Modélisation avec UML

Vue générale du cours

1) Introduction au langage de modélisation UML
- points de vue et diagrammes
- cas d'utilisation, analyse, conception, implémentation

2) Le diagramme des cas d'utilisations
- acteur
- cas d'utilisation et scénario

3) Notion de classes et objets et leur diagramme
- introduction aux classes, aux objets
- notion de relation, de composition et d'héritage
- recherche d'un diagramme de classes à partir du cahier des charges

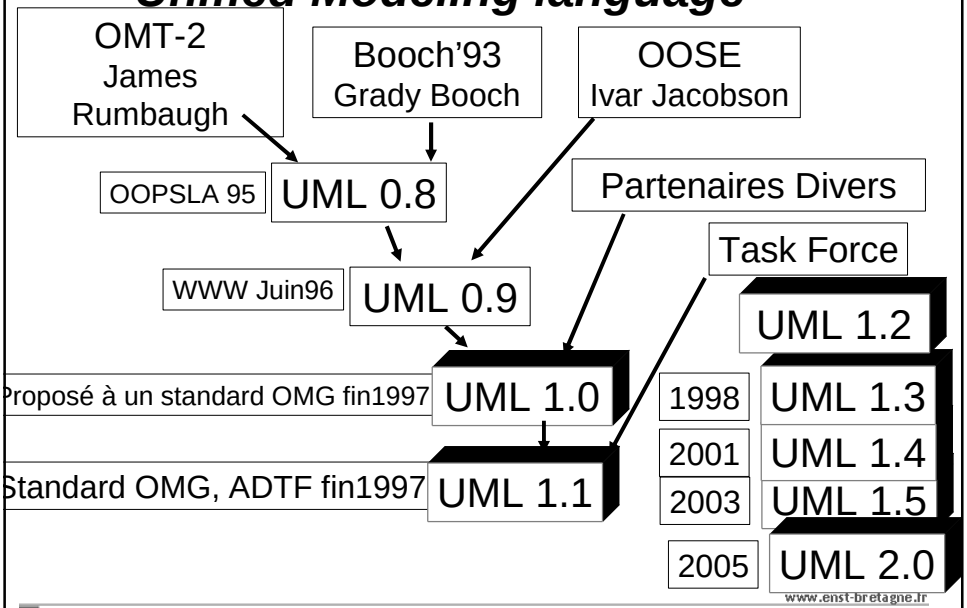
4) Modèle dynamique
- diagramme de séquences, de collaboration, d'état et d'activité
- réalisation des cas d'utilisation par les diagrammes de séquences
- réalisation des cas d'utilisation par les diagrammes de collaboration

5) Conception
- diagramme de déploiement et de composants

6) Le langage de contrainte OCL

1) Introduction au langage de modélisation UML

Unified Modeling language



Pourquoi modéliser

- Un modèle est une simplification de la réalité qui permet de mieux comprendre le système à développer.
- Il permet
  - De visualiser le système comme il est ou comme il devrait l'être.
  - De valider le modèle vis à vis des clients
  - De spécifier les structures de données et le comportement du système.
  - De fournir un guide pour la construction du système.
  - De documenter le système et les décisions prises.

Les principes de la modélisation

1) Le modèle doit être connecté au monde réel

2) Un modèle peut être exprimé avec différents niveaux de précision

3) Un simple modèle n'est pas suffisant, il y a plusieurs façons de voir un système.
   - plan de masse
   - vue de face, de coté, …
   - plan des niveaux
   - plan électrique
   - plan de plomberie
   - plan des calculs de construction

Qu'apporte la modélisation objet

- Plus grande indépendance du modèle par rapport aux fonctionnalités demandées.
- Des fonctionnalités peuvent être rajoutées ou modifiées, le modèle objet ne change pas.
- Plus proche du monde réel.

Les objectifs d'UML

- Représenter des systèmes entiers
- Etablir un couplage explicite entre les concepts et les artefacts exécutables
- Prendre en compte les facteurs d'échelle
- Créer un langage de modélisation utilisable à la fois par les humains et les machines

Recherche d'un langage commun :

- Utilisable par toutes les méthodes
- Adapté à toutes les phases du développement
- Compatible avec toutes les techniques de réalisation

UML un langage

- UML n 'est pas une méthode
- UML est un langage de modélisation objet
- UML a été adopté par toutes les méthodes objet
- UML est dans le domaine public, c'est une norme

UML un langage pour

- visualiser
  - chaque symbole graphique a une sémantique
- spécifier
  - de manière précise et complète, sans ambiguïté,
- construire
  - les classes, les relations SQL peuvent être générées automatiquement
- documenter
  - les différents diagrammes, notes, contraintes, exigences seront présentés dans un document.

UML et les domaines d'utilisation

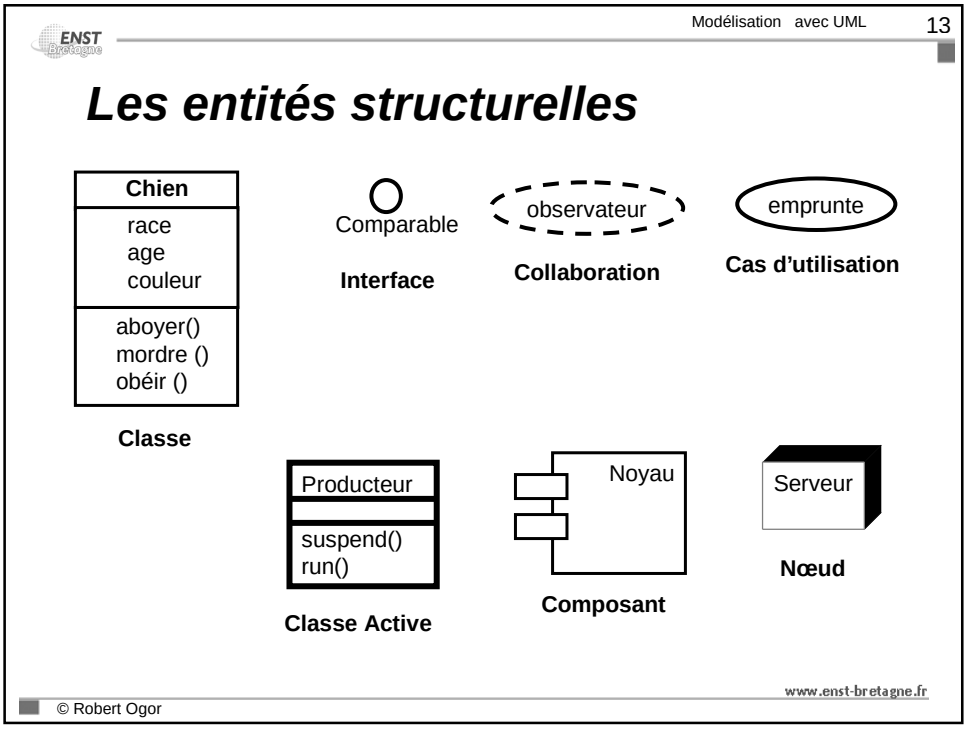
- Systèmes d'information des entreprises
- Les Banques et les services financiers
- Télécommunications
- Transport
- Défense et aérospatiale
- Scientifique
- Applications distribuées par le WEB

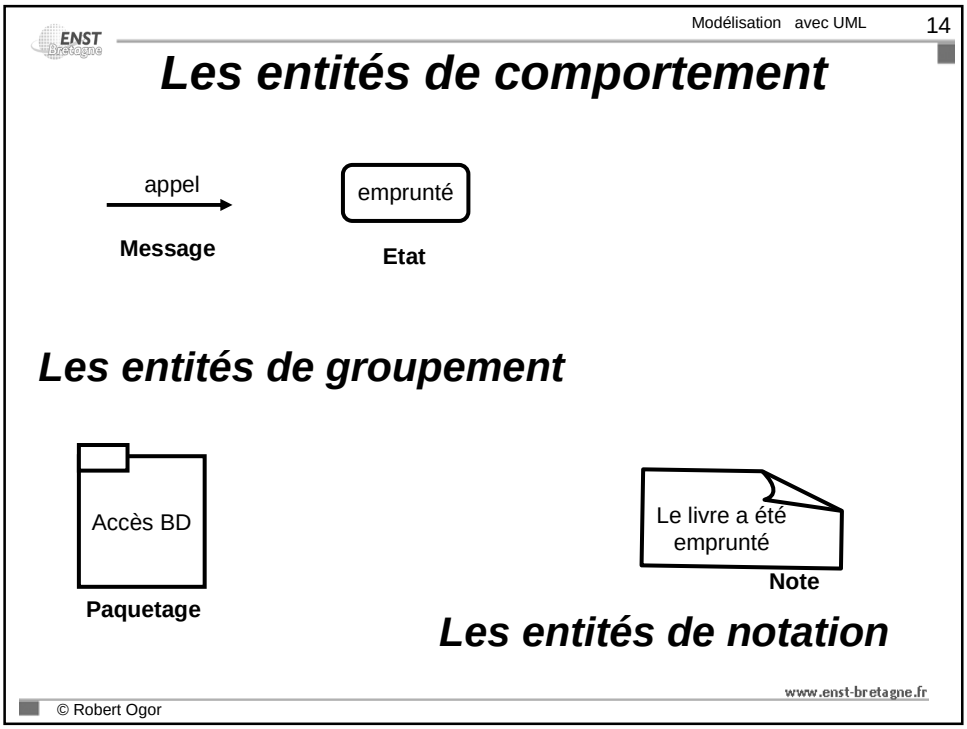
Les trois éléments de base en UML

1) les blocs de base pour construire
   - les entités utilisées → Entités structurelles, Entités de comportement, Entités de regroupement, Entité d'annotation
   - la notion de relation
   - les diagrammes

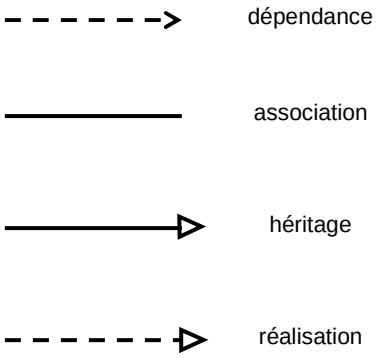
2) les règles à observer pour utiliser ces blocs de base
   - règles sémantiques
   - règles de présentation

3) les mécanismes communs
   - spécification
   - présentation
   - extension des modèles

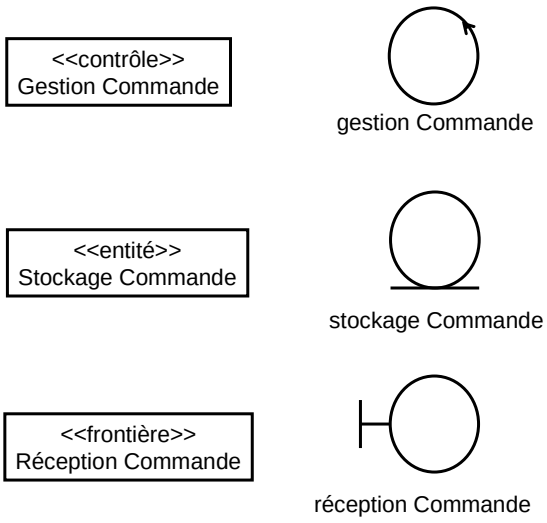

Modélisation avec UML 13
Les entités structurelles
Chien
race
age
couleur
aboyer()
mordre ()
obéir ()
Classe
Comparable
Interface
observateur
Collaboration
emprunte
Cas d'utilisation
Producteur
suspend()
run()
Classe Active
Noyau
Composant
Serveur
Nœud
www.enst-bretagne.fr
© Robert Ogor

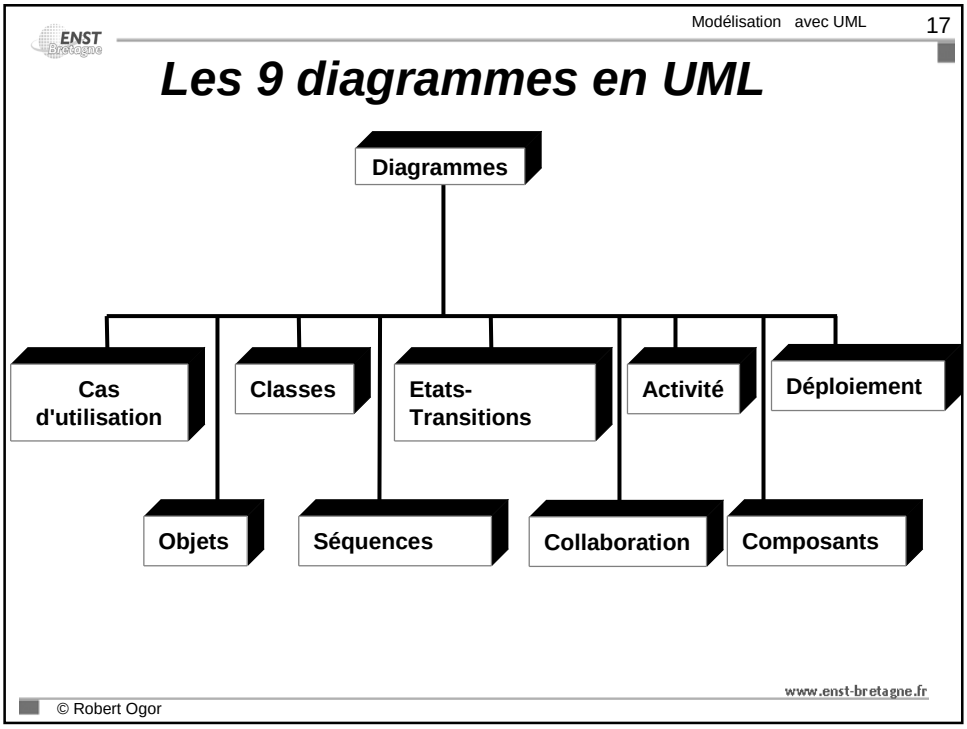

Modélisation avec UML 14
Les entités de comportement
appel
Message
emprunté
Etat
Les entités de groupement
Accès BD
Paquetage
Le livre a été emprunté
Note
Les entités de notation
www.enst-bretagne.fr
© Robert Ogor

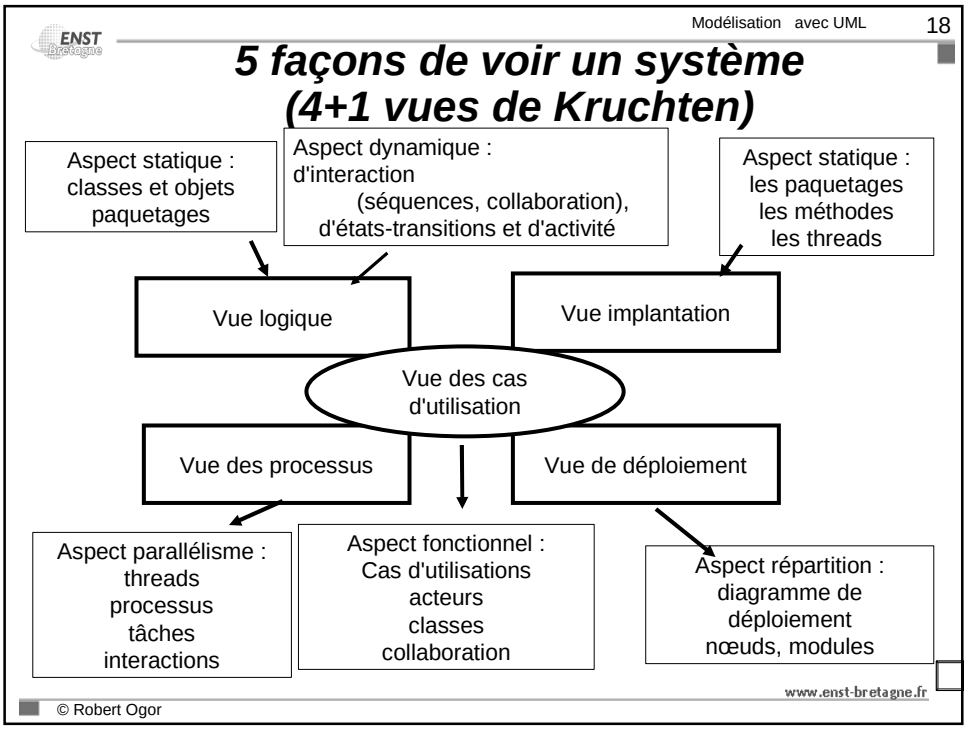
Les relations



Stéréotypes et icônes associées




Modélisation avec UML 17
Les 9 diagrammes en UML
Diagrammes
Cas d'utilisation
Classes
Etats-Transitions
Activité
Déploiement
Objets
Séquences
Collaboration
Composants
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML 18
5 façons de voir un système
(4+1 vues de Kruchten)
Aspect statique : classes et objets paquetages
Aspect dynamique : d'interaction (séquences, collaboration), d'états-transitions et d'activité
Aspect statique : les paquetages les méthodes les threads
Vue logique
Vue implantation
Vue des cas d'utilisation
Vue des processus
Vue de déploiement
Aspect parallélisme : threads processus tâches interactions
Aspect fonctionnel : Cas d'utilisations acteurs classes collaboration
Aspect répartition : diagramme de déploiement nœuds, modules
www.enst-bretagne.fr
© Robert Ogor

Point de vue des cas d'utilisation

- Vue du système par ses utilisateurs finaux
- Regroupe le comportement du système selon
  - Priorité: critique, important, accessoire
  - Risques à circonscrire
  - Options disponibles
  - Couverture de l'architecture
  - Autres objectifs tactiques et contraintes

Point de vue logique

- Décomposition orientée-objet
  - Décomposition en objets et classes
  - Regroupement en paquetages.
  - Connexions par héritage, association, etc.
  - Accent sur l'abstraction, l'encapsulation, l'uniformité.
  - Réalisation des scénarios des cas d'utilisation.

Point de vue processus

- Décomposition en tâches et processus
- Regroupement des groupes de processus
- Communication
- Information sur les caractéristiques suivantes
  - Disponibilité, fiabilité
  - Intégrité, performance
  - Contrôle

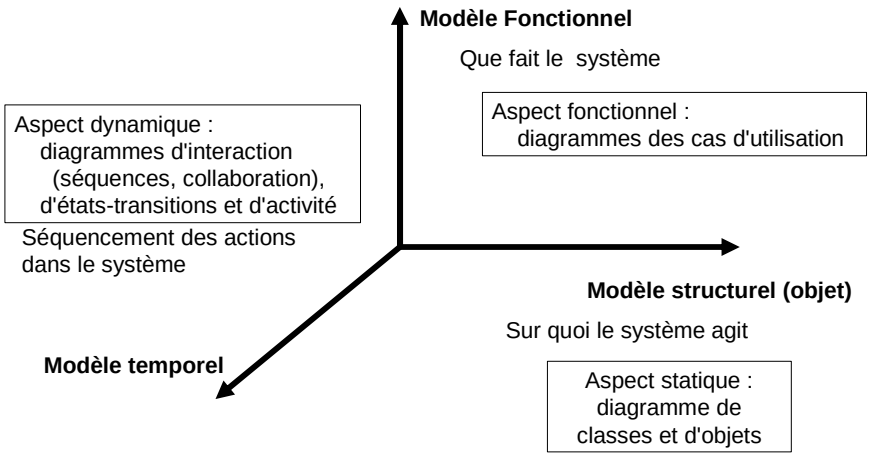
Point de vue implantation

- Décomposition en modules et niveaux
- Regroupement de modules en paquetages
- Organisation des sous-systèmes en niveaux pour :
  - Réduire le couplage et la visibilité
  - Augmenter la robustesse
- Information sur les caractéristiques suivantes :
  - Facilité de développement
  - Potentiel de réutilisation
  - Gestion de configuration

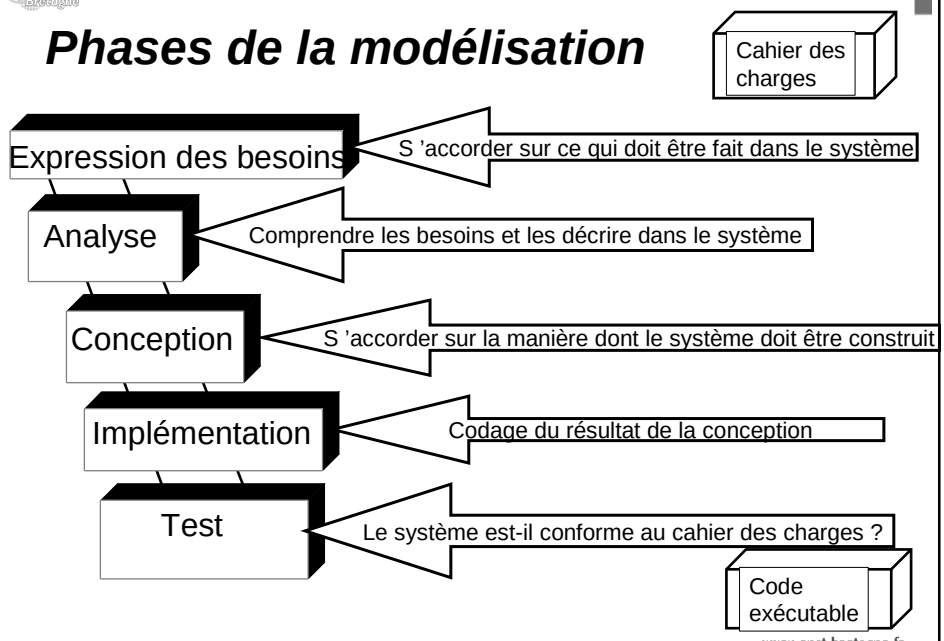
Point de vue déploiement

- Décomposition en nœuds d'exécution
- Rôle d'un nœud
- Inter-connectivité, topologie
- Information sur les caractéristiques suivantes :
  - Performance, disponibilité
  - Installation, maintenance

Les trois composantes d'une modélisation



Phases de la modélisation



Rôle de l'expression des besoins

- Permettre une meilleure compréhension du système
- Comprendre et structurer les besoins du client
  - clarifier, filtrer et organiser les besoins, ne pas chercher l'exhaustivité
- Une fois identifiés et structurés, ces besoins :
  - définissent le contour du système à modéliser (ils précisent le but à atteindre),
  - permettent d'identifier les fonctionnalités principales (critiques) du système.

