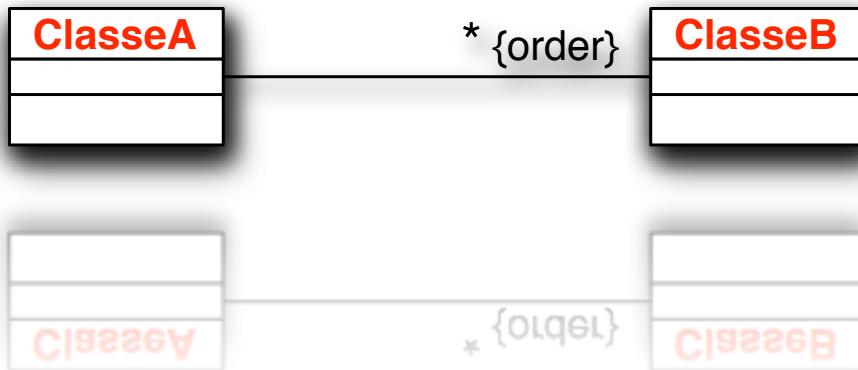
Implémentation d'un héritage multiple



```

class Livre : public Imprimable :  
public Empruntable {  
  
private:  
    string titre;  
    string auteur;  
    ISBN isbn;  
  
public:  
    void Imprimer(){  
        // ... .... ....  
    }  
    void Emprunter(){  
        // ... .... ....  
    }  
    void Retourner(){  
        // ... .... ....  
    }  
}
  
```

Implémentation de la composition (agrégation)



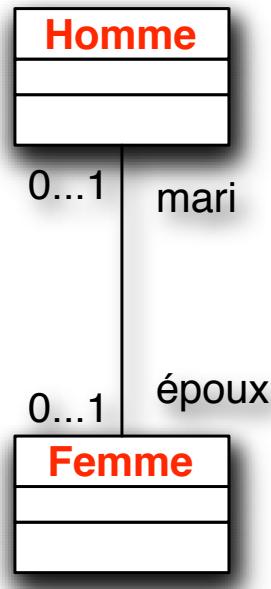
```

public class A1 {
private:
    ClasseB leB;
    // ... ...
}

public class A2 {
private:
    ClasseB[] lesB;
    // ... ...
}

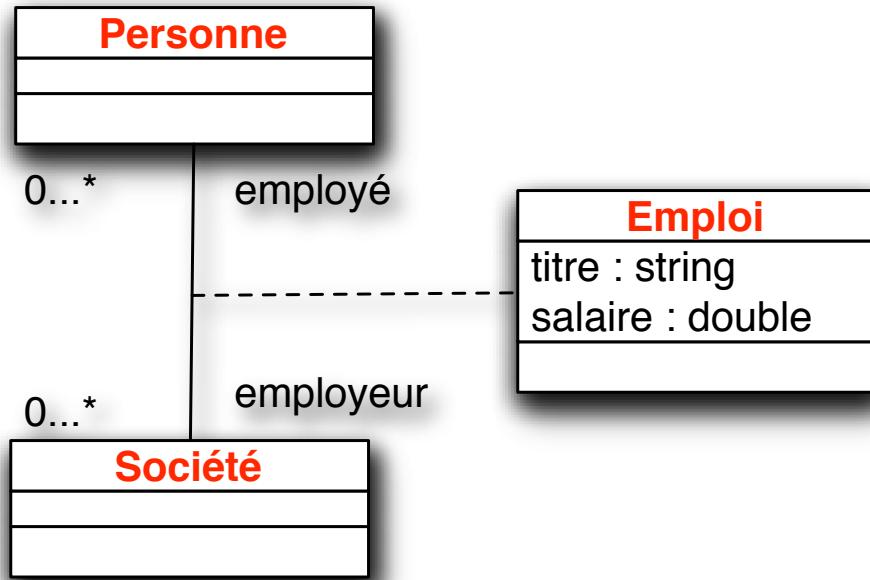
public class A3 {
private:
    ArrayList lesB;
    // ... ...
    lesB = = new ArrayList();
}
  
```

Implémentation de la composition (agrégation)



```
class Homme {  
private:  
    Femme épouse;  
    // ... ....  
}  
  
class Femme {  
private:  
    Homme mari;  
    // ... ....  
}
```

Implémentation d'une association entre classes



```
class Emploi {  
private:  
    string titre;  
    double salaire;  
    Personne employé;  
    Société employeur ;  
    // ... ....  
}
```





“ Les diagrammes
de séquence et
de collaboration ”



Les diagrammes de séquence

- Mise en évidence de l'*aspect temporel* des traitements (ordre chronologique de réalisation),
- Mise en évidence des *objets et des messages échangés* par les entités interagissant avec et/ou dans le système,
- La modélisation est basée sur l'affichage des objets participant à la séquence et à l'ordre des messages et des actions associées.

Définition de la sémantique du diagramme

- Les *diagrammes de séquence* permettent de modéliser les interactions entre les objets :
 - ◆ Communications *synchrones* (flèches pleines),
 - ◆ Communications *asynchrones* (flèches pointillés),
 - ◆ Les données transmises entre les objets (annotation des flèches),
- Ils permettent aussi de connaître pour chaque objet :
 - ◆ *Ses dates* de création et de destruction,
 - ◆ Sa *durée de vie*, si la création est externe au diagramme (ligne verticale en pointillés).