Expression des besoins

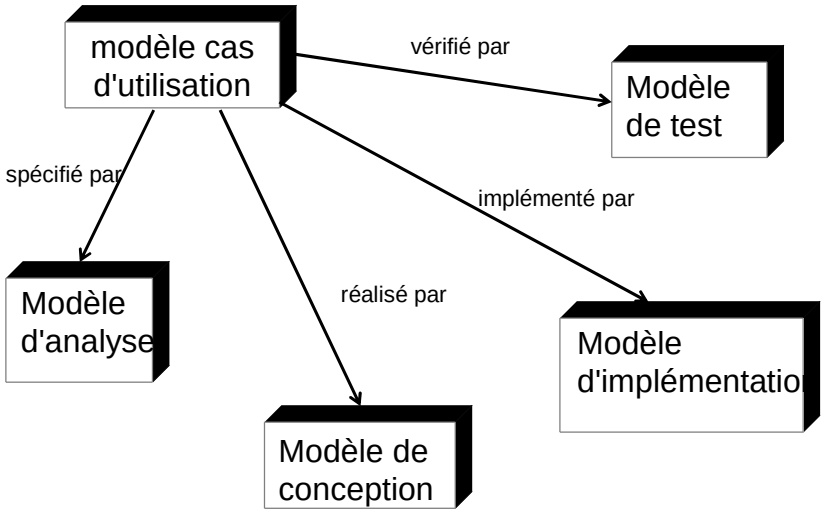
- Comprendre le contexte du système en définissant un modèle du domaine et du métier
- Recenser les besoins fonctionnels et les définir par des cas d'utilisations
- Noter les contraintes, exigences non fonctionnelles.

Le modèle du domaine regroupe les objets qui se situent dans le contexte du système :
- entités existantes ou événements qui s'y produisent.

Exigences non fonctionnelles

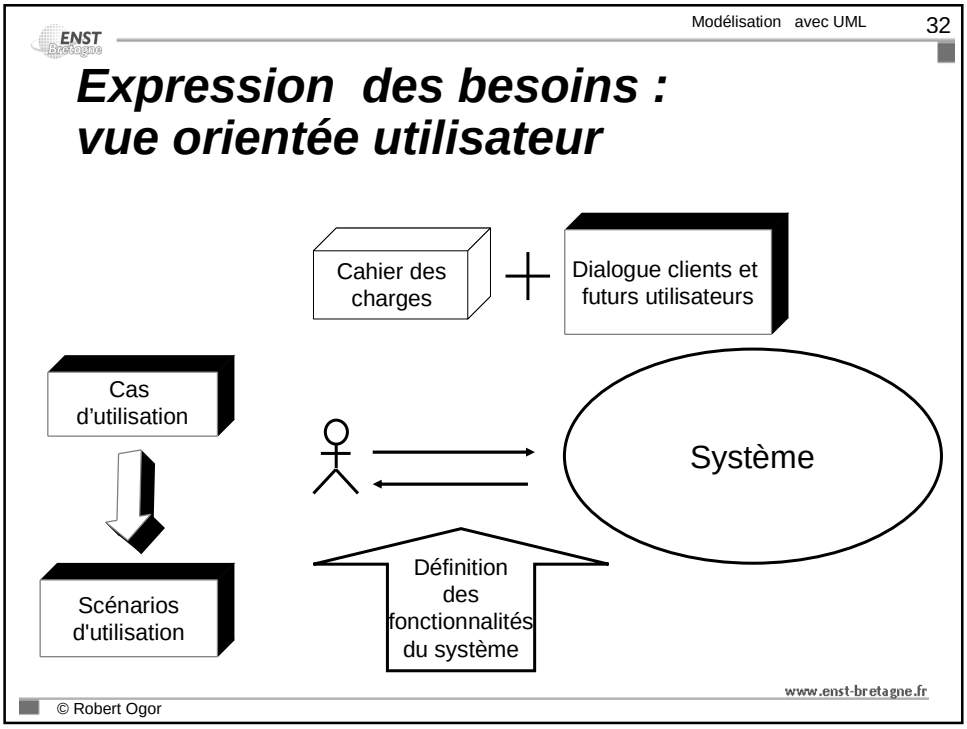
- Contraintes de concurrence
- Contraintes de temps de réponse
- Contraintes de distribution
- Contraintes de performance
- Contraintes de répartition

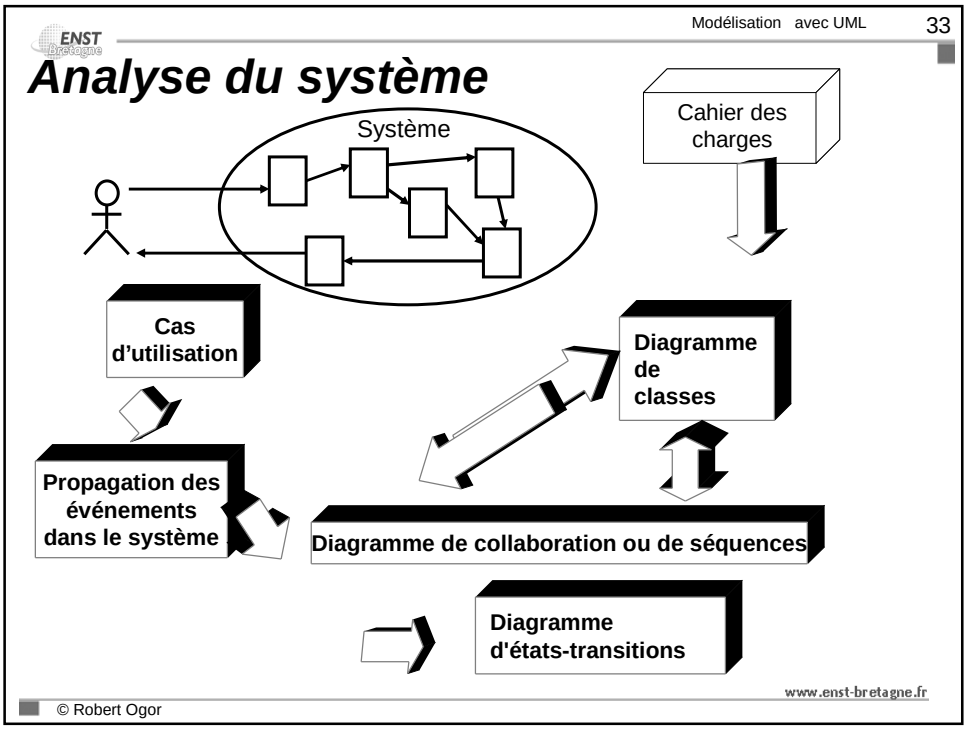
Dépendances entre phases de modélisation



L'analyse

- Le but de l'analyse est de traduire dans un langage proche de celui des informaticiens les modèles exprimés dans l'expression des besoins.
- Cependant pour rester compréhensible par les clients ou utilisateurs, il n'intervient que des entités du domaine (métier) considéré.
- Elle sert d'interface, avec l'expression des besoins, aux dialogues et aux contrats avec les clients et les utilisateurs.
- L'analyse doit servir de support pour la conception, l'implémentation et la maintenance.


Modélisation avec UML
32
Expression des besoins : vue orientée utilisateur
Cahier des charges
Dialogue clients et futurs utilisateurs
Cas d'utilisation
Scénarios d'utilisation
Système
Définition des fonctionnalités du système
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML
33
Analyse du système
Système
Cahier des charges
Cas d'utilisation
Diagramme de classes
Propagation des événements dans le système
Diagramme de collaboration ou de séquences
Diagramme d'états-transitions
www.enst-bretagne.fr
© Robert Ogor

La notation graphique

BUT :

- Modéliser les objets, les relations entre objets, les interactions avec le système.
- Servir de support de communication entre l'analyste, le client, et les utilisateurs.

Cahier des charges : gestion de bibliothèque

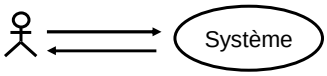
- Un gérant de bibliothèque désire automatiser la gestion des prêts.
- Il commande un logiciel permettant aux utilisateurs de connaître les livres présents, d'en réserver jusqu'à 2. L'adhérent peut connaître la liste des livres qu'il a empruntés ou réservés.
- L'adhérent possède un mot de passe qui lui est donné à son inscription.
- L'emprunt est toujours réalisé par les employés qui travaillent à la bibliothèque. Après avoir identifié l'emprunteur, ils savent si le prêt est possible (nombre max de prêts = 5), et s'il a la priorité (il est celui qui a réservé le livre).
- Ce sont les employés qui mettent en bibliothèque les livres rendus et les nouveaux livres. Il leur est possible de connaître l'ensemble des prêts réalisés dans la bibliothèque.

2) Le diagramme des Use Cases ou des cas d'utilisation

Ce que doit faire le système sans spécifier comment il le fait

But des Use Cases

Les cas d'utilisation représentent les fonctionnalités que le système doit savoir faire.



Chaque cas d'utilisation peut être complèté par un ensemble d'interactions successives d'une entité en dehors du système (l'utilisateur) avec le système lui même.

Les Uses Cases permettent :

- De connaître le comportement du système sans spécifier comment ce comportement sera réalisé.
- De définir les limites précises du système
- Au développeur de bien comprendre l'attente des utilisateurs et les experts du domaine.

De plus les Use Cases sont :

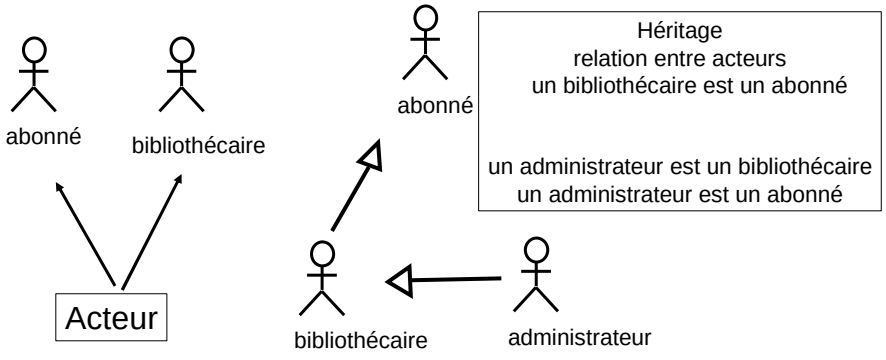
- Des instruments de validation et de test du système en cours et en fin de construction.

Modèle des cas d'utilisations

- Un diagramme de cas d'utilisation définit :
  - le système
  - les acteurs
  - les cas d'utilisations
  - les liens entre acteurs et cas d'utilisations
- Un modèle de cas d'utilisation se définit par :
  - des diagrammes de cas d'utilisation
  - une description textuelle des scénarios d'utilisation
  - une description de ces scénarios par :
    - les diagrammes de séquences
    - les diagrammes de collaboration

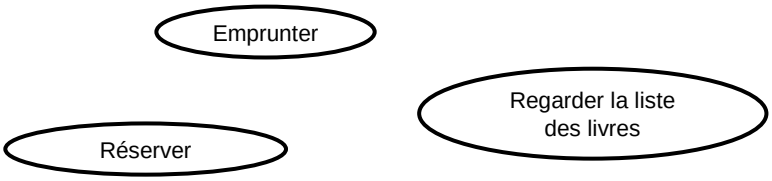
Les Acteurs

- Un acteur représente une personne ou un périphérique qui joue un rôle (interagit) avec le système.
- Relation entre acteurs : généralisation (héritage)



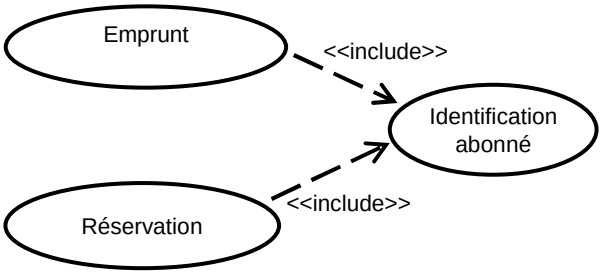
Les cas d'utilisation (use-case)

- Un cas d'utilisation est un moyen de représenter les différentes possibilités d'utiliser un système.
- Il exprime toujours une suite d'interactions entre un acteur et l'application.
- Il définit une fonctionnalité utilisable par un acteur.



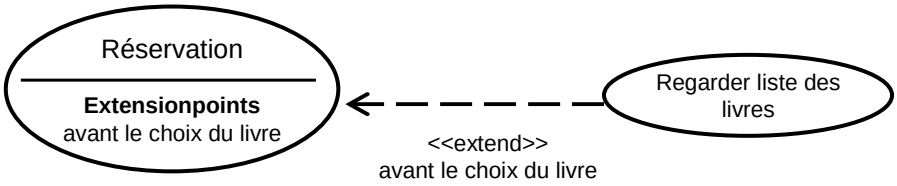
Organisation des Use Cases : include

- La relation "*include*" précise qu'un cas d'utilisation contient le comportement défini dans un autre cas d'utilisation.
- Cette relation permet de mettre en commun des comportements communs à plusieurs cas d'utilisation..



Organisation des Use Cases : extend

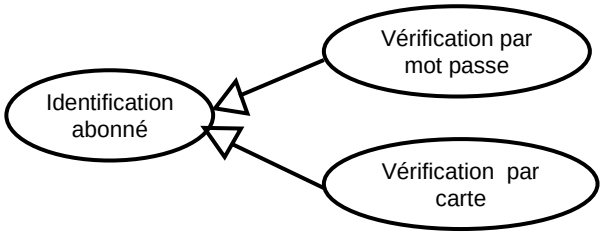
- La relation "*extend*" précise qu'un cas d'utilisation peut dans certains cas augmenter le comportement d'un autre cas d'utilisation.
- Une condition devra valider cette augmentation.
- Le point d'utilisation de cette augmentation peut être défini dans un "point d'extension".



- Dans cet exemple, le cas d'utilisation "Regarder la liste des livres" augmente le cas d'utilisation d'une réservation, avant le choix du livre, si l'utilisateur en fait la demande.

Organisation des Use Cases : généralisation

- Cette relation "*est un*" introduit la notion d'héritage.
- Les cas d'utilisation "Vérification par mot passe" et "Vérification par carte" sont des spécialisations du cas d'utilisation "Identification abonné".
- Autrement dit : si l'on dit que l'on fait une "Identification abonné", ce peut être une "Vérification par carte" ou une "Vérification par mot passe".

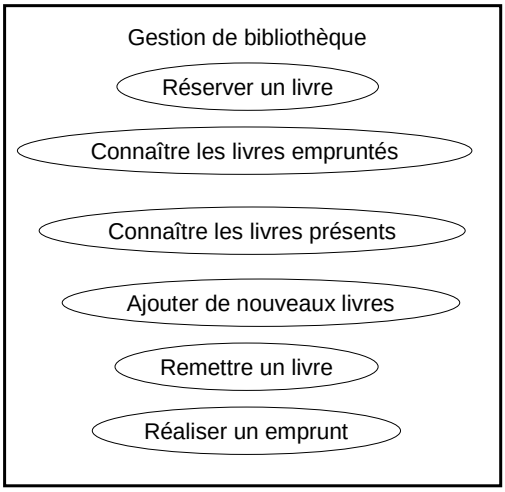


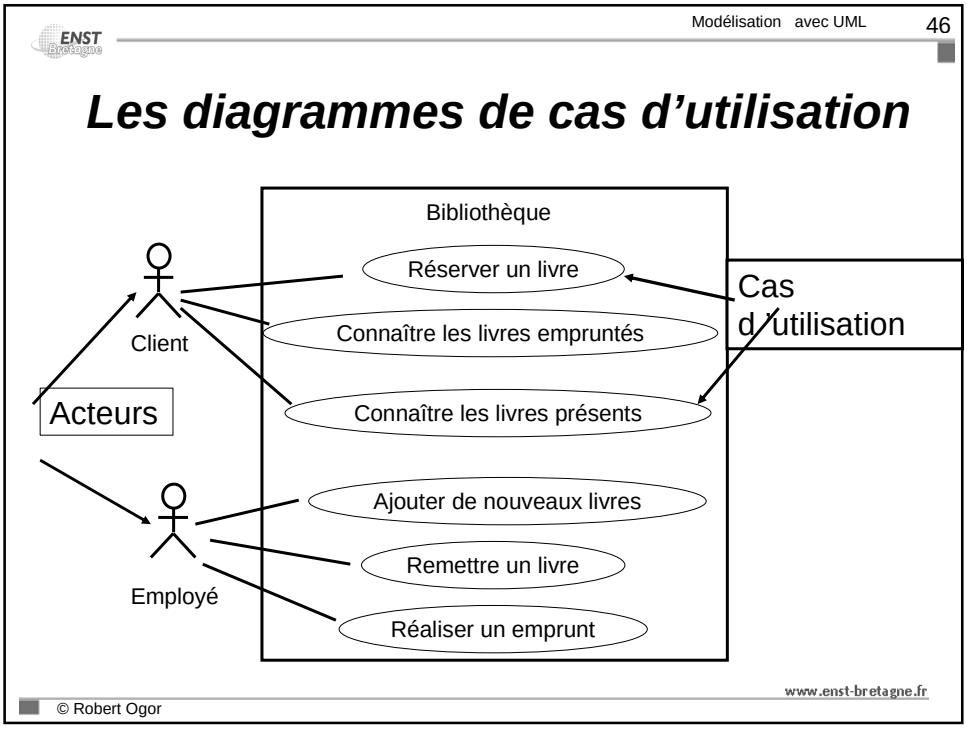
Modélisation d'un système: obtenir les cas d'utilisation

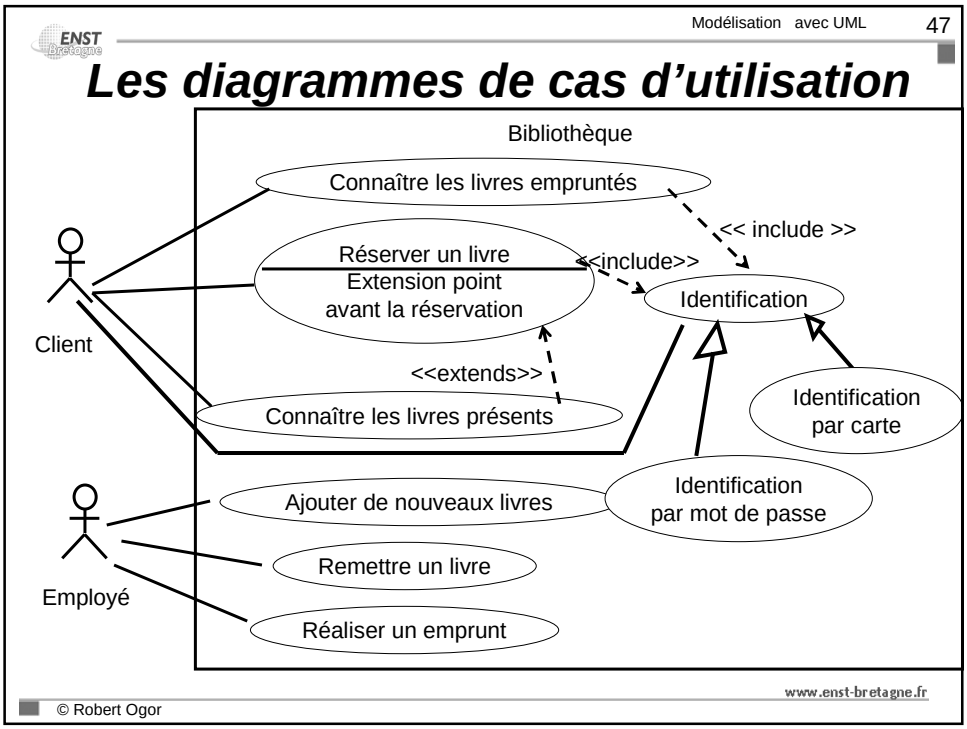
- Identifier les acteurs qui utilisent, qui gèrent, qui exécutent des fonctions spécifiques.
- Organiser les acteurs par relation d'héritage.
- Pour chaque acteur, rechercher les cas d'utilisation avec le système. En particulier, ceux qui modifient l'état du système ou qui attendent une réponse du système.
- Ne pas oublier les variantes d'interactions (cas d'erreur, cas interdits).
- Organiser ces interactions par héritage, par utilisation et par extension.

Le système

- Le système définit l'application informatique, il ne contient donc pas les acteurs, mais les cas d'utilisation et leur associations




Modélisation avec UML
46
Les diagrammes de cas d'utilisation
Bibliothèque
Réserver un livre
Connaître les livres empruntés
Connaître les livres présents
Ajouter de nouveaux livres
Remettre un livre
Réaliser un emprunt
Client
Employé
Acteurs
Cas d'utilisation
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML
47
Les diagrammes de cas d'utilisation
Bibliothèque
Connaître les livres empruntés
<< include >>
Réserver un livre
Extension point
avant la réservation
<<include>>
Identification
Client
<<extends>>
Identification
par carte
Connaître les livres présents
Identification
par mot de passe
Ajouter de nouveaux livres
Remettre un livre
Employé
Réaliser un emprunt
www.enst-bretagne.fr
© Robert Ogor

Scénarios d'un cas d'utilisation

- La description d'un cas d'utilisation se fait par des scénarios qui définissent la suite logique des interactions qui constituent ce cas.
- On peut définir des scénarios simples ou des scénarios plus détaillés faisant intervenir les variantes, les cas d'erreurs, etc.
- Cette description se fait de manière simple, par un texte compréhensible par les personnes du domaine de l'application.
- Elle précise ce que fait l'acteur et ce que fait le système
- La description détaillée pourra préciser les contraintes de l'acteur et celles du système.

Scénarios d'un cas d'utilisation

Réservation d'un livre

description simplifiée

Le client se présente devant un terminal:

- (1) Le système affiche un message d 'accueil.
- (2) Le client choisit l'opération réservation parmi les différentes opérations proposées.
- (3) Le système lui demande de s'authentifier.
- (4) Le client donne son identification (nom, mot de passe).
- (5) Le système lui demande de choisir un livre.
- (6) Le client précise le livre qu'il désire.
- (7) Le système lui précise si un exemplaire du livre lui est réservé.

Scénarios d'un cas d'utilisation

Réservation d'un livre

description détaillée

Pré-conditions: Le client doit être inscrit à la bibliothèque
Le client ne doit pas avoir atteint le nombre maximum de réservation
Un exemplaire du livre doit être enregistré

Post-conditions: (Si l'opération s'est bien déroulée)
Le client a une réservation supplémentaire
Le nombre d'exemplaires disponibles du livre est décrémenté de 1

Scénarios d'un cas d'utilisation

Réservation d'un livre

Cas normal

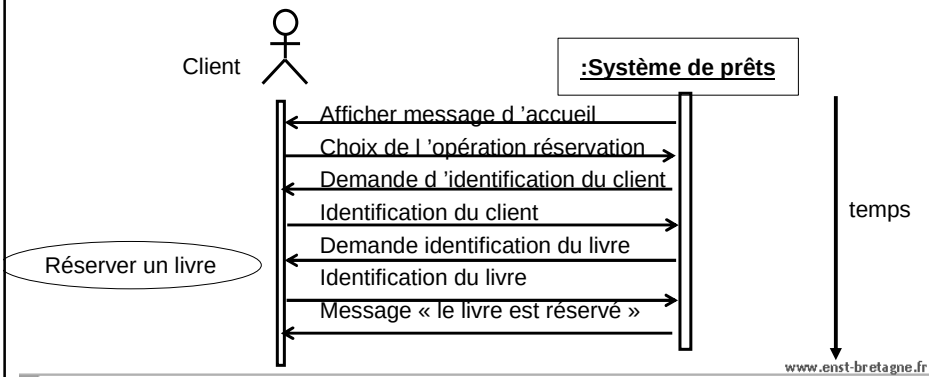
description détaillée

1. (1) Le système affiche un message d 'accueil sur le terminal avec un choix d'opérations
2. (2) Le client choisit l'opération réservation parmi les différentes opérations proposées.
3. (3) Le système lui demande de s'authentifier.
4. (4) Le client donne son identification (nom, mot de passe).
5. (5) Le système demande le titre du livre en donnant la possibilité de choisir dans une liste.
6. (6) Le client précise le livre qu'il désire.
7. (7) Le système lui précise que un exemplaire du livre lui est réservé.

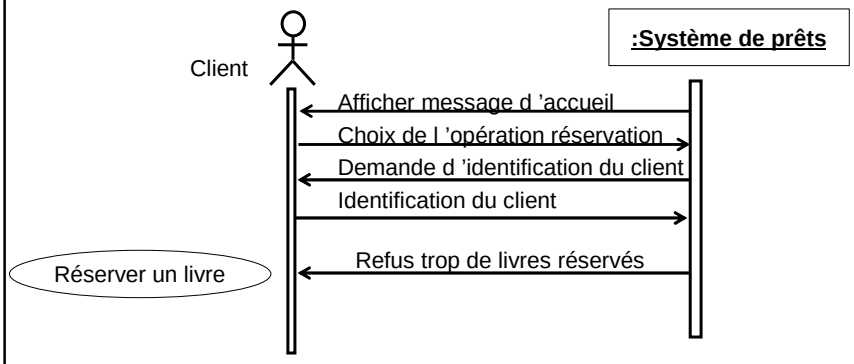
variante 1	en (6) le client demande à connaître les livres présents
variante 2	en (4) le client n'est pas reconnu, dans ce cas, tant qu'il n'est pas reconnu, on lui redemande de s'authentifier
variante 3	en (4) le client est reconnu, mais le mot de passe est incorrect, il peut avoir 5 essais, par la suite le client sera interdit pendant 1 heure.
variante 4	en (4) le client n'a plus le droit de réserver
variante 5	en (7) le livre n'est pas libre

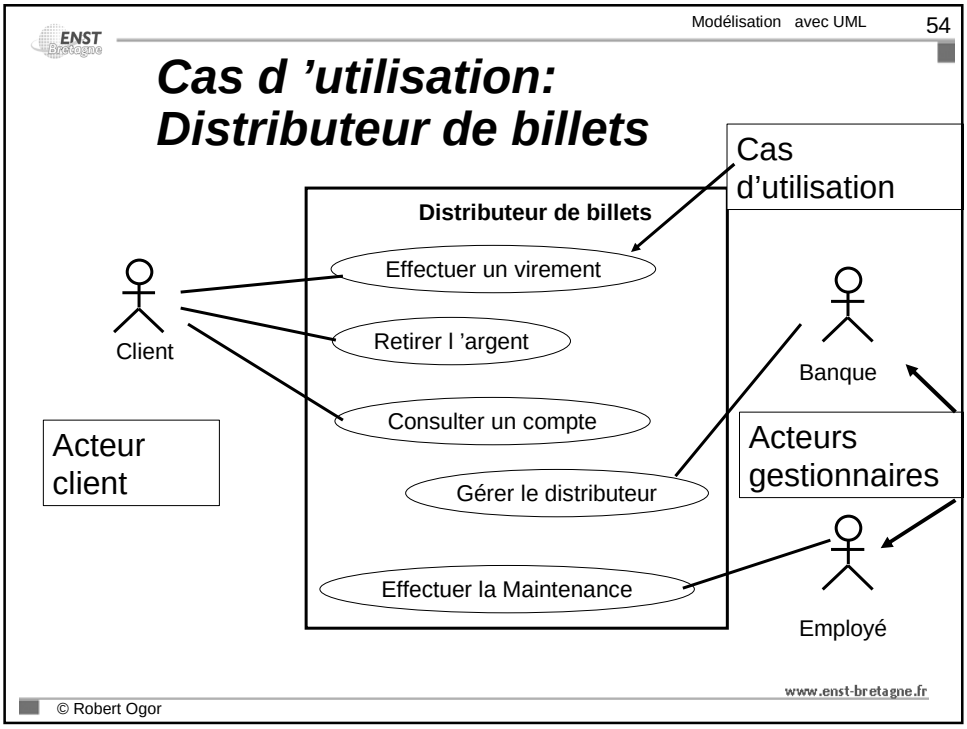
Scénario par diagramme de séquences

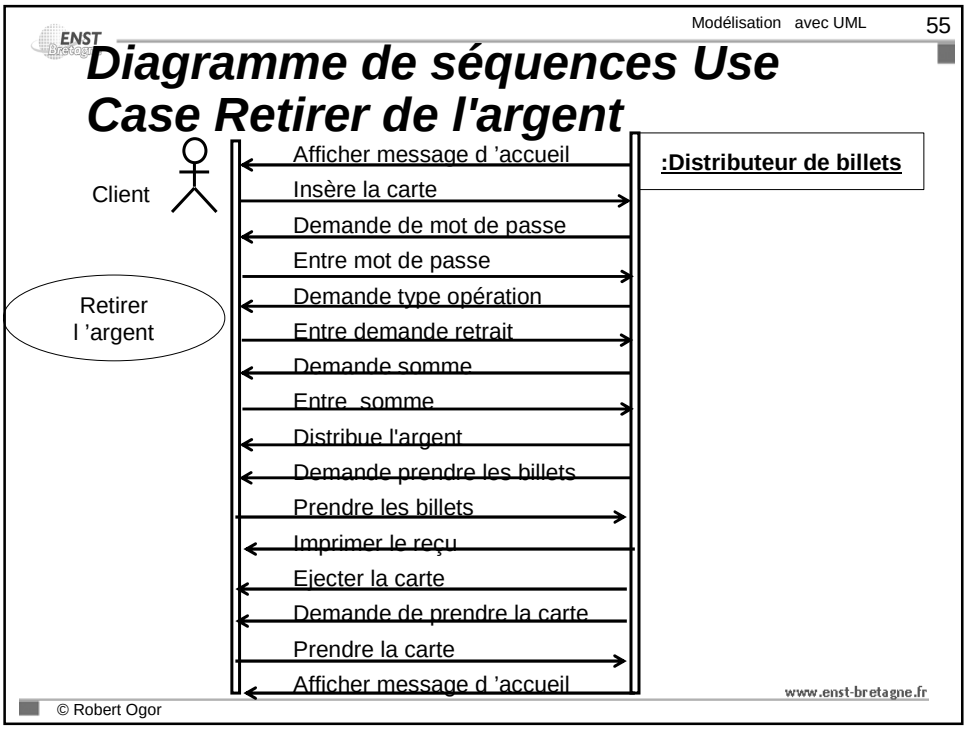
- Suite aux descriptions textuelles, le scénario peut être représenté en utilisant un diagramme de séquences.
  Le diagramme de séquences permet :
  - de visualiser l'aspect temporel des interactions
  - de connaître le sens des interactions (acteur vers système ou contraire)



Variation du scénario




Modélisation avec UML 54
Cas d ’utilisation: Distributeur de billets
Cas d’utilisation
Distributeur de billets
Effectuer un virement
Retirer l ’argent
Consulter un compte
Gérer le distributeur
Effectuer la Maintenance
Client
Banque
Employé
Acteur client
Acteurs gestionnaires
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML 55
Diagramme de séquences Use Case Retirer de l'argent
Client
Retirer l ’argent
:Distributeur de billets
Afficher message d ’accueil
Insère la carte
Demande de mot de passe
Entre mot de passe
Demande type opération
Entre demande retrait
Demande somme
Entre somme
Distribue l'argent
Demande prendre les billets
Prendre les billets
Imprimer le reçu
Ejecter la carte
Demande de prendre la carte
Prendre la carte
Afficher message d ’accueil
www.enst-bretagne.fr
© Robert Ogor

3) Concepts objets et diagrammes de classes

La classe

CLASSE

Personne

nom
age
adresse
mot de passe
nbre livres empruntés

changerAdresse()
donnerAge()
donnerAdresse()
vérifierMotDePasse()

Caractéristiques
attributs
données membres
informations
propriétés

Famille d'objets ayant
mêmes caractéristiques
et même comportement

Comportement
opérations
méthodes
fonctions
procédures
messages
services

CLASSE

Personne

La classe et ses membres

Personne

nom : String
âge : Integer
adresse : String
mot de passe : String
nbre livres empruntés : Integer

0<= nbre livres empruntés <=5

<<constructeur>>
Personne(nom:String, âge : Integer
adresse:String, motDePasse:String)
<<caractéristiques>>
changerAdresse (String)
<<authentification>>
vérifierMotDePasse(String) : Boolean
changerMotDePasse(old:String,
new:String)

www.enst-bretagne.fr

La classe et l'objet

instance de

un objet

CLASSE

:Personne

Personne

nom
age
adresse
mot de passe
nbre livres empruntés

changerAdresse()
obtenirAge()
obtenirAdresse()
vérifierMotDePasse()

Attributs

Comportement

nom = Alain Dupont
age = 45
adresse = France
mot de passe = 4RSA67
nbre livres empruntés = 4

: Personne

André Roué
25
France
6FT34
0

:Personne

www.enst-bretagne.fr

Protection des attributs et des opérations : principe de l'encapsulation

- Peut-on accéder à tous les attributs ou à toutes les méthodes d'un objet ? NON
  - La classe définit ce qui est accessible.

    C'est le principe de l'encapsulation.

    Un objet complexe ne peut être utilisé qu'au travers de ce qui est accessible.

Exemple :

- On ne peut utiliser une *voiture* qu'à travers son *volant*, son *frein*, son *accélérateur*, etc.
- L'accès au *carburateur* est impossible sauf par les méthodes qui le font de manière cohérente (méthode *accélérer* de l'*accélérateur*).

Protection des attributs et des opérations : usage et notation

- Les attributs sont en général inaccessibles (secret).
  - Ils sont alors qualifiés de :
    - "***protected***" (notation UML : **#**)
    - ou "***private***" (notation UML: **-**)
  - Leur lecture ou modification n'est possible qu'au travers de certaines opérations (accesseurs : *changerAdresse()*, *obtenirAge()*, etc.)
- Les opérations sont en général accessibles (publiques) :
  - Elles sont alors qualifiées de :
    - "***public***" (notation UML: **+**)
- Certaines opérations peuvent cependant être privées et certains attributs peuvent être publics (non souhaitable / principe d'encapsulation)

Protection des attributs et des opérations : notation UML

CLASSE

Personne

- nom
- âge
- adresse

changerAdresse()
obtenirRue()
+ obtenirAge()

+ : attribut public
# : attribut protected
- : attribut private

+ : opération public
# : opération protected
- : opération private

Attributs et opérations de classe

Personne

- nom
- age
- adresse
- mot de passe
- nbreLivresEmpruntés
- nbreLivresEmpruntables = 5

+ changerAdresse()
+ obtenirAge()
+ obtenirAdresse()
+ vérifierMotDePasse()
+ getNbLivresEmpruntables()

Le nombre de livres empruntables n'est pas une caractéristique d' Alain Dupont (objet). C'est une caractéristique valable pour l'ensemble des personnes (la classe).

une_personne:Personne

nom = Alain Dupont
age = 45
adresse = France
mot de passe = 4RSA67
nbreLivresEmpruntés= 4

La méthode getNbLivresEmpruntables() utilise la valeur nbreLivresEmpruntables connue par la classe. Cette méthode peut être appliquée directement à la classe Personne et bien sûr aussi aux objets instances de Personne.

Attributs et opérations de classe Notation UML

CLASSE

Personne

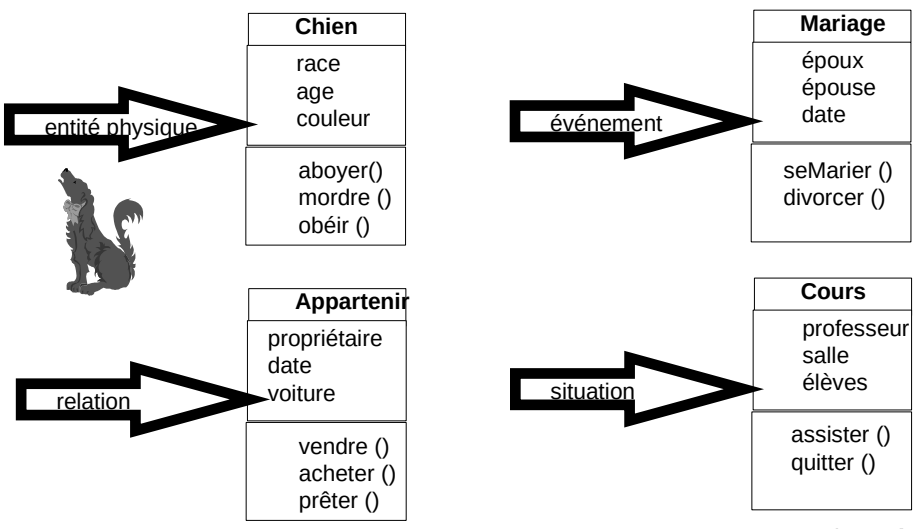
- nom
- âge
- adresse
- **nombrePersonne**

changerAdresse()
obtenirRue()
+ obtenirAge()
+**obtenirNombrePersonne()**

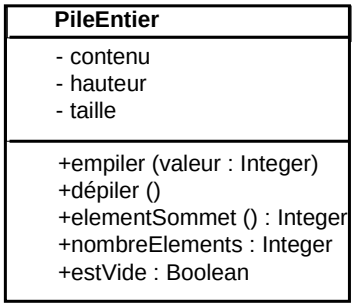
+ : attribut public
# : attribut protected
- : attribut private
__: attribut de classe

+ : opération public
# : opération protected
- : opération private
__: opération de classe