Diagramme de séquence - Ex. de l'ascenseur

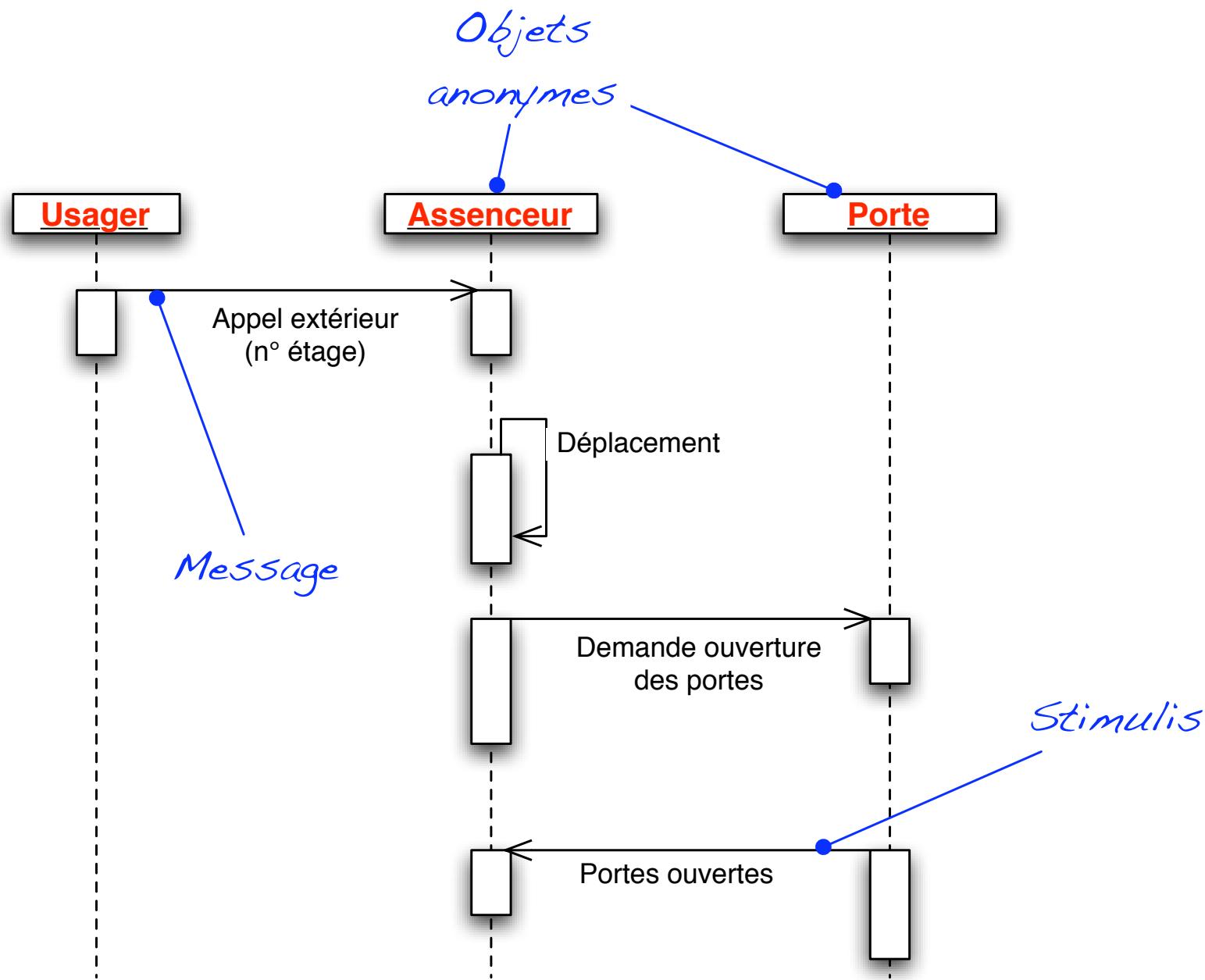
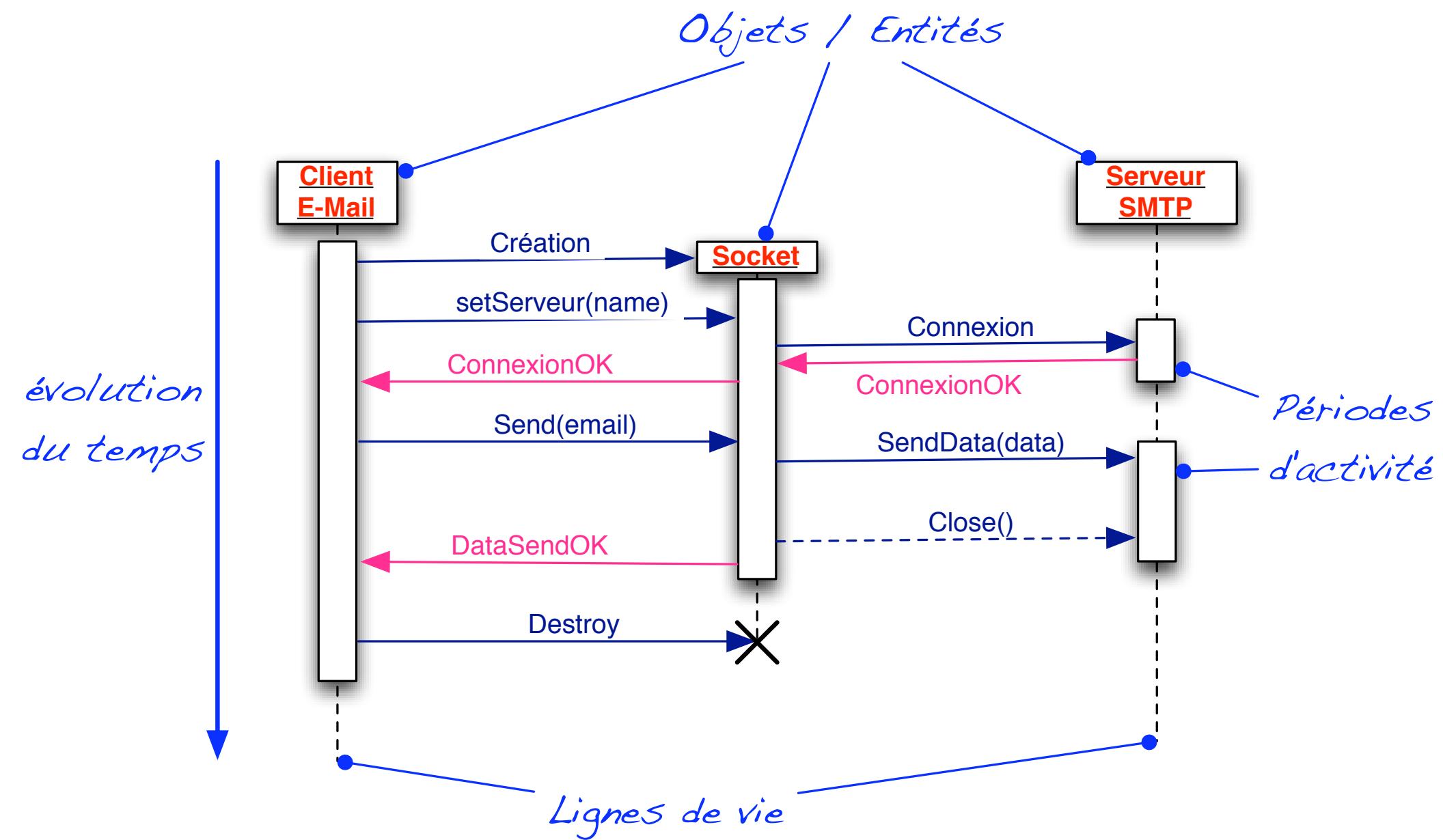


Diagramme de séquence - Ex. de client e-mail



Le diagramme de collaboration

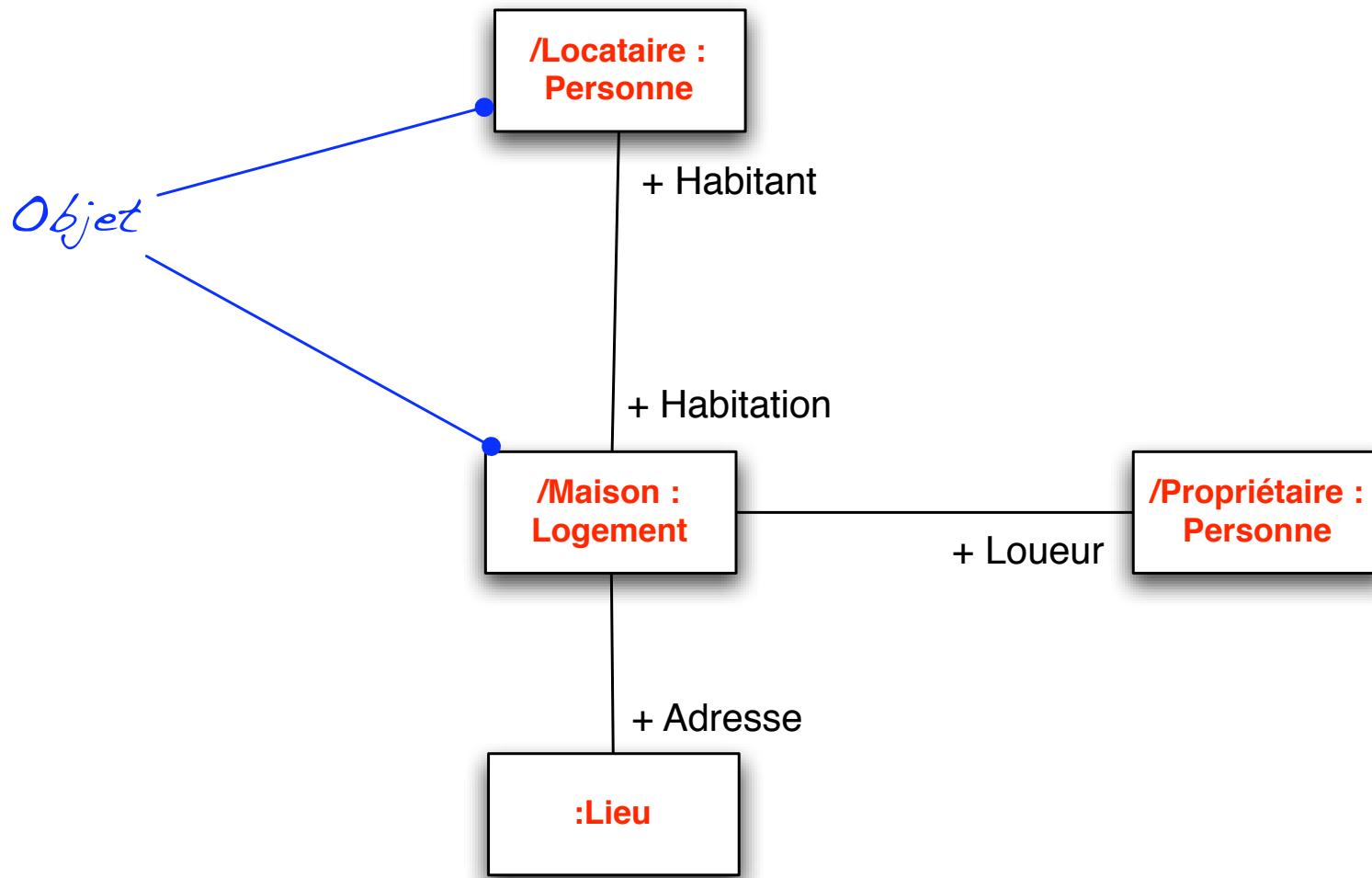
- Mise en évidence de *l'organisation des objets qui vont collaborer* pour effectuer une interaction.
 - ◆ On représente les objets qui vont intervenir (les sommets),
 - ◆ On modélise les liens qui vont modéliser les communications entre les objets (les arcs),
 - ◆ On annote les liens à l'aide des informations qui vont être échangées entre les entités,
- Visualisation claire du flux de contrôle dans le contexte de l'organisation structurelle.