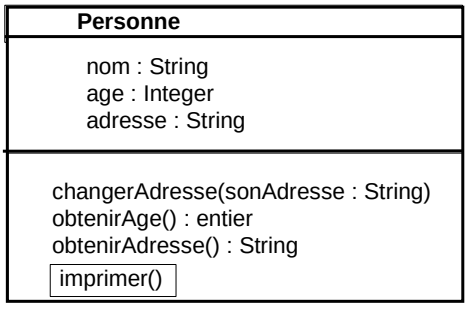
La réification

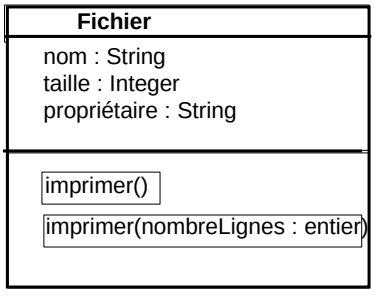


Modèle du type abstrait Pile

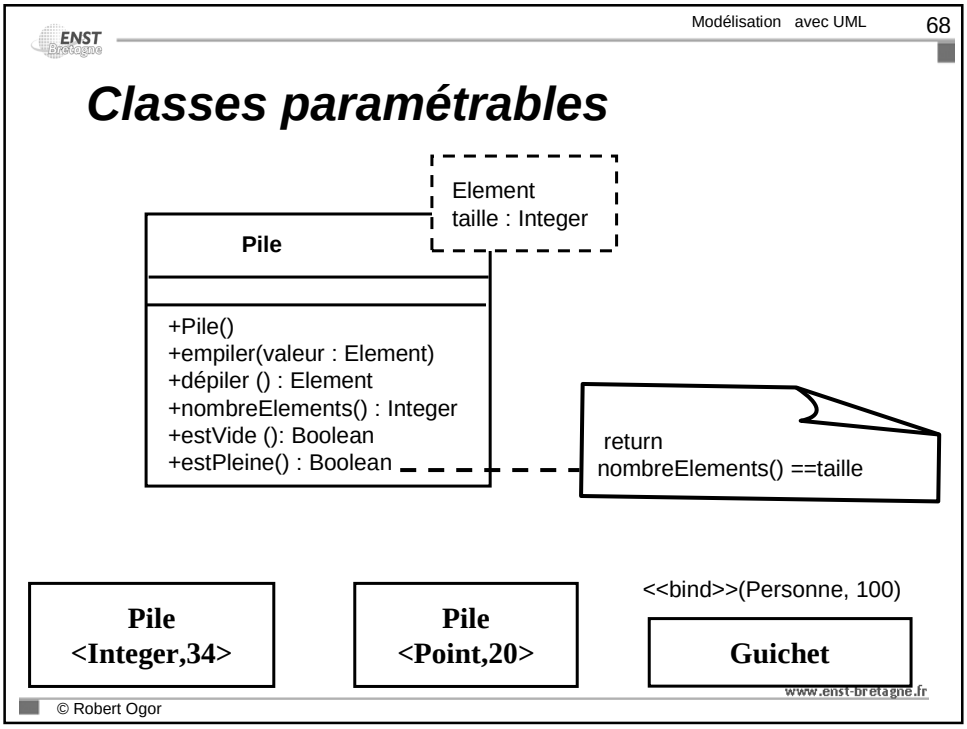


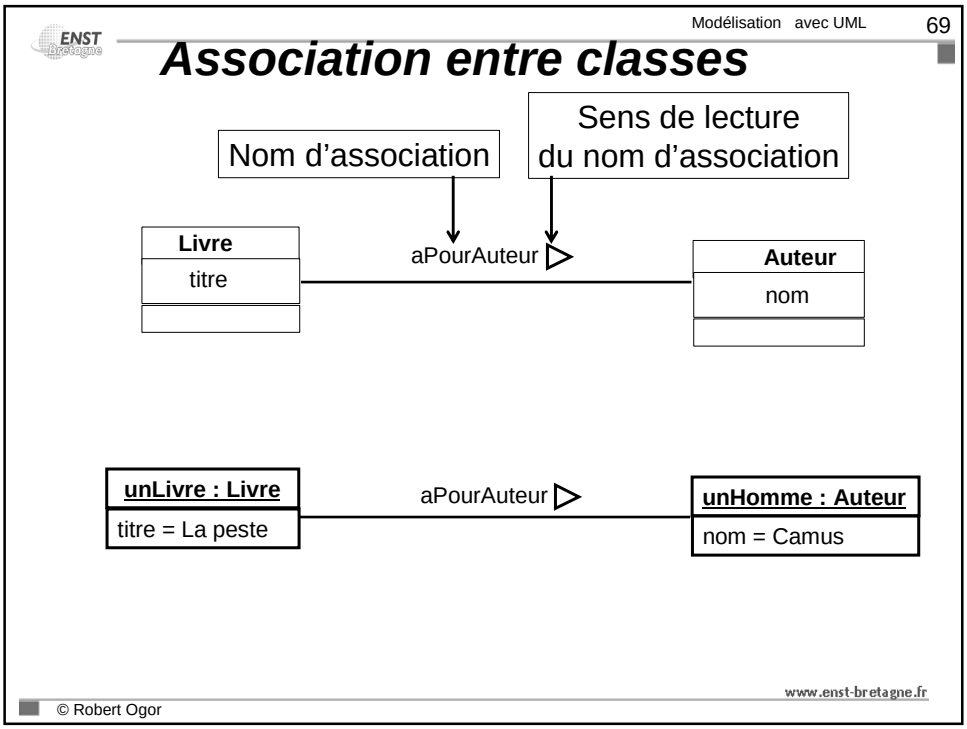
Surcharge et polymorphisme



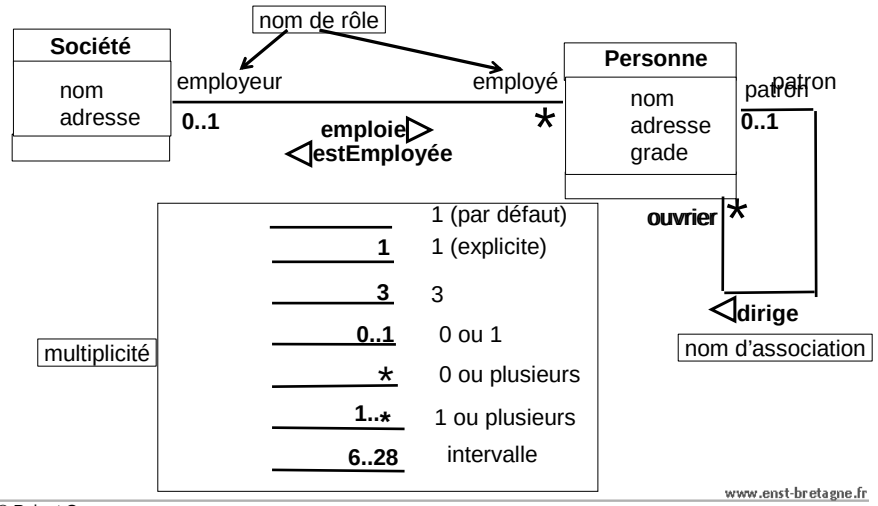


notions : polymorphisme
surcharge
signature

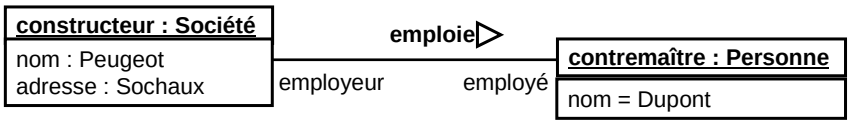

Modélisation avec UML 68
Classes paramétrables
Element
taille : Integer
Pile
+Pile()
+empiler(valeur : Element)
+dépiler () : Element
+nombreElements() : Integer
+estVide (): Boolean
+estPleine() : Boolean
return
nombreElements() ==taille
Pile
<Integer,34>
Pile
<Point,20>
<<bind>>(Personne, 100)
Guichet
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML 69
Association entre classes
Nom d'association
Sens de lecture
du nom d'association
Livre
titre
aPourAuteur
Auteur
nom
unLivre : Livre
titre = La peste
aPourAuteur
unHomme : Auteur
nom = Camus
www.enst-bretagne.fr
© Robert Ogor

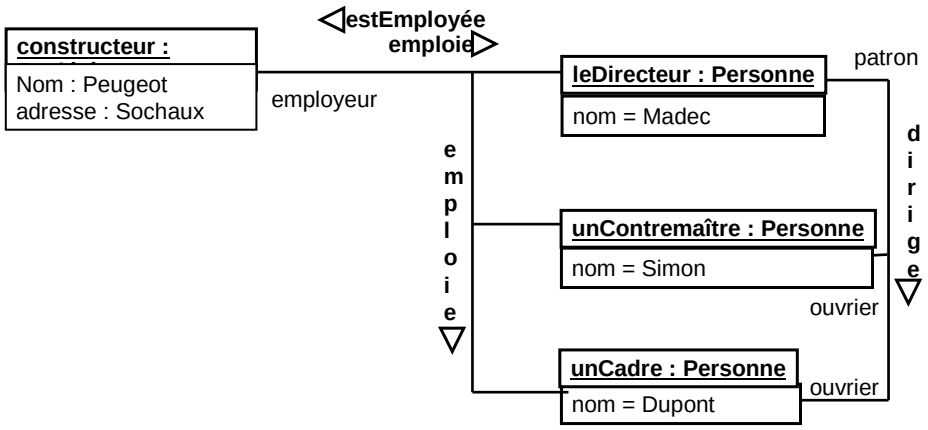
Multiplicité, nom d'association, nom de rôle



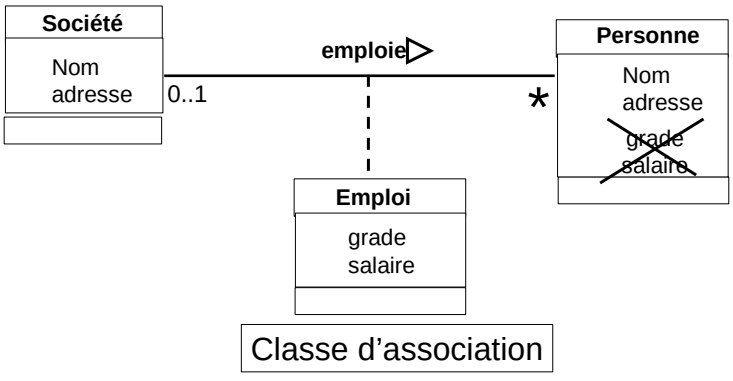
Lien entre objets



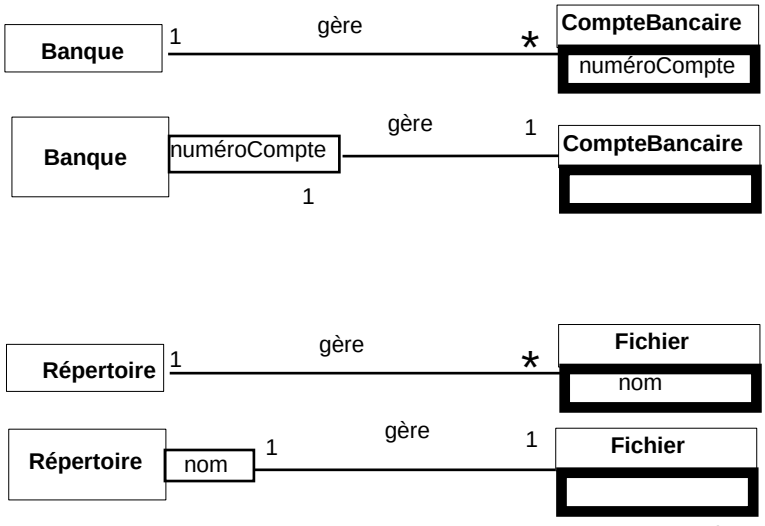
Lien entre objets



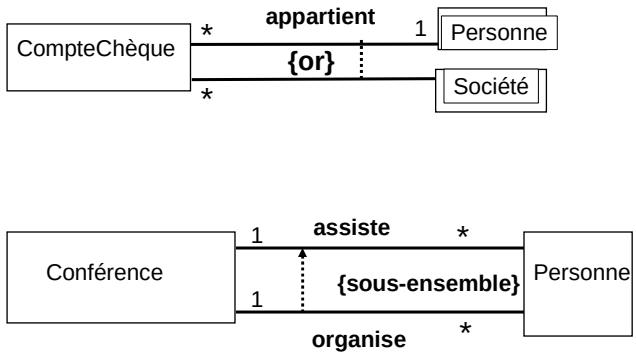
Classe d'association



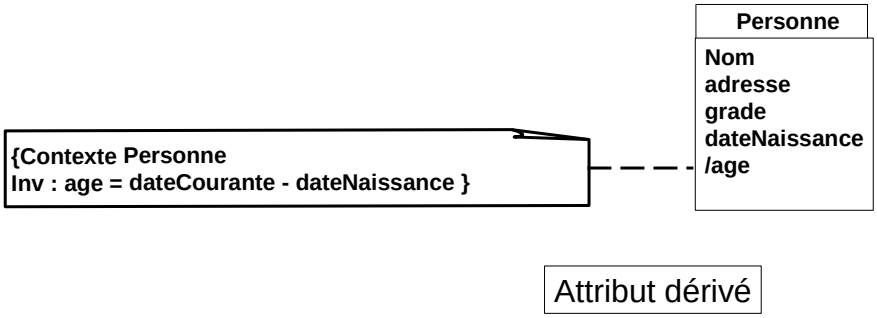
Qualificateurs : restriction de la cardinalité



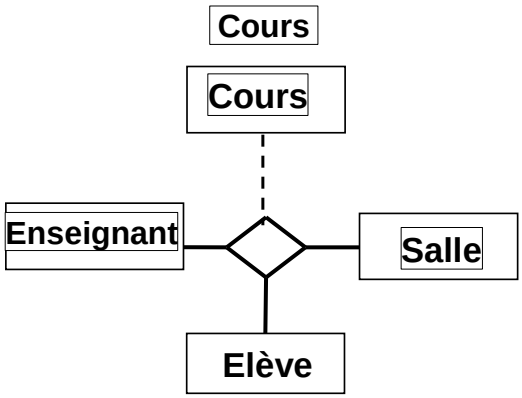
Contraintes sur les associations



Attribut dérivé



Les associations ternaires

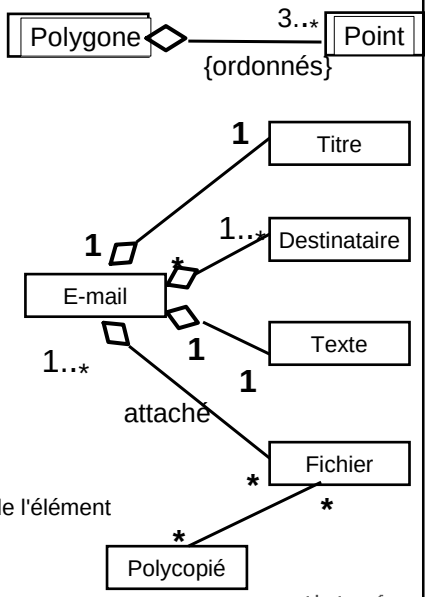


Agrégation

C'est une association qui exprime un **couplage fort** lié à une **relation de subordination**, elle est **asymétrique** du type **ensemble/élément**.

Règles permettant de choisir une agrégation :
- •Est ce une partie de?
- •Les opérations appliquées à l'ensemble sont elles appliquées à l'élément?
- •Les changements d'états sont-ils liés ?

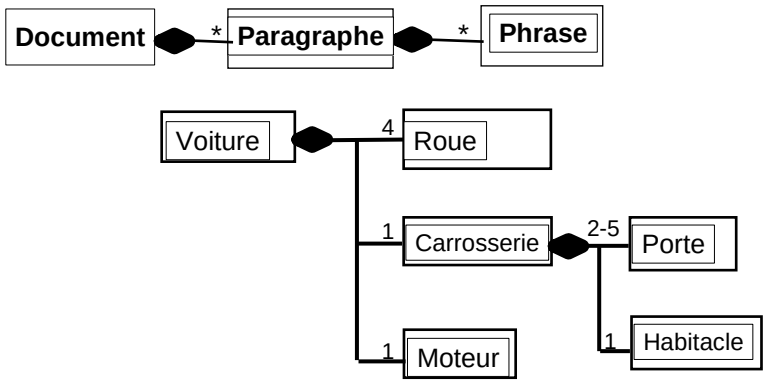
Mais
- un élément agrégé peut être lié à d'autres classes
- la suppression de l'ensemble n'entraîne pas celle de l'élément



Composition : agrégation forte

La composition est une **agrégation forte** qui **lie les cycles de vie** entre le composé (ensemble) et les composants (élements).

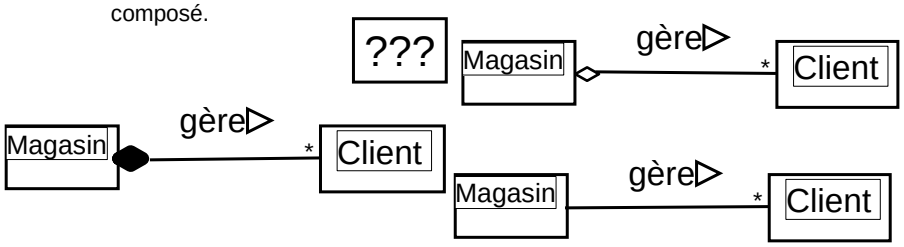
Le choix entre composition et agrégation peut être laissé à la phase de conception.



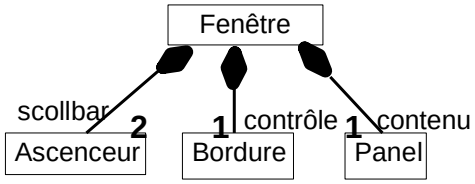
Association, agrégation ou composition ?

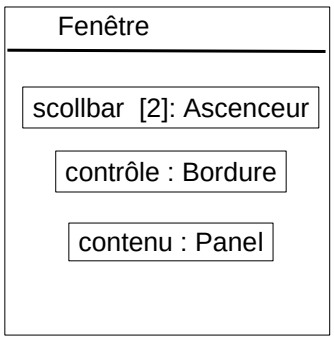
1. Règles obligatoires pour l'agrégation :
   - Est ce une partie de?
   - Les opérations appliquées au composé sont elles appliquées au composant?
   - Les changements d'états sont-ils liés ?

2. Règles obligatoires pour la composition :
   - La suppression du composé entraîne t-elle la suppression des composants ?
   - Les attributs du composé sont-ils utilisés dans les composants ?
   - Les composants sont des instances du composé ?
   - Un composant ne peut pas être en relation avec d'autres classes externes au composé.



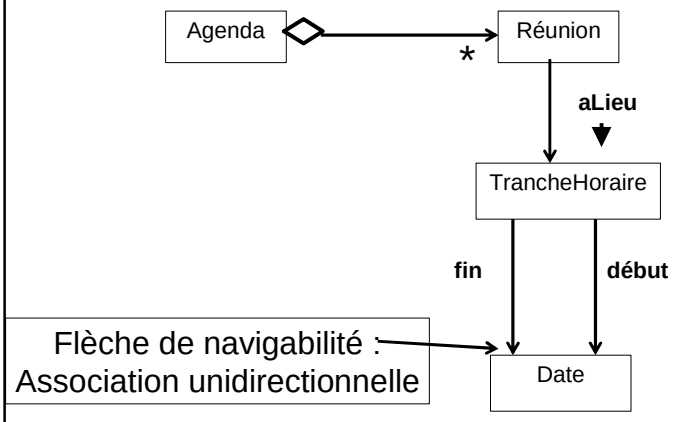
Composition



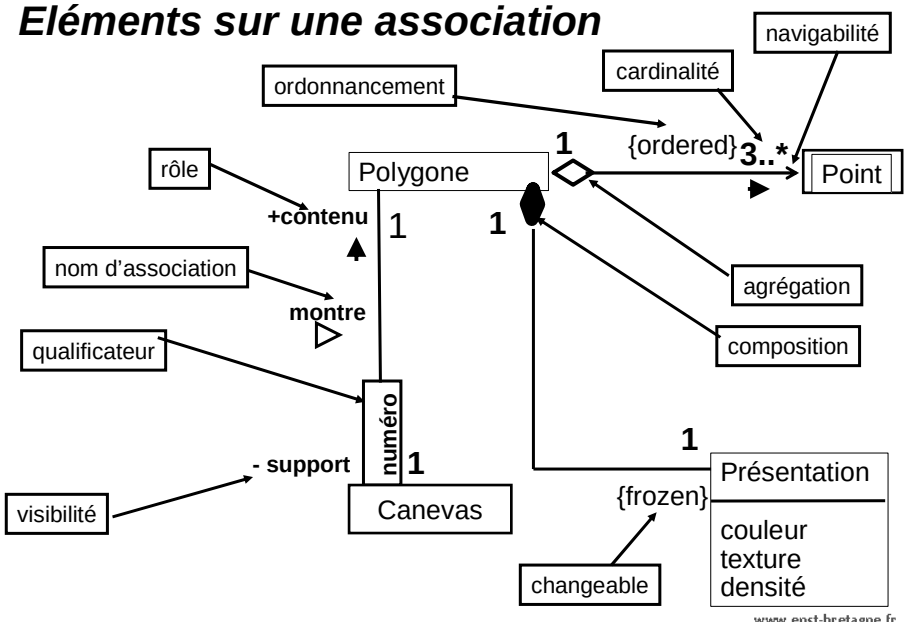


Navigabilité

Par défaut une association est bidirectionnelle.
Il est possible de réduire la portée en la rendant unidirectionnelle.
En général, ce choix se fait dans la phase de conception.



Eléments sur une association



3) Concepts objets et diagrammes de classes

→ Héritage

Héritage : buts et principes 1/3 Nouvelles classes dérivées de classes existantes

BUT

Permettre une réutilisation optimale des classes déjà écrites, utilisées et validées

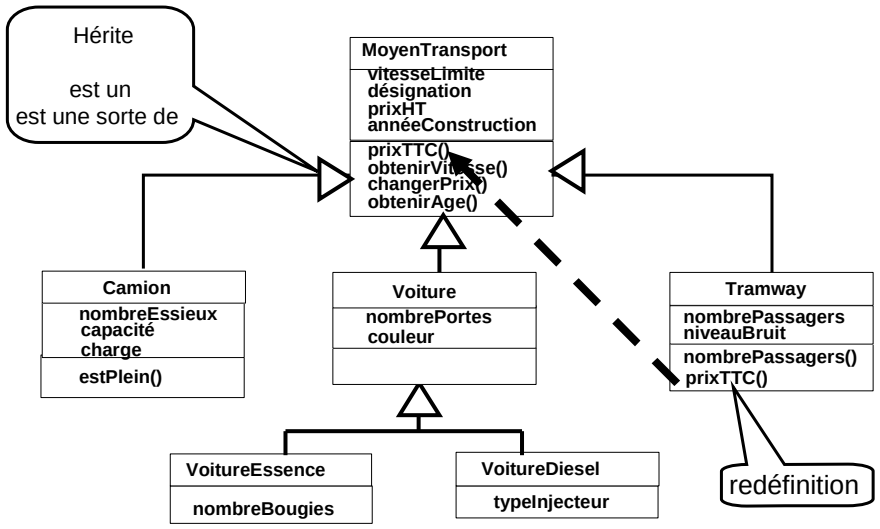
- réutilisation de la structure des données héritées
- réutilisation du code des services hérités

PRINCIPE

Ne pas modifier les classes déjà écrites cela modifierait l'utilisation qui en est faite.