Le diagramme de collaboration

■ 2 niveaux de représentation (d'abstraction)

- ◆ Le niveau spécification
 - Définition des classes et de leurs rôles,
- ◆ Le niveau instance
 - Définition des objets et des messages échangés,

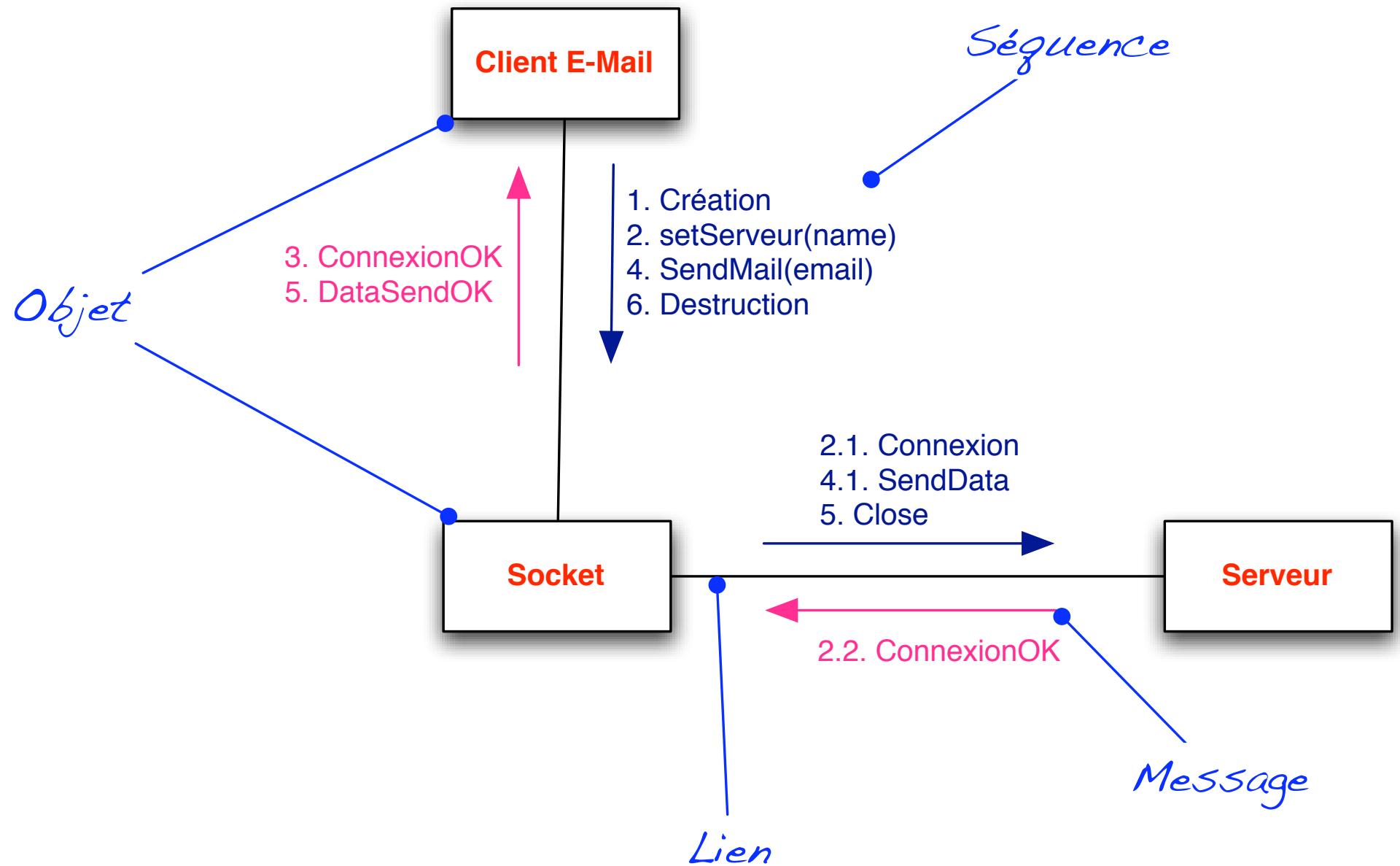
Modélisation d'un graphe de rôle



Le diagramme de collaboration

- La *modélisation temporelle* est en partie masquée,
 - ◆ Utilisation de la numérotation des séquences pour ordonner les traitements réalisés,
- Contrairement au diagramme de séquence, certaines informations ne sont pas représentées :
 - ◆ On ne modélise pas le ligne de vie des objets,
 - ◆ On ne modélise pas les temps d'activité des objets,