- ne pas hésiter à créer des classes, extensions d'autres déjà validées

Héritage : adaptation



Héritage : buts et principes 2/3
Ajout d'une classe de base (analyse)

BUT 2

Permettre une factorisation des caractéristiques et des comportements communs à plusieurs classes

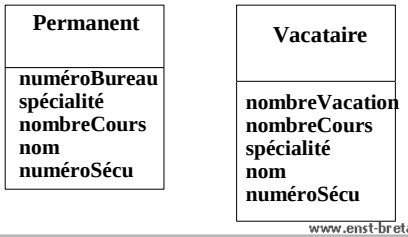
- mise en commun des structure des données
- mise en commun du code des services

PRINCIPE

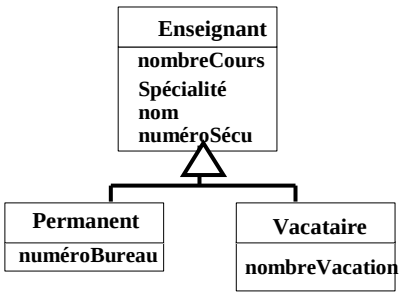
Lorsque plusieurs classes ont des caractéristiques et des comportements communs la création d'une classe ancêtre permet de regrouper ce qui est commun.

Cette classe ancêtre peut correspondre à une classe concrète ou à une classe abstraite

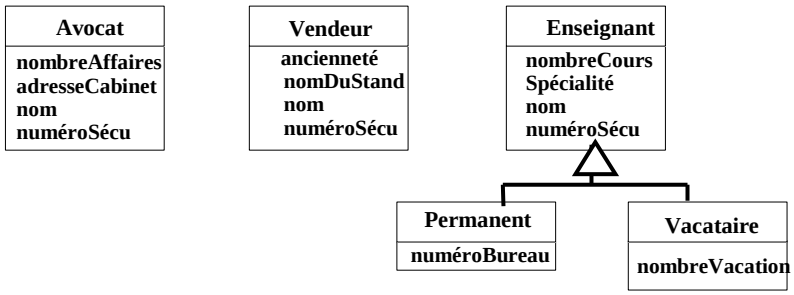
Héritage : généralisation
Factorisation des propriétés



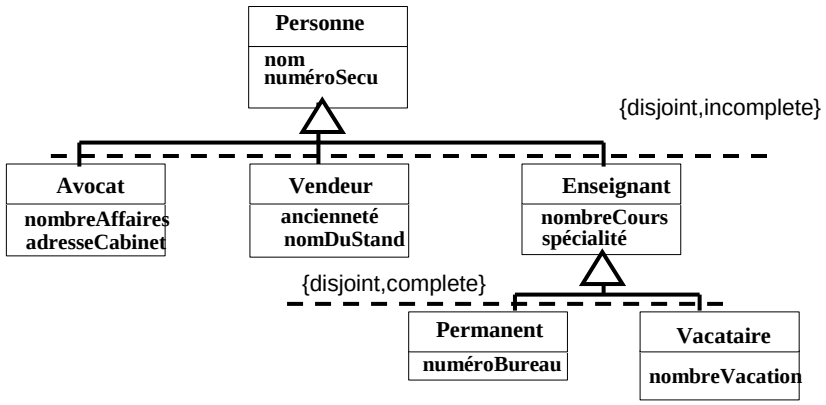
Héritage : généralisation
Factorisation des propriétés



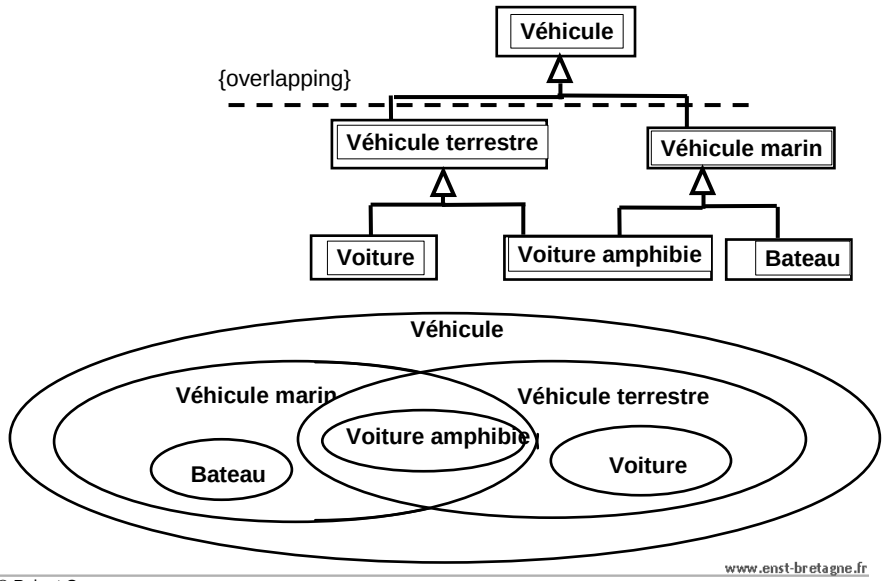
Héritage : généralisation Factorisation des propriétés



Héritage : généralisation Factorisation des propriétés



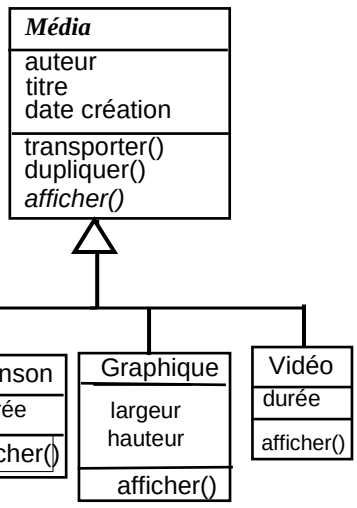
Héritage multiple et répété



Classe abstraite

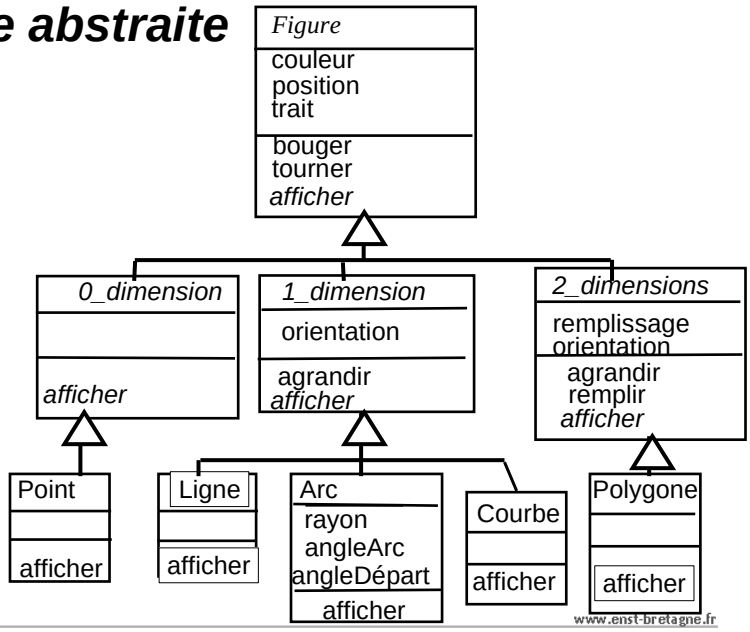
Notation UML :
nom de classe italique
ou stéréotype <<abstract>>

Un média peut être transporté, dupliqué, affiché.
Le transport et la duplication sont indépendants du type du média (copie de fichiers).
Par contre, tout média peut être affiché et ce n'est pas la même chose pour l'audio, la vidéo, le graphisme, le texte.
Un média ne peut pas définir comment s'afficher tant qu'il ne sait pas ce qu'il est.



Il n'y a pas d'instance de la classe média .
Un média n'existe qu'en tant que livre, chanson, graphique ou vidéo.

Classe abstraite



Héritage : buts et principes 3/3 Polymorphisme

BUT 3

Créer des sous-types (sous-classes). Une sous-classe sera du même type que la classe dont elle hérite (super-classe).

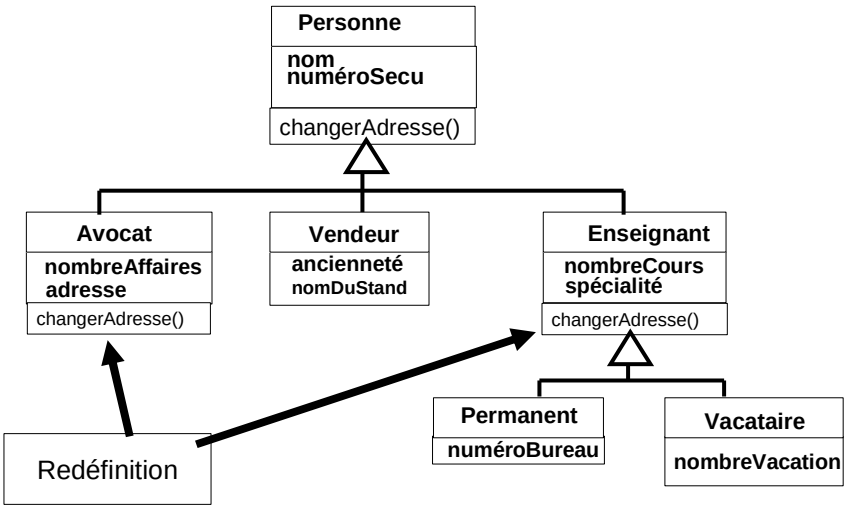
Ceci permet de mettre en œuvre le polymorphisme et la liaison dynamique

PRINCIPE

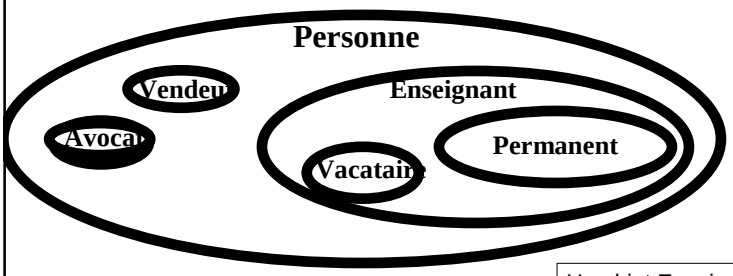
Un objet d'une classe donnée pourra toujours faire référence à des objets de ses sous classes (polymorphisme).

Une opération exécutée par un objet sera celle que connaît l'objet dont il fait référence (liaison dynamique).

Redéfinition et liaison dynamique



Héritage et sous typage



Un objet Personne peut désigner une instance des classes :
- Personne
- Avocat
- Vendeur
- Enseignant
- Vacataire
- Permanent

Conformante

Un objet Enseignant peut désigner une instance des classes :
- Enseignant
- Vacataire
- Permanent

Il ne peut pas désigner une instance des classes :
- Personne
- Avocat
- Vendeur

Conformante

- Première définition :
  - Se dit d'une classe plus spécialisée
- Exemple
  - Un Enseignant est **conformant** à une Personne
- Utilisation
  - Un objet conformant à un autre peut être utilisé à sa place sans pour autant déclencher d'erreur de type
  - étant plus spécialisé, il hérite de tous les services fournis par sa super-classe...

Exemple de conformante

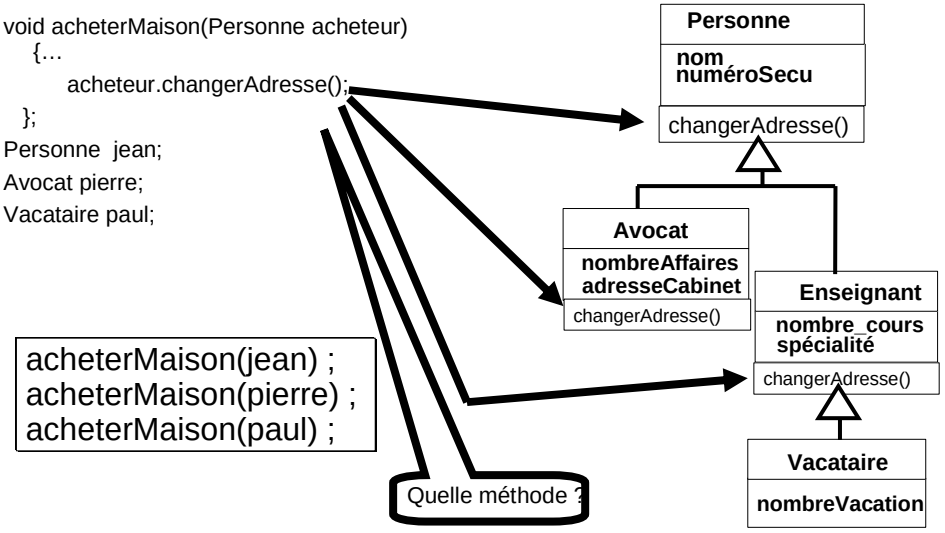
```
void acheterMaison(Personne acheteur)
  {…
       acheteur.changerAdresse();
  };

Personne  jean;
Avocat pierre;
Vacataire paul;
```

Le paramètre peut être abstrait ou concret

```
acheterMaison(jean) ;   // licite car jean est une Personne
acheterMaison(pierre) ; // licite car pierre est conformant à Personne
acheterMaison(paul) ;   // licite car paul est conformant à  Personne
```

Conformante et liaison dynamique

```
void acheterMaison(Personne acheteur)
   {…
       acheteur.changerAdresse();
  };
Personne  jean;
Avocat pierre;
Vacataire paul;
```

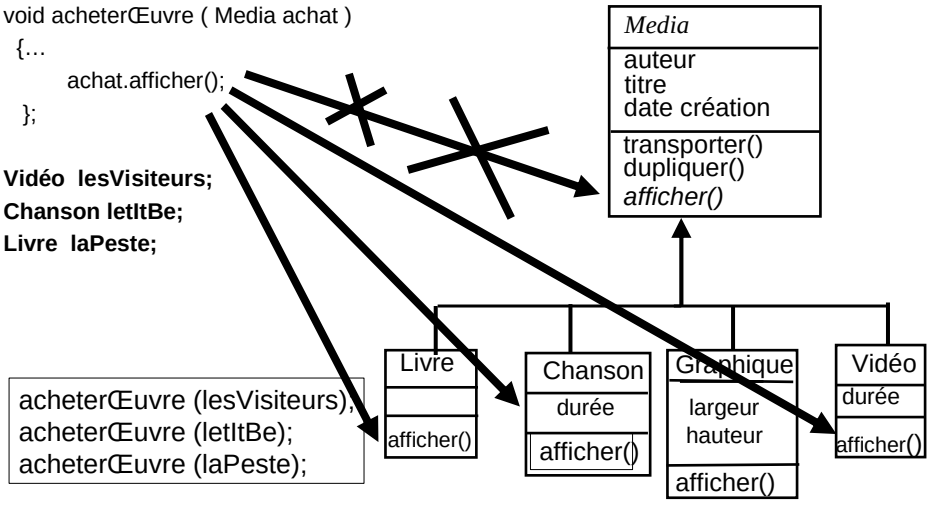



```
acheterMaison(jean) ;
acheterMaison(pierre) ;
acheterMaison(paul) ;
```

Quelle méthode ?

Liaison dynamique et classe abstraite

```
void acheterŒuvre ( Media achat )
 {…
       achat.afficher();
  };

Vidéo  lesVisiteurs;
Chanson letItBe;
Livre  laPeste;
```



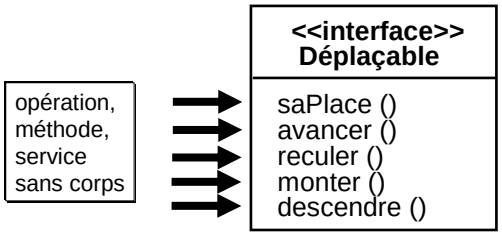

```
acheterŒuvre (lesVisiteurs);
acheterŒuvre (letItBe);
acheterŒuvre (laPeste);
```

Les Interfaces

- Une **interface** permet de décrire le **comportement** d'une entité (classe, paquetage ou composant), c'est à dire un savoir faire sous la forme d'une **liste d'opérations**.
- Une interface ne peut donner lieu à **aucune implémentation**.
- Une interface est équivalente à une classe abstraite sans attributs où toutes les méthodes sont abstraites.

- **Une classe** peut déclarer qu'elle **implémente une interface**. Elle doit alors implémenter toutes les opérations de cette interface. Elle peut ensuite être utilisée partout où ce comportement est exigé.

Interface

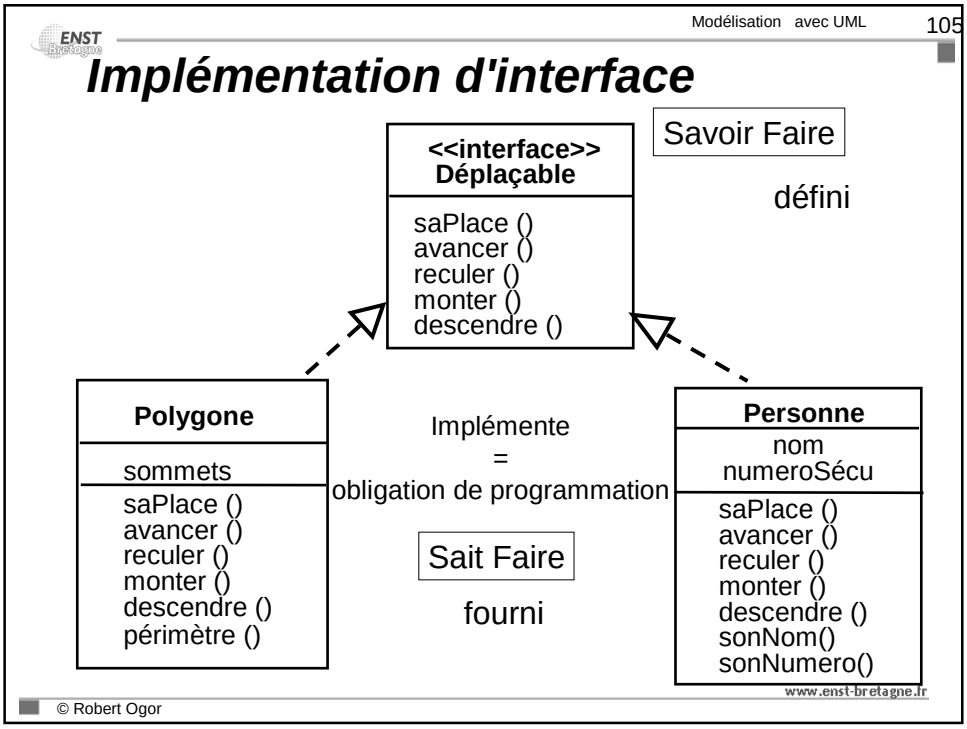


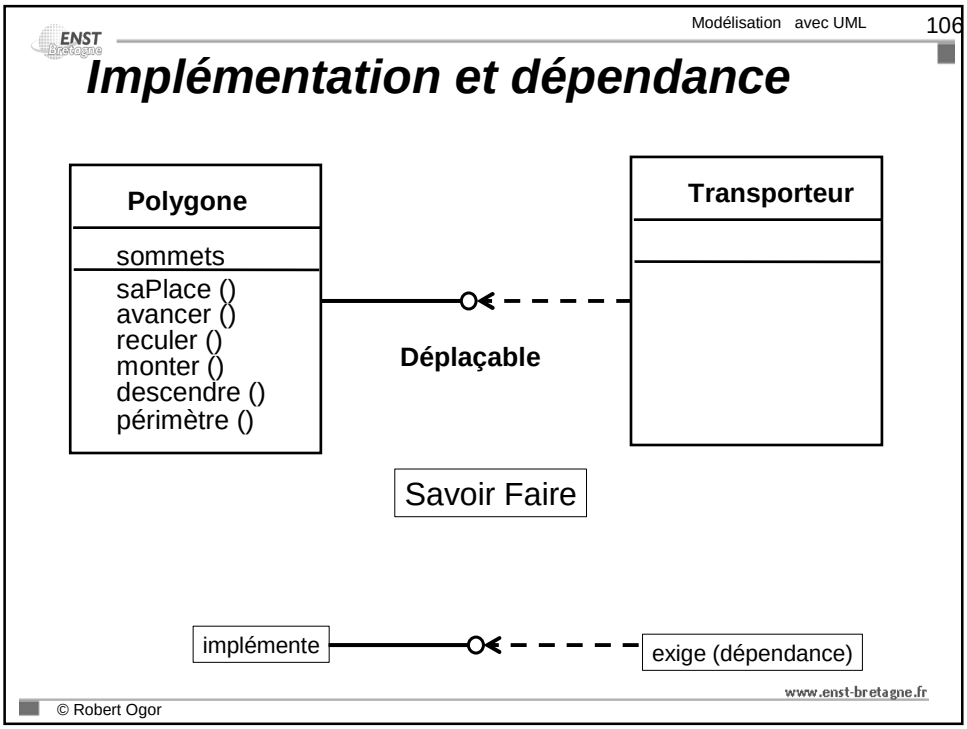
Une interface n'est PAS une classe
C'est une liste de services