- Par contre on modélise explicitement les liens qui existent entre les différents objets.

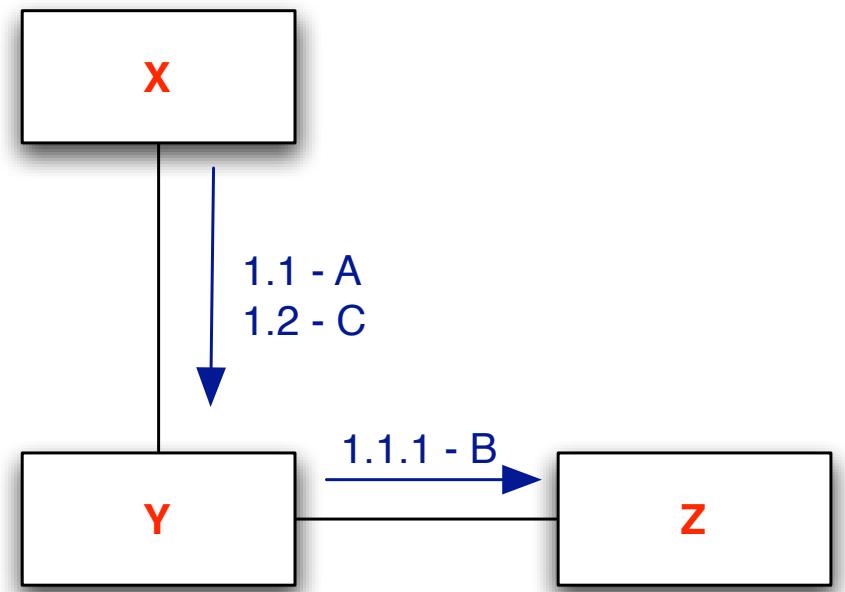
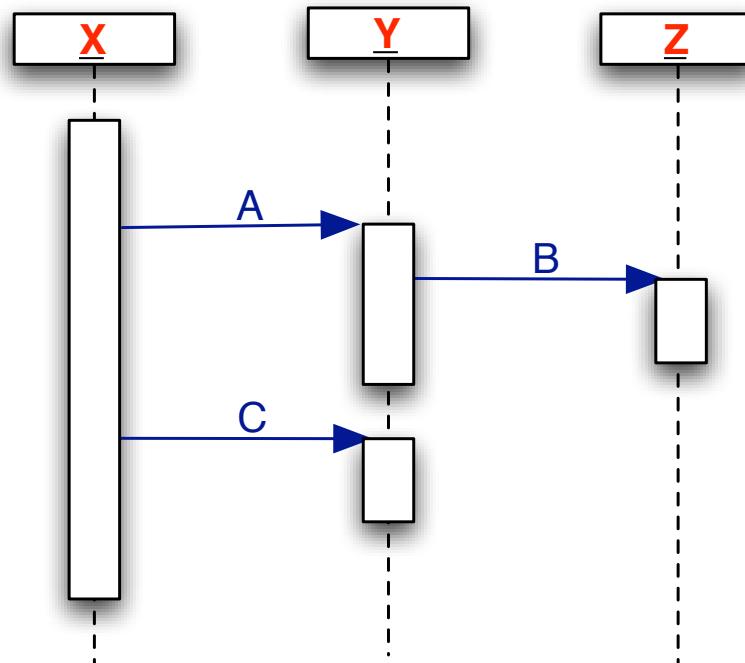
Exemple de diagramme de collaboration



Utilisation de ces diagrammes

- Ces diagrammes sont utilisés pour modéliser les *comportements dynamiques* d'un système,
 - ◆ Modéliser le fonctionnement d'un sous système,
 - ◆ Possibilité de joindre ces diagrammes aux cas d'utilisation afin de spécifier un scénario.
- Il est possible de *rajouter des contraintes* sur les diagrammes :
 - ◆ Contraintes *temporelles* (durée d'exécution),
 - ◆ Formaliser le flot de contrôle : ajouter des *pré/post-conditions*.

Equivalence entre les diagrammes



La transition d'un modèle de représentation à l'autre est une tâche aisée pour le concepteur. Ces 2 modèles expriment des informations différentes !

7

“ Diagrammes
d'états-
transitions ”

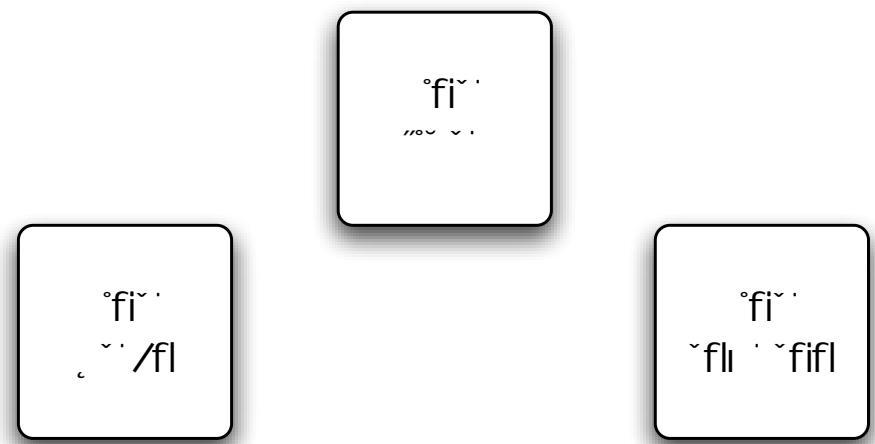
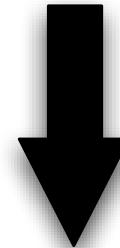


ÉCOLE NATIONALE SUPÉRIEURE
ÉLECTRONIQUE, INFORMATIQUE & RADIOPROGRAMMATION
BORDEAUX

SOCIETATIS COMMUNICATIO & AUDITAMQVI IN EQUIVOCIS TERRIS

La recherche des états

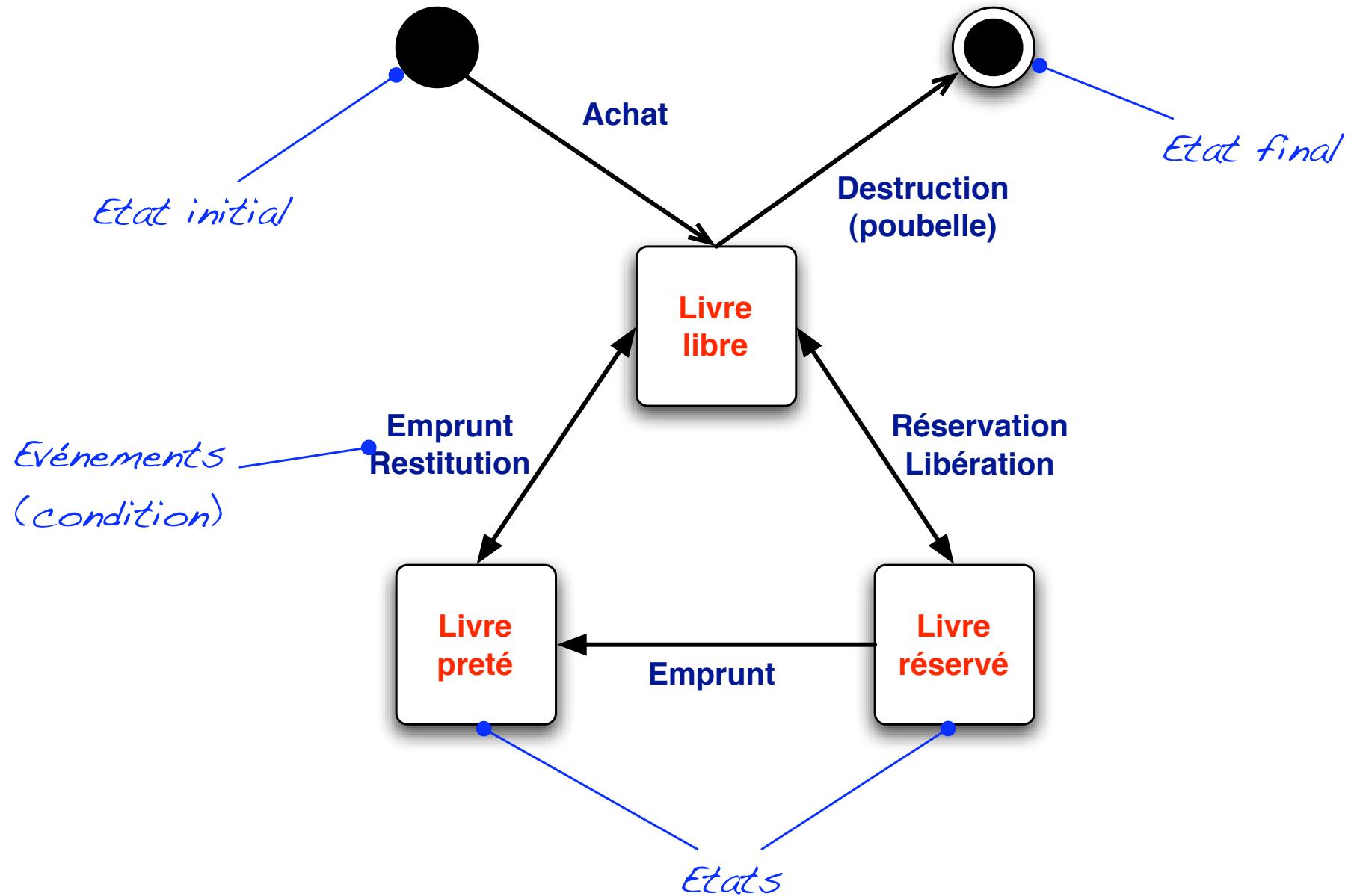
- L'état d'un objet est lié aux valeurs de ses attributs et de ses interactions avec les autres objets,
- Il est nécessaire de conserver les états significatifs,
- La réponse d'un événement à un stimuli dépend :
 - ◆ Du stimuli reçu par l'objet,
 - ◆ De l'état dans lequel se trouve l'objet.