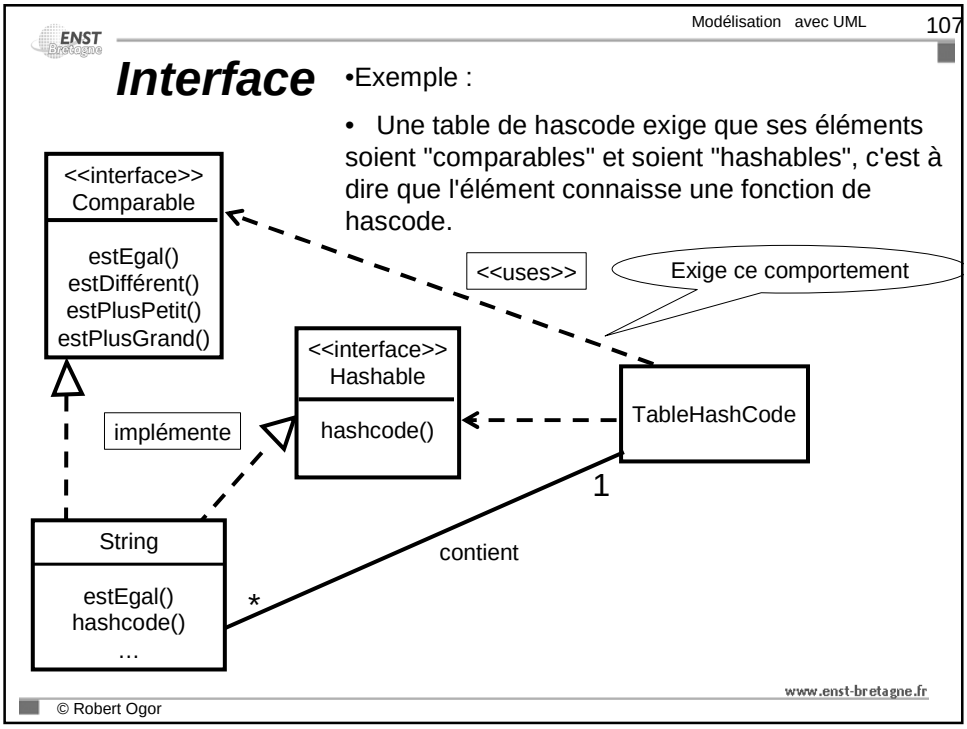
Elle ne peut pas servir à créer un objet

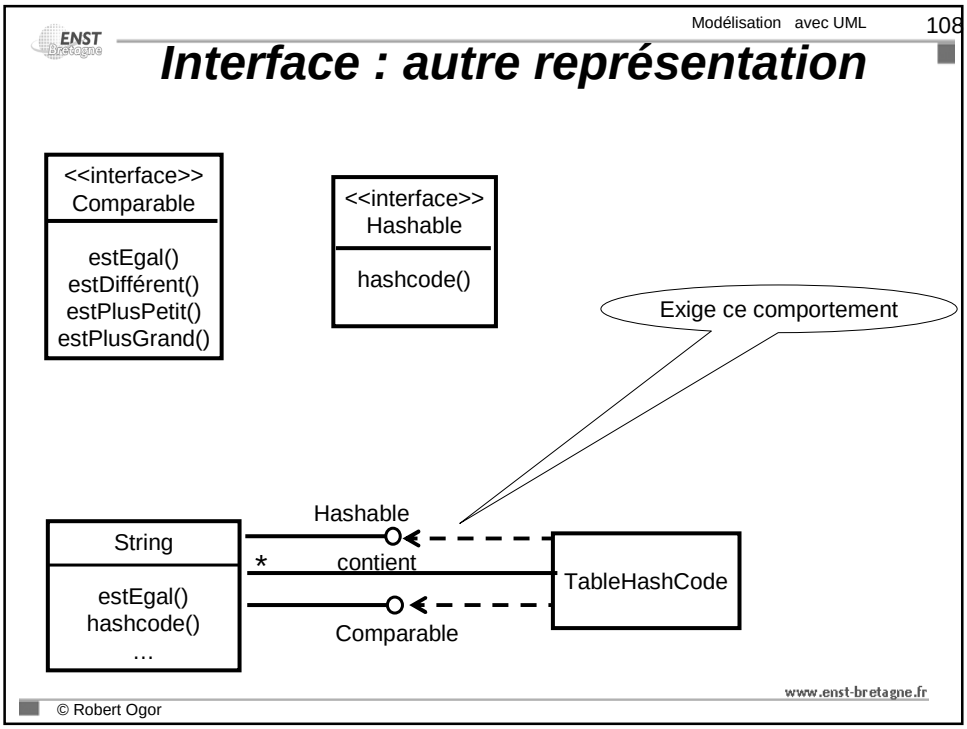
Une interface exprime un savoir faire

TYPE = CLASSE + INTERFACE


Modélisation avec UML
105
Implémentation d'interface
<<interface>>
Déplaçable
saPlace ()
avancer ()
reculer ()
monter ()
descendre ()
Savoir Faire
défini
Polygone
sommets
saPlace ()
avancer ()
reculer ()
monter ()
descendre ()
périmètre ()
Implémente
=
obligation de programmation
Sait Faire
fourni
Personne
nom
numeroSécu
saPlace ()
avancer ()
reculer ()
monter ()
descendre ()
sonNom()
sonNumero()
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML
106
Implémentation et dépendance
Polygone
sommets
saPlace ()
avancer ()
reculer ()
monter ()
descendre ()
périmètre ()
Transporteur
Déplaçable
Savoir Faire
implémente
exige (dépendance)
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML
107
Interface
•Exemple :
• Une table de hascode exige que ses éléments soient "comparables" et soient "hashables", c'est à dire que l'élément connaisse une fonction de hascode.
<<interface>>
Comparable
estEgal()
estDifférent()
estPlusPetit()
estPlusGrand()
<<uses>>
Exige ce comportement
<<interface>>
Hashable
hashcode()
implémente
TableHashCode
1
String
estEgal()
hashcode()
…
contient
*
www.enst-bretagne.fr
© Robert Ogor

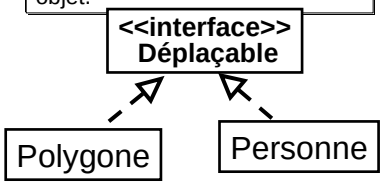

Modélisation avec UML
108
Interface : autre représentation
<<interface>>
Comparable
estEgal()
estDifférent()
estPlusPetit()
estPlusGrand()
<<interface>>
Hashable
hashcode()
Exige ce comportement
Hashable
String
estEgal()
hashcode()
…
*
contient
TableHashCode
Comparable
www.enst-bretagne.fr
© Robert Ogor

Utilisation d'interface

Une interface s'utilise pour désigner un objet.

Il définit l'ensemble des services que doit savoir cet objet.

```
void manipuler(Déplaçable element)
   { …
        element.avancer(8);
        element.monter(4);
… };
```




```
Personne  jean;
Polygone   poly;
```

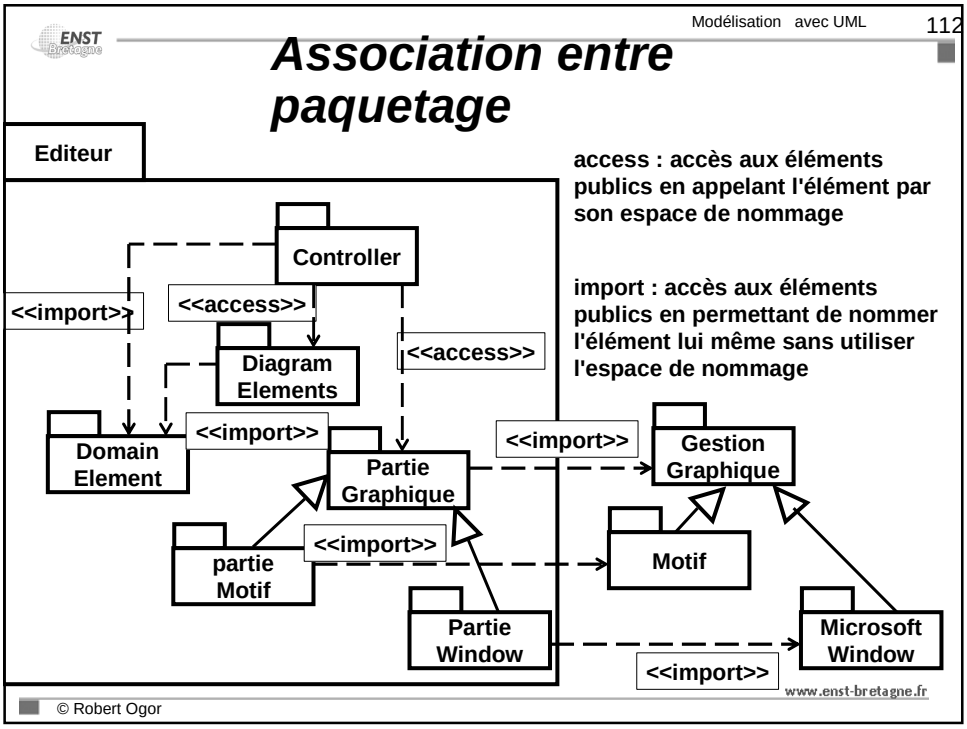
Personne et Polygone savent faire les opérations définies dans Déplaçable

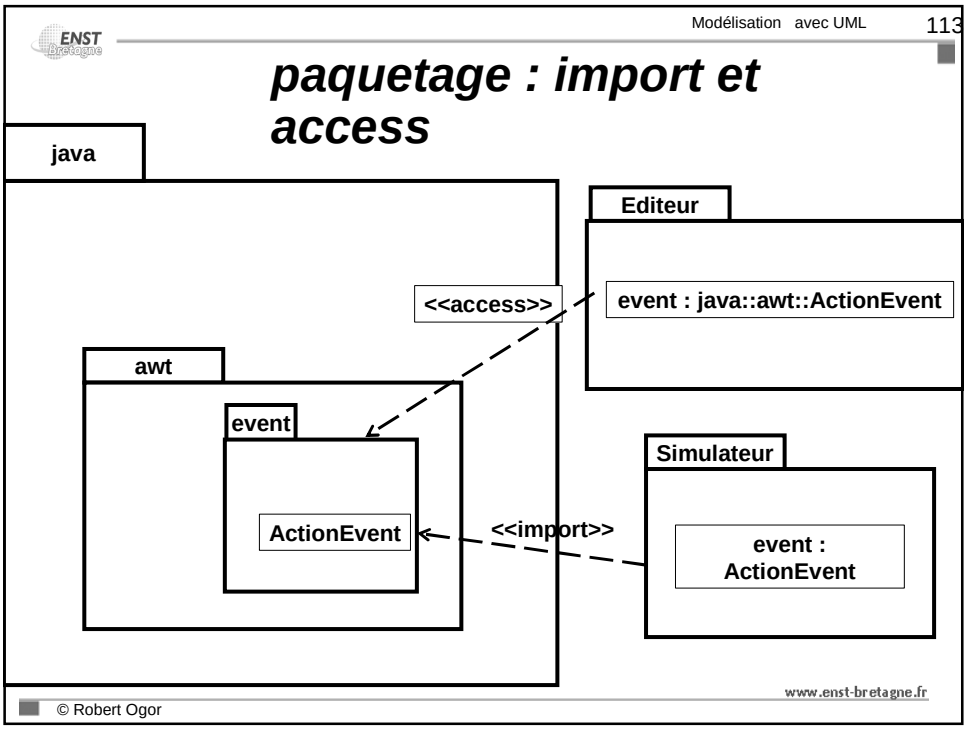
```
obj. manipuler(jean) ;
```
// licite car Personne implémente Déplaçable

```
obj. manipuler(poly) ;
```
// licite car Polygone implémente Déplaçable

Les Paquetages

- Une application est constituée de plusieurs classes, des dizaines ou des centaines. Il est important de les organiser en groupes (en fonction de certains critères surtout logiques).
- C'est le paquetage (package) qui permet ce regroupement.
- Un paquetage regroupe des classes, des interfaces, des paquetages.
- Il permet d'encapsuler certains éléments de la modélisation. Un élément du paquetage peut être inaccessible de l'extérieur du paquetage, il n'est alors connu que par les éléments du même paquetage.
- Il met en œuvre un espace de nommage


Modélisation avec UML
112
Association entre paquetage
Editeur
access : accès aux éléments publics en appelant l'élément par son espace de nommage
import : accès aux éléments publics en permettant de nommer l'élément lui même sans utiliser l'espace de nommage
Controller
<<import>>
<<access>>
Diagram Elements
<<access>>
Domain Element
<<import>>
Partie Graphique
<<import>>
Gestion Graphique
partie Motif
<<import>>
Motif
Partie Window
Microsoft Window
<<import>>
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML
113
paquetage : import et access
java
Editeur
<<access>>
event : java::awt::ActionEvent
awt
event
ActionEvent
Simulateur
<<import>>
event : ActionEvent
www.enst-bretagne.fr
© Robert Ogor

Diagrammes de classes de la gestion de bibliothèque, recherche à partir du cahier des charges.

Recherche par responsabilité

Phases de la modélisation objet

- Identifier les classes candidates.
- Préparer le dictionnaire de données : classes retenues.
- Identifier les associations entre classes (en incluant les agrégations).
- Identifier les attributs.
- Organiser et simplifier les classes en utilisant l'héritage.
- Supprimer les associations inutiles
- Vérifier que le diagramme inclut toutes les demandes du cahier des charges.
- Itérer et affiner le modèle.
- Grouper les classes en modules (paquetages).

Identifier les classes : les classes candidates

- Un gérant de bibliothèque désire automatiser la gestion des prêts.
- Il commande un logiciel permettant aux utilisateurs de connaître les livres présents, d'en réserver jusqu'à 2. L'adhérent peut connaître la liste des livres qu'il a empruntés ou réservés.
- L'adhérent possède un mot de passe qui lui est donné à son inscription.
- L'emprunt est toujours réalisé par les employés qui travaillent à la bibliothèque. Après avoir identifié l'emprunteur, ils savent si le prêt est possible (nombre max de prêts = 5), et s'il a la priorité (il est celui qui a réservé le livre).
- Ce sont les employés qui mettent en bibliothèque les livres rendus et les nouveaux livres. Il leur est possible de connaître l'ensemble des prêts réalisés dans la bibliothèque

Gérant bibliothèque gestion prêts logiciel utilisateurs livres
adhérent liste mot de passe inscription emprunt employés emprunteur ensemble

Les classes retenues

- Gérant — non pertinente, n'intervient pas
- *bibliothèque* — *oui* responsabilité : gérer les livres, adhérents, prêts
- gestion — non vague
- *prêts* — *oui* responsabilité : contenir les infos et actions sur les prêts
- logiciel — non vague
- utilisateurs — (choix entre utilisateur, adhérent, emprunteur)
- *livres* — *oui* responsabilité : permettre de connaître son état
- *adhérent* — *oui* responsabilité : permettre à la personne d'être identifiée
- liste — non implémentation ou conception
- mot de passe — non attribut
- Inscription — non action
- emprunt — non action
- *employés* — *oui* responsabilité : reconnaître qui a fait un prêt, etc..
- emprunteur — (choix entre utilisateur, adhérent, emprunteur)
- Ensemble — non implémentation ou conception

Dictionnaire des données

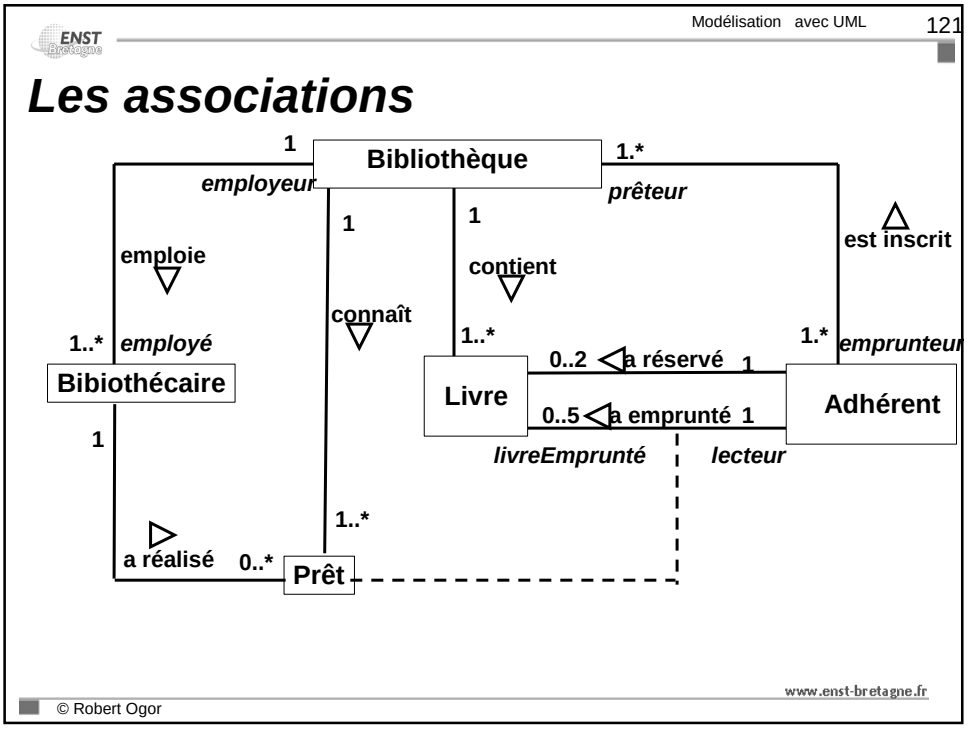
- bibliothèque : organisme gérant une collection de livres qui peuvent être empruntés par ses adhérents. Une bibliothèque est gérée par ses employés.
- prêt : un prêt est caractérisé par le numéro du livre, la date, la durée. Il ne peut être fait que par un adhérent.
- livre ouvrage pouvant être emprunté.
- adhérent personne inscrite à la bibliothèque.
- employé personne travaillant à la bibliothèque.

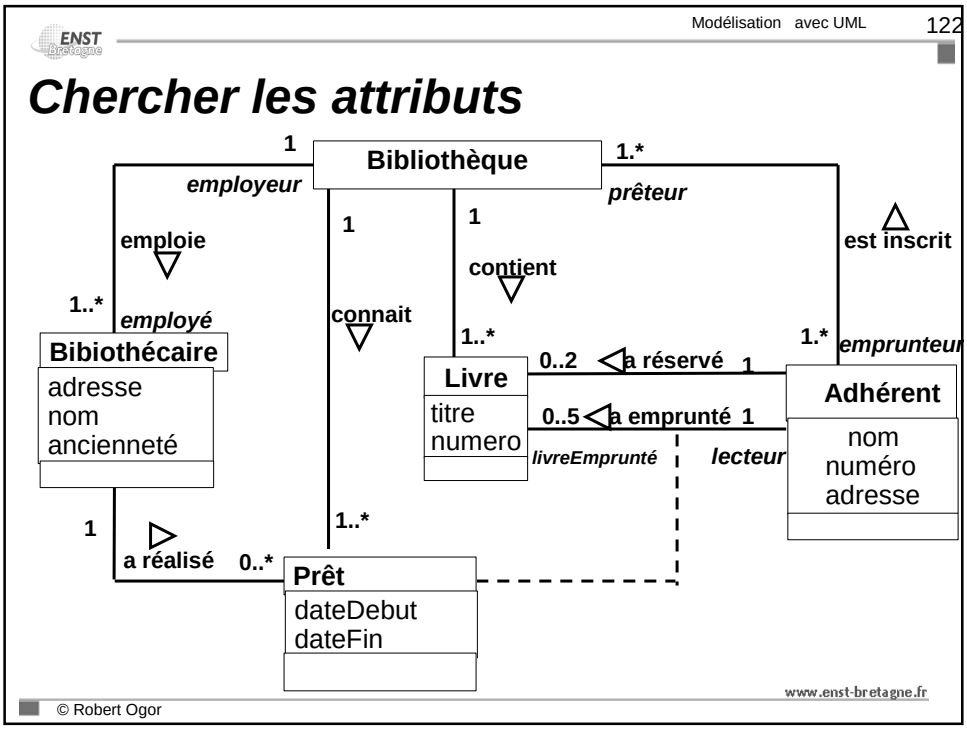
Chercher les associations

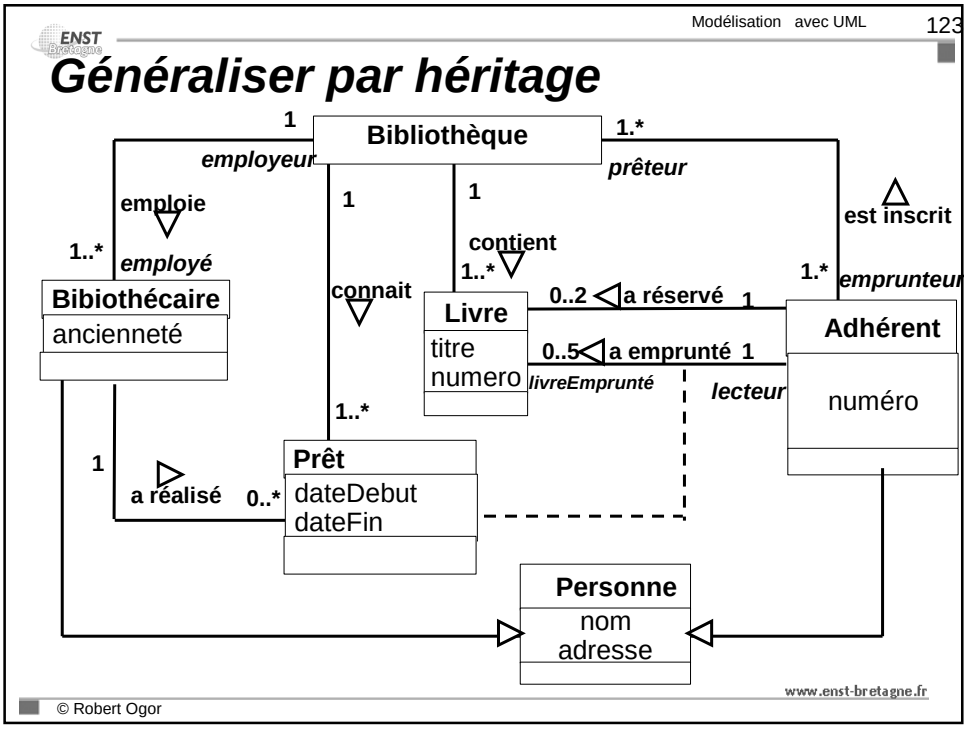
- Un gérant de bibliothèque désire automatiser la gestion des prêts.
- Il commande un logiciel permettant aux utilisateurs de *connaître* les livres présents, d'en *réserver* jusqu'à 2. L'adhérent peut connaître la liste des livres qu'il *a empruntés* ou *réservés*.
- L'adhérent possède un mot de passe qui lui est donné à son inscription.
- L'emprunt *est* toujours *réalisé* par les employés qui *travaillent* à la bibliothèque. Après avoir identifié l'emprunteur, ils savent si le prêt est possible (nombre max de prêts = 5), et s'il a la priorité (il est celui qui a réservé le livre).
- Ce sont les employés qui mettent en bibliothèque les livres rendus et les nouveaux livres. Il leur est possible de connaître l'ensemble des prêts *réalisés* dans la bibliothèque