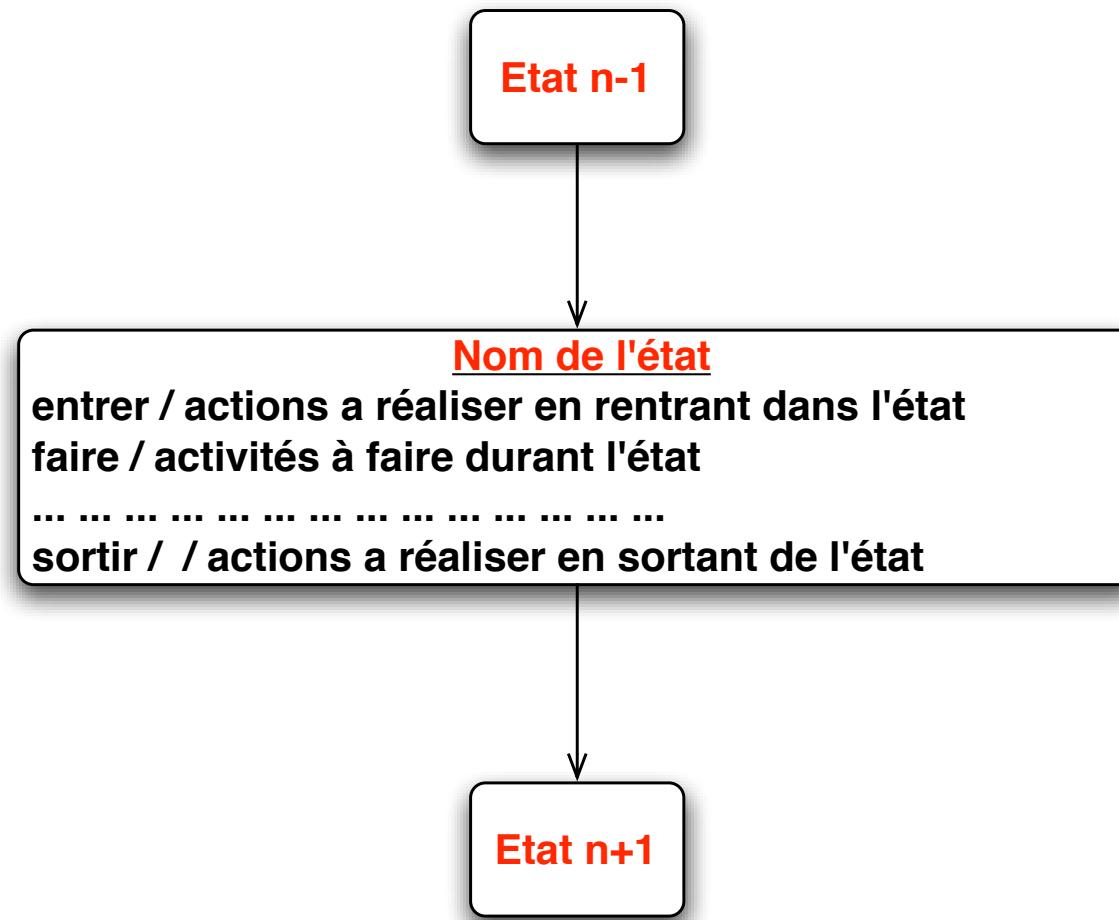
Règles liées au diagramme d'états

- Chaque transition (passage d'un état à un autre) est provoqué par un événement,
- Une transition n'a pas de durée (utilisation possible d'états de temporisation),
- La durée d'un état est l'intervalle qui s'écoule entre l'événement d'entrée et celui de départ.

Diagramme d'états d'un livre



Sémantique liée à la représentation



8

“ Et la conception
dans tout cela ... ”



Et la conception du système ...