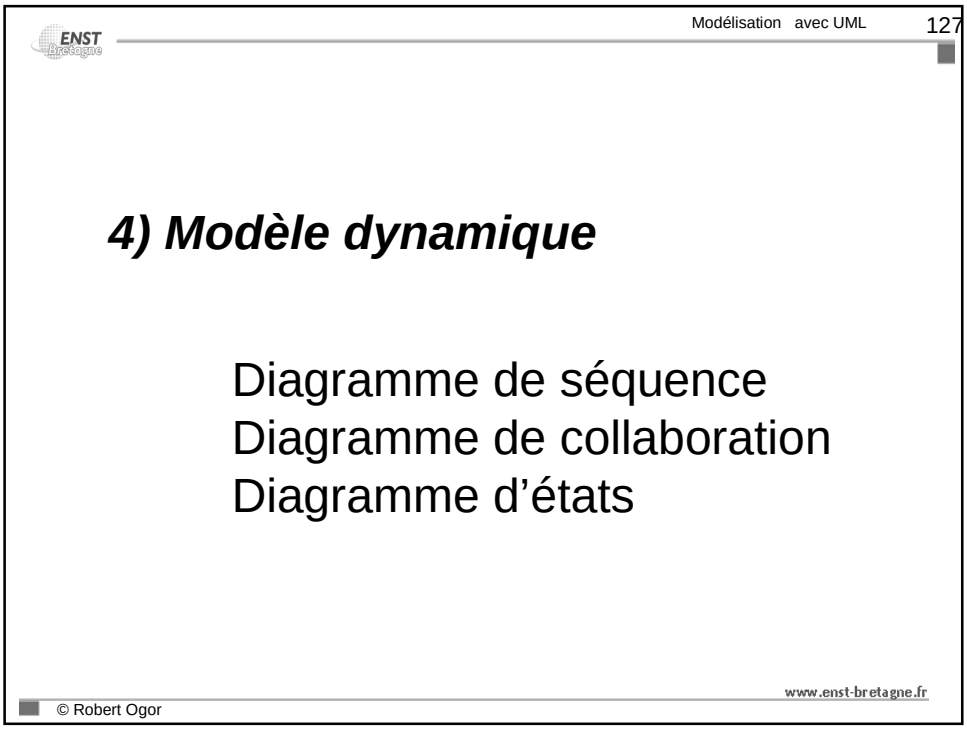
Associations sous entendues

Une adhérent *est inscrit* à la bibliothèque.
La bibliothèque *contient* des livres


Modélisation avec UML
121
Les associations
Bibliothèque
employeur
prêteur
emploie
contient
est inscrit
connaît
employé
emprunteur
Bibiothécaire
Livre
a réservé
Adhérent
a emprunté
livreEmprunté
lecteur
a réalisé
Prêt
www.enst-bretagne.fr
© Robert Ogor

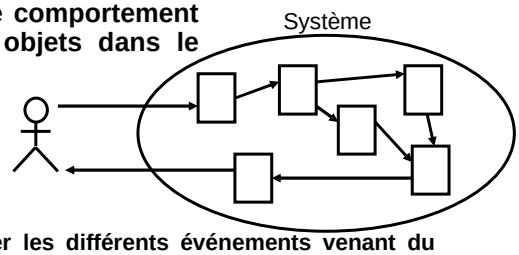

Modélisation avec UML
122
Chercher les attributs
Bibliothèque
employeur
prêteur
emploie
contient
est inscrit
connait
employé
emprunteur
Bibiothécaire
adresse
nom
ancienneté
Livre
titre
numero
a réservé
Adhérent
a emprunté
livreEmprunté
lecteur
nom
numéro
adresse
a réalisé
Prêt
dateDebut
dateFin
www.enst-bretagne.fr
© Robert Ogor





Modèle dynamique :

Le modèle dynamique montre le comportement du système et l'évolution des objets dans le temps.



Le modèle dynamique va identifier les différents événements venant du monde externe et montrer l'enchaînement dans le système que provoquent ces événements externes.

***événement* :**

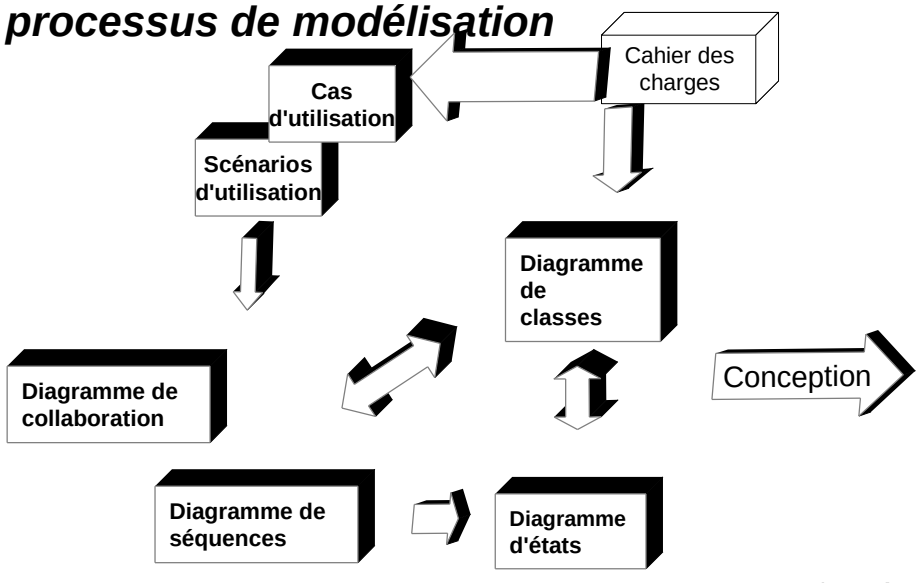
Quelque chose qui se produit à un moment donné dans le temps, et qui n'a pas de durée.

exemples : ***l'utilisateur décroche son téléphone,***
le conducteur appuie sur un bouton.

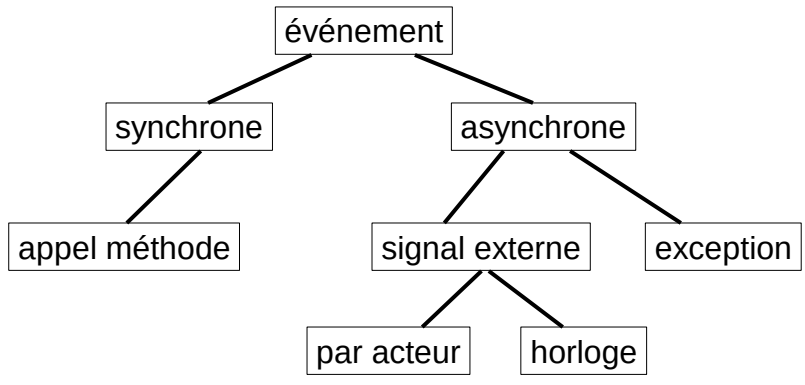
But du modèle dynamique

- Trouver les relations temporelles et événementielles entre les objets.
- Définir les états des objets qui déterminent une réaction différente face à un événement.
- Trouver les actions effectuées par les objets suite à la réception d'événements

Place du modèle dynamique dans le processus de modélisation



Classification des événements



Les types de messages

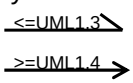
synchrone	L'expéditeur est bloqué pendant le traitement du message par l'expéditeur.
retour d'appel	Un message synchrone peut être un appel de procédure, le retour peut être représenté (optionnel, le retour est implicite)
asynchrone 	L'expéditeur continue son exécution pendant le traitement du message
minuté	Comme le synchrone, mais un chien de garde est positionné, c'est à dire que l'expéditeur se réveille au bout d'un certain temps s'il ne reçoit pas de réponse.

Diagramme de collaboration ou diagramme de séquences

- Représente la **collaboration entre objets** pour la réalisation d'un cas d'utilisation ou la réalisation d'une opération.
- Met en évidence l'expéditeur et le récepteur de l'événement.
- Chaque **événement** reçu par un objet devra se traduire par une **opération** pour le gérer dans le diagramme de classes.

Diagramme de séquence

- Met en évidence l'aspect temporel (haut vers le bas)
- Un objet a une ligne de vie représentée par une ligne verticale en pointillé.
- Une flèche reçue par un objet se traduit par l'exécution d'une opération.
- La durée de vie de l'opération est symbolisée par un rectangle.
- Certains objets vivent pendant tout le diagramme, d'autres sont activés et/ou meurent pendant la séquence.
- Il est possible de définir des choix et des itérations.

Contrôle et diagrammes de séquences

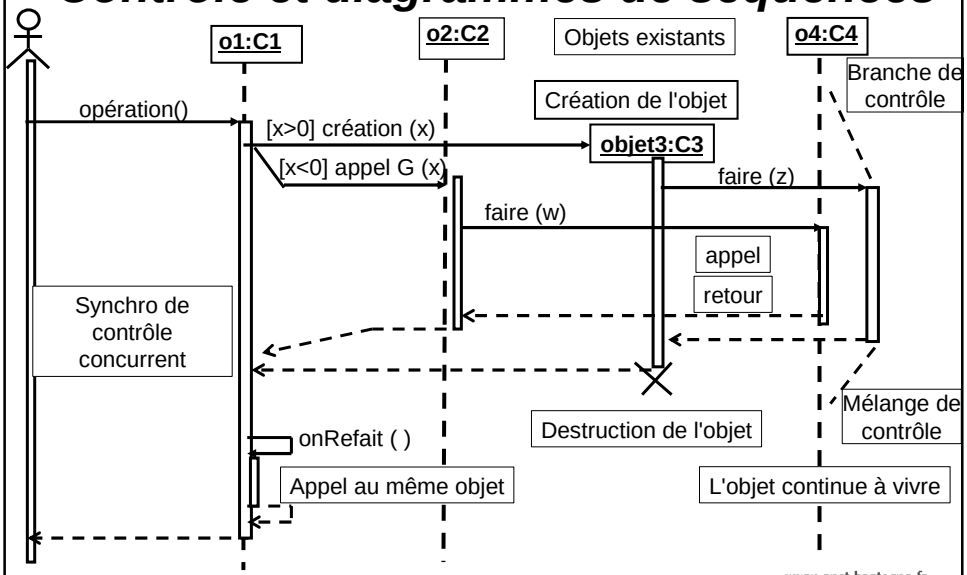
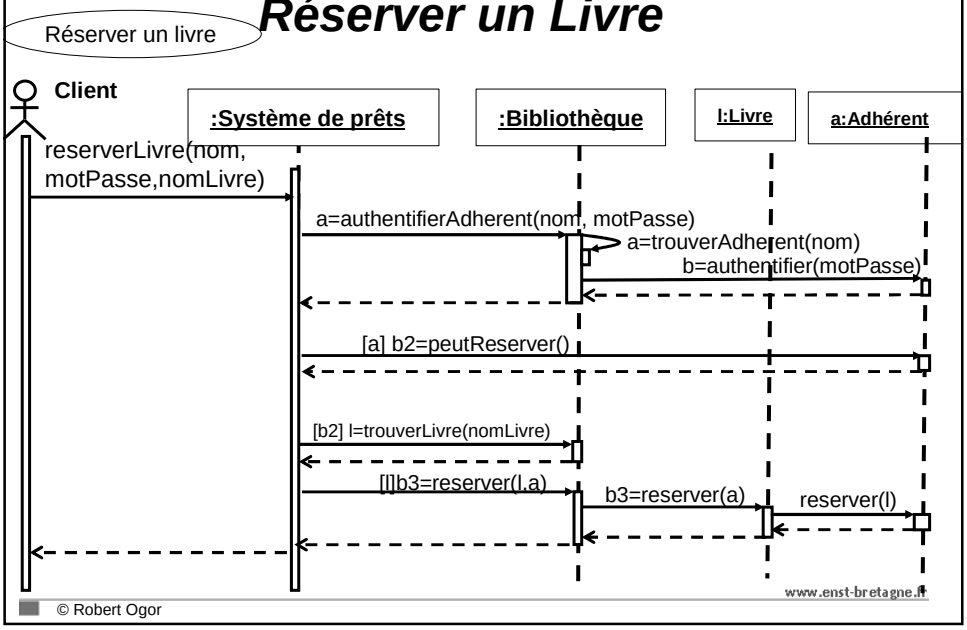
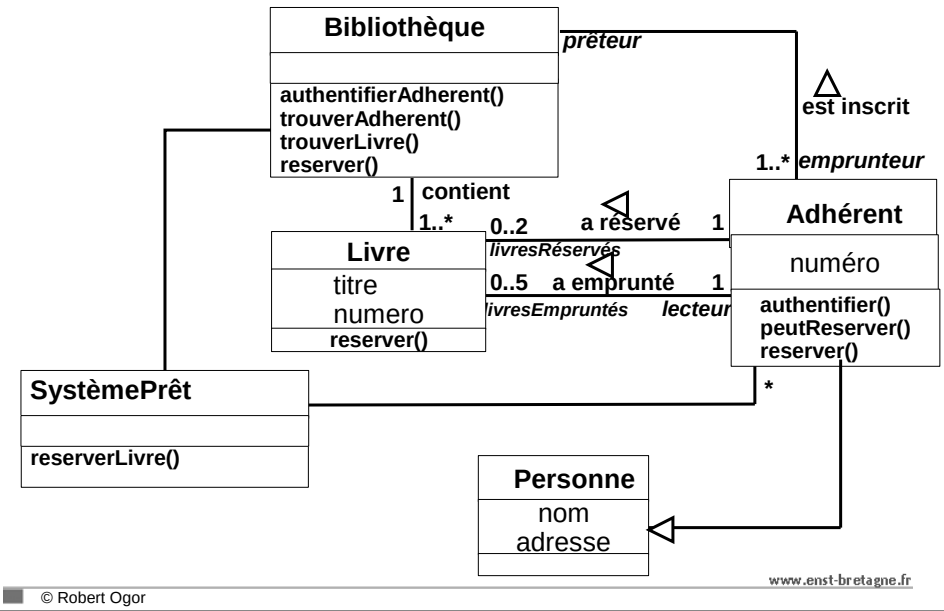


Diagramme de séquence : Réserver un Livre



Ajout des opérations dans les classes



Les Diagrammes de collaborations entre classes

Diagramme de collaborations

- Le diagramme de collaborations sous une forme distincte du diagramme de séquences représente les interactions entre classes en mettant moins en évidence l'aspect temporel.
- Ce modèle explique la coopération entre objets pour la réalisation d'une fonctionnalité, cette coopération implique les différents événements qui se propagent d'un objet à l'autre. Un objet doit avoir une méthode appropriée pour traiter chaque événement qu'il reçoit (message).
- L'aspect temporel n'est pas complètement caché car chaque message est numéroté.

Le flot de données intervenant dans ces interactions peut être précisé.

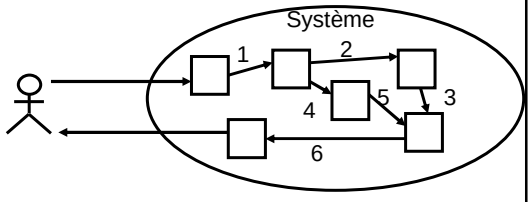
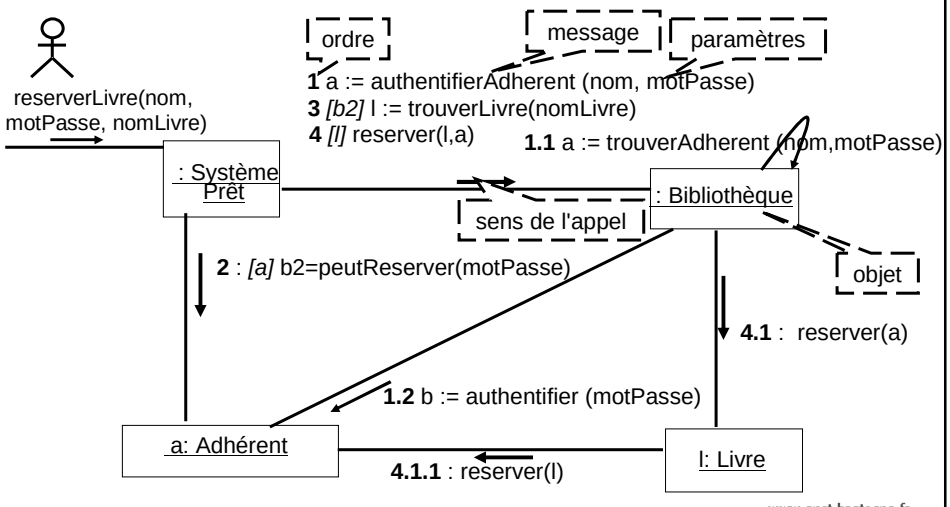


Diagramme de collaboration
Réserver un livre



Diagrammes de classes de la gestion de bibliothèque, recherche à partir des cas d'utilisation

Définition de classes d'analyses

On s'aperçoit que dans l'analyse d'un problèmes trois types de classes apparaissent couramment:

classe frontière
La classe qui permet au système de communiquer avec le monde réel, elle est à la frontière du système, elle se conçoit en général par une interface graphique, nous la représentons par l'icône suivante

réception Commande

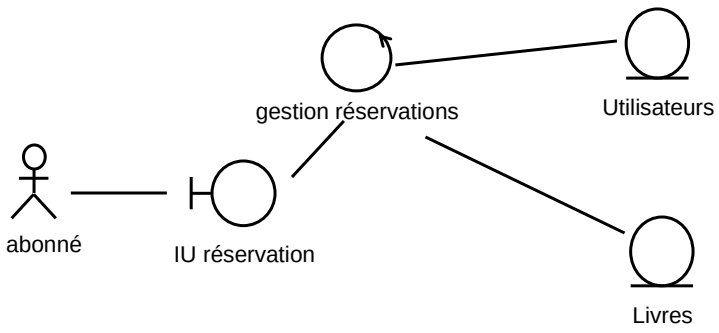
classe entité
La classe qui mémorise et gère des données, par exemples les livres présents, les prêts effectuées, nous la représentons par par l'icône suivante

stockage Commande

classe contrôle
La classe qui réalise le contrôle nécessaire pour interpréter le scénario décrivant un cas d'utilisation

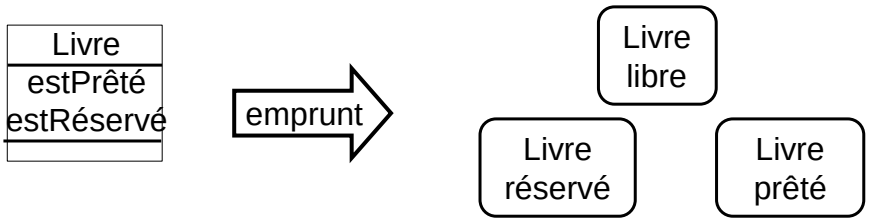
gestion Commande

Réalisation analyse cas d'utilisation



Recherche des Etats

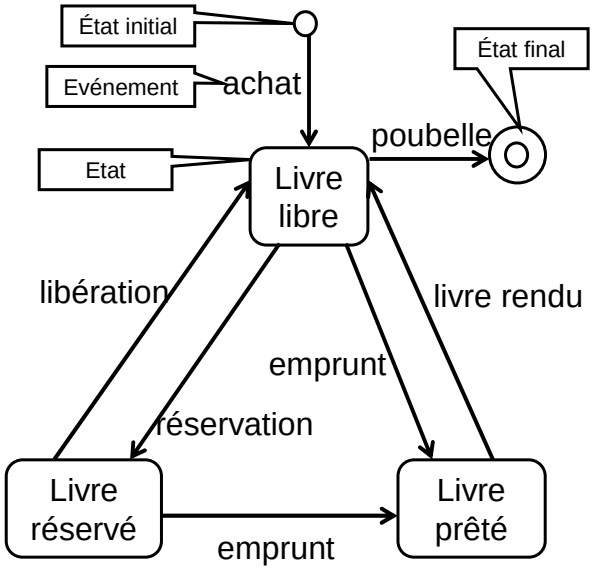
- L'état d'un objet est lié aux valeurs de ses variables d'instances et des interactions avec les autres objets. Il s'agit de ne conserver que les états significatifs.
- La réponse d'un objet à un événement dépend de l'état dans lequel cet objet se trouve.