- La phase de conception a pour objectif de chercher comment est implémenté le système,
 - ◆ L'analyse vise quant à elle à identifier ce qui doit être fait (et non comment).
- L'approche est hiérarchique s'attachant d'abord au grandes parties du système et à leurs interactions, puis raffinement progressif,
- Prise en compte des contraintes intrinsèques et explicites exprimées à la conception (langage, BDD, temps d'exécution, processeurs, périphériques, etc.).
- C'est une étape où on va faire intervenir des spécialistes de chaque discipline en fonction des technologies mises en oeuvre.

g

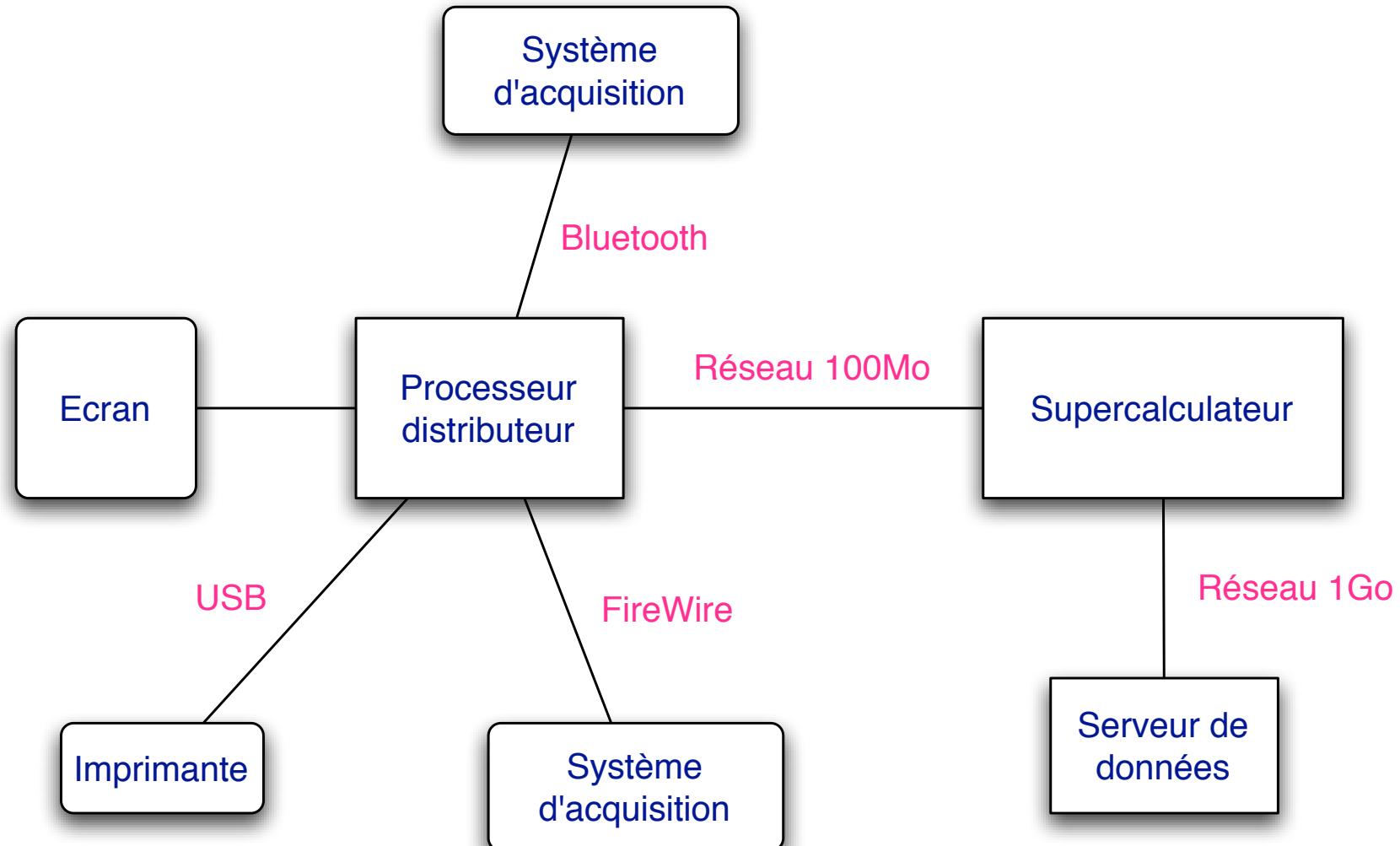
“ Diagrammes de
déploiement ”



Le diagramme de déploiement

- Ce modèle définit le diagramme de l'architecture matérielle du système ou de l'application,
- Il représente les différentes ressources matérielles à disposition du concepteur pur : processeurs et périphériques et la répartition de l'application sur le système hétérogène,
- Il précise aussi les liens de communication existants entre ces différentes ressources avec leurs caractéristiques.

Le diagramme de déploiement



10

“ Et la conception
dans tout cela ... ”



Conclusions sur UML

- Langage permettant d'appréhender la réalisation d'un système informatique complexe,
- Diagrammes qui permettent d'aider la réflexion, de permettre la discussion sur des bases normalisées et formalisées (clients, développeurs),
- Pas de solution unique mais un ensemble de solutions plus ou moins acceptables,
 - ◆ Contraintes clients (performances et fonctionnalités),
 - ◆ Architectures logicielles et matérielles à disposition,
- Des itérations successives du flot permettent d'aboutir à une solution.

Bibliographie

- **Modélisation avec UML**, *Robert Ogor*, ENST Bretagne, Mai 2003.
- **UML : Unified Modeling Language**, *Mireille Blay-Fornarino*, ESSI, Sophia Antipolis
- **UML**, *Martin Fowler*, Collection “Le tout en poche”, CampusPress
- **Le Guide de l’utilisateur UML**, *Grady Booch and al.*, Edition Eyrolles.