Un objet passe dans un état donné par l'événement qui modifie ses variables, et quitte cet état par un autre événement qui les modifie à nouveau. Ce sont des transitions d'états.

Diagramme de transitions d'états

- Chaque transition est provoqué par un événement.
- Une transition n'a pas de durée.
- La durée d'un état est l'intervalle de temps qui s'écoule entre l'événement d'entrée et celui de départ.



Activité liée à un état

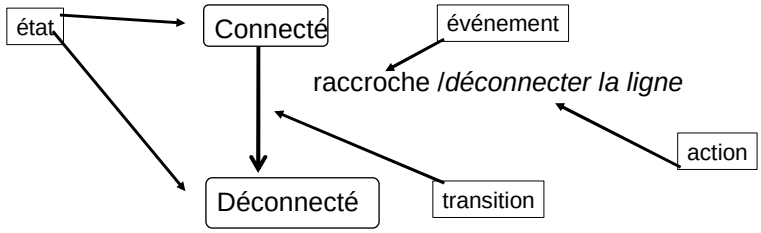
Occupé
do : jouer tonalité occupé

Dans cet état, l'activité de la ligne téléphonique est d'émettre une tonalité.

Une activité est une opération associée à un état.

Une activité « do » se répète tant qu'on est dans l'état.

Action liée à un événement

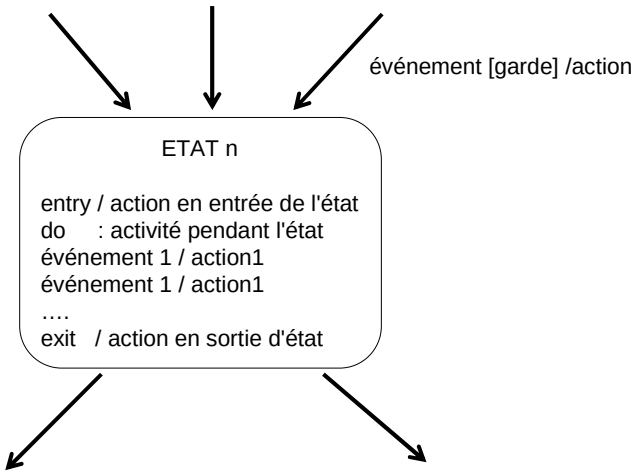


La transition connecté vers déconnecté est déclenchée par l'événement extérieur « **le poste est raccroché** », une action est réalisée avant d'entrer dans l'état déconnecté : « **la ligne est déconnectée** ».

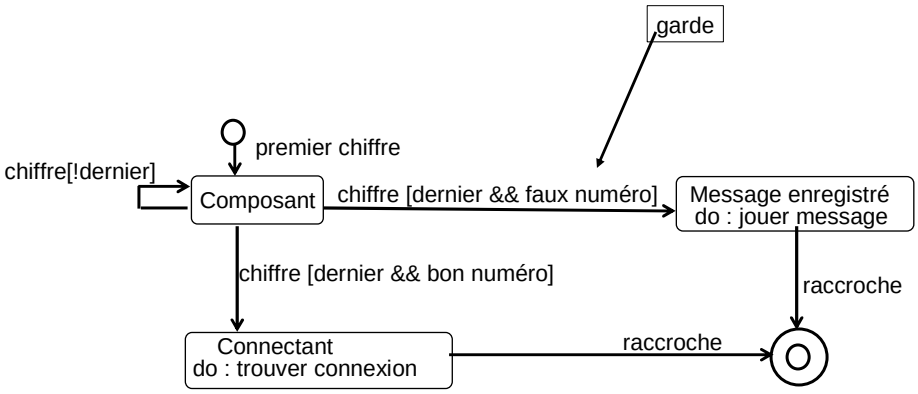
Une action n'a pas de durée, elle est associée à un événement.

Le changement de valeur d'un attribut est un exemple d'action.

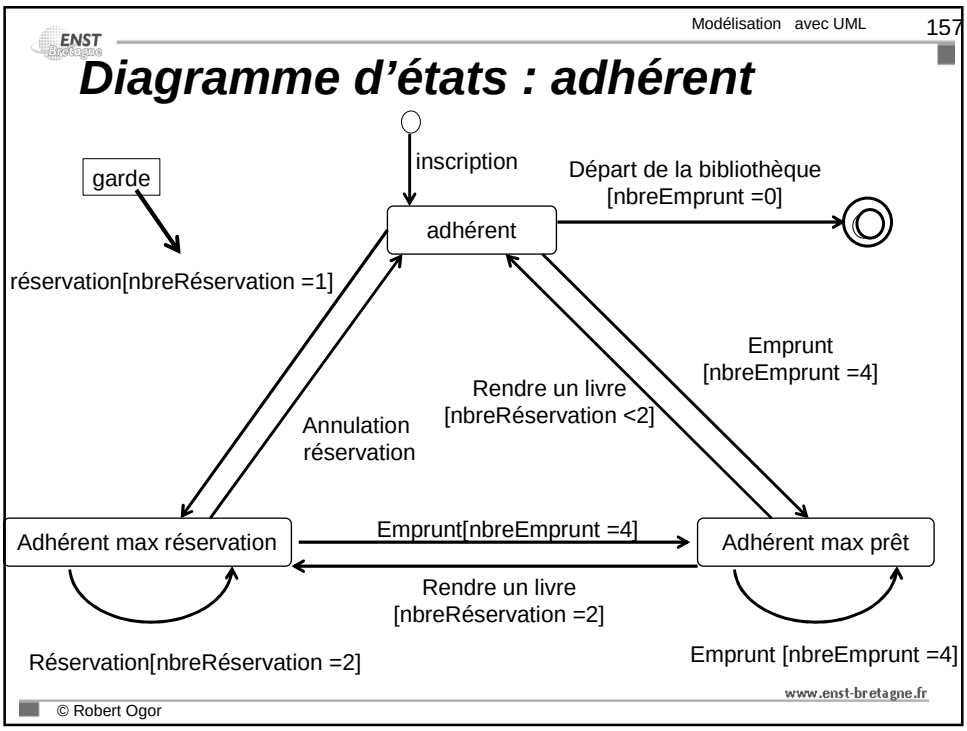
Les opérations possibles dans l'état et les transitions

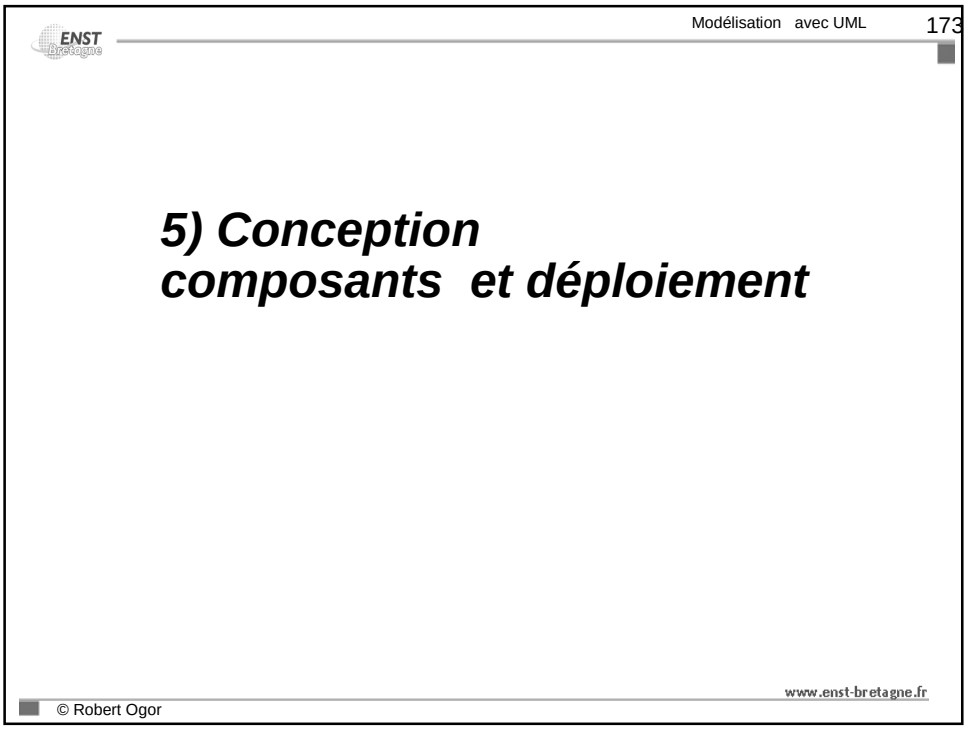


Garde



Une garde est une condition booléenne qui permet lors de l'occurrence d'un événement d'accepter une transition ou pas.


Modélisation avec UML
157
Diagramme d'états : adhérent
inscription
garde
Départ de la bibliothèque
[nbreEmprunt =0]
adhérent
réservation[nbreRéservation =1]
Emprunt
[nbreEmprunt =4]
Rendre un livre
[nbreRéservation <2]
Annulation
réservation
Adhérent max réservation
Emprunt[nbreEmprunt =4]
Adhérent max prêt
Rendre un livre
[nbreRéservation =2]
Réservation[nbreRéservation =2]
Emprunt [nbreEmprunt =4]
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML
173
5) Conception
composants et déploiement
www.enst-bretagne.fr
© Robert Ogor

La notion de composant

Dans le monde du bâtiment, le modèle de l 'architecte (logique) permet de visualiser, spécifier et documenter sur papier les caractéristiques de la future construction :

place des murs, des fenêtres, etc…

Lors de la construction, on utilise des composants fenêtres, portes, murs. Ce sont des choses physiques qui existent dans le monde réel.

Ils rendent des services mais définissent leurs exigences (taille, espace, etc.).

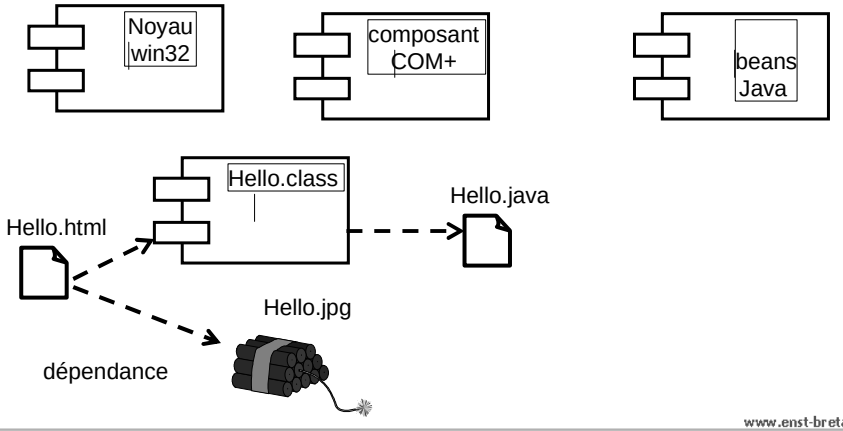
De même, dans un système informatique, le modèle logique dans une application permet de visualiser, spécifier et documenter la structure et le comportement des entités qui collaborent.

La construction va s 'appuyer sur des composants qui existent dans le monde des ordinateurs : librairies, fichiers, tables, documents exécutables, etc

Un composant définit les services qu'il rend et aussi les services dont il est en attente pour pouvoir fonctionner.

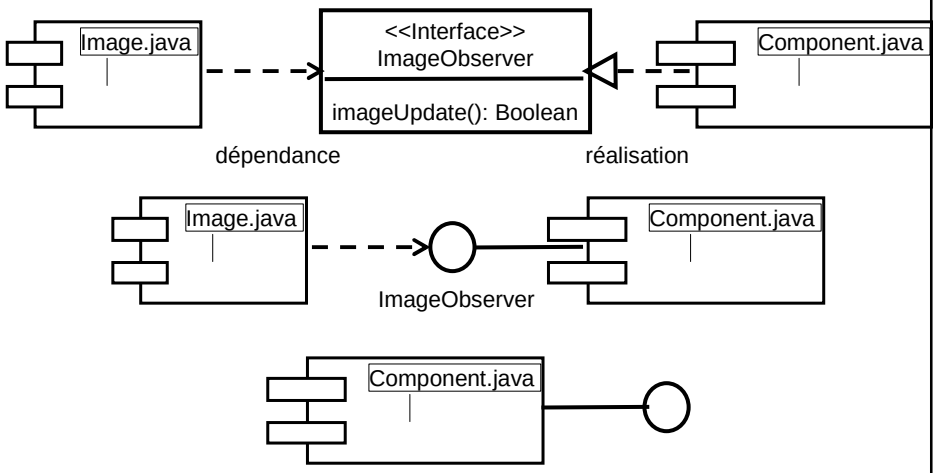
Définition des composants

- Un composant est une partie physique et remplaçable d'un système qui sait faire et fournit la réalisation d'un ensemble d'interfaces.



Interface et composant

Un composant offre une interface, mais exige la disposition de certains services

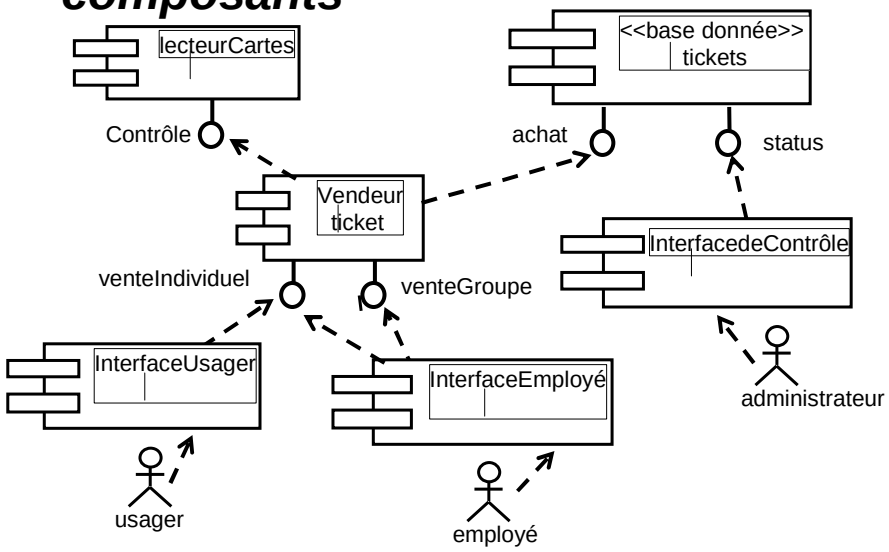


Les diagrammes de composants

Les diagrammes de composants

- Ce modèle définit l'architecture logicielle du système dans un environnement de développement donné.
- Il est issu de la conception et permet de représenter le système et les sous systèmes du modèle physique de l'architecture logicielle à réaliser.
- Un système ou un sous système défini un espace de visibilité et regroupe des classes.
- Tout les langages ne supporte pas cette notion de système mais elle existe sous forme de "package" en ADA.

Les diagrammes de composants

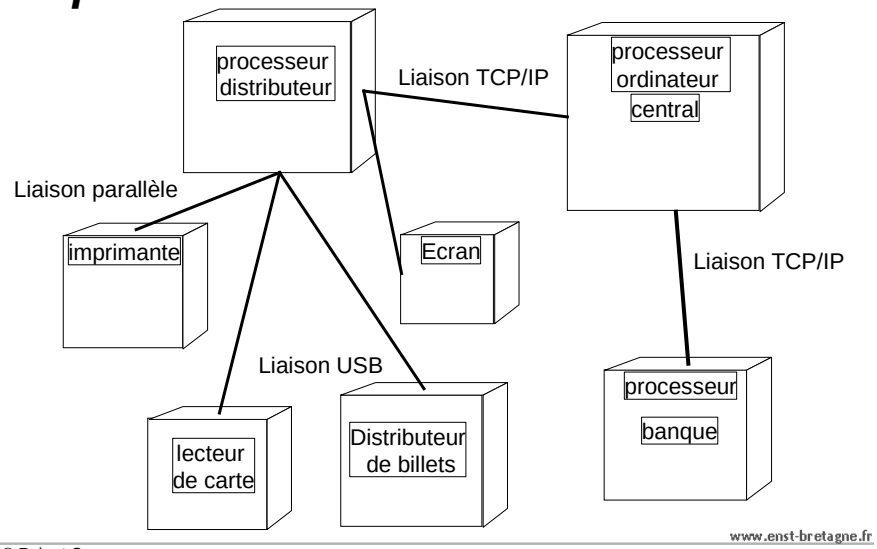


Les diagrammes de déploiement

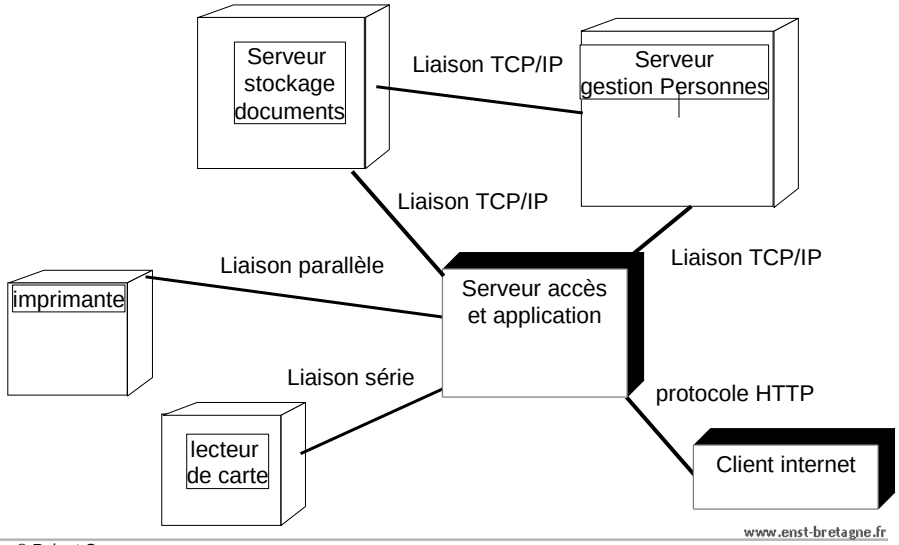
Les diagrammes de déploiement

- Ce modèle définit le diagramme de l'architecture matérielle du système ou de l'application.
- Il représente les différents processeurs, périphériques et la répartition du système sur ces différents éléments.
- Il montre les liens de communication entre ces différentes entités.

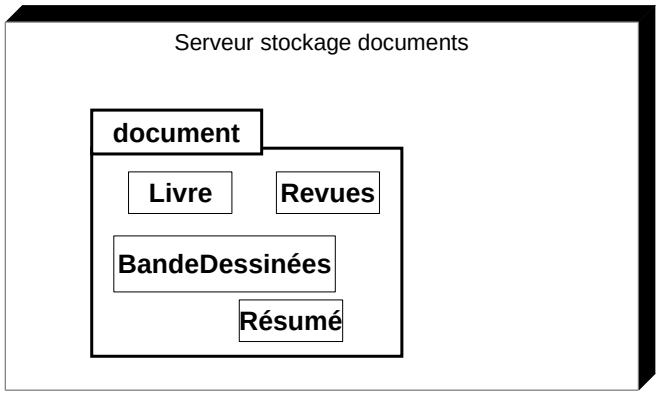
Les diagrammes de déploiement



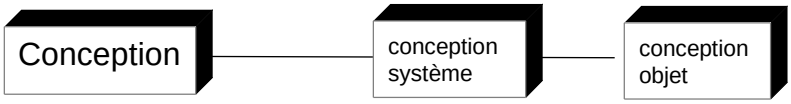
Les diagrammes de déploiement de l'application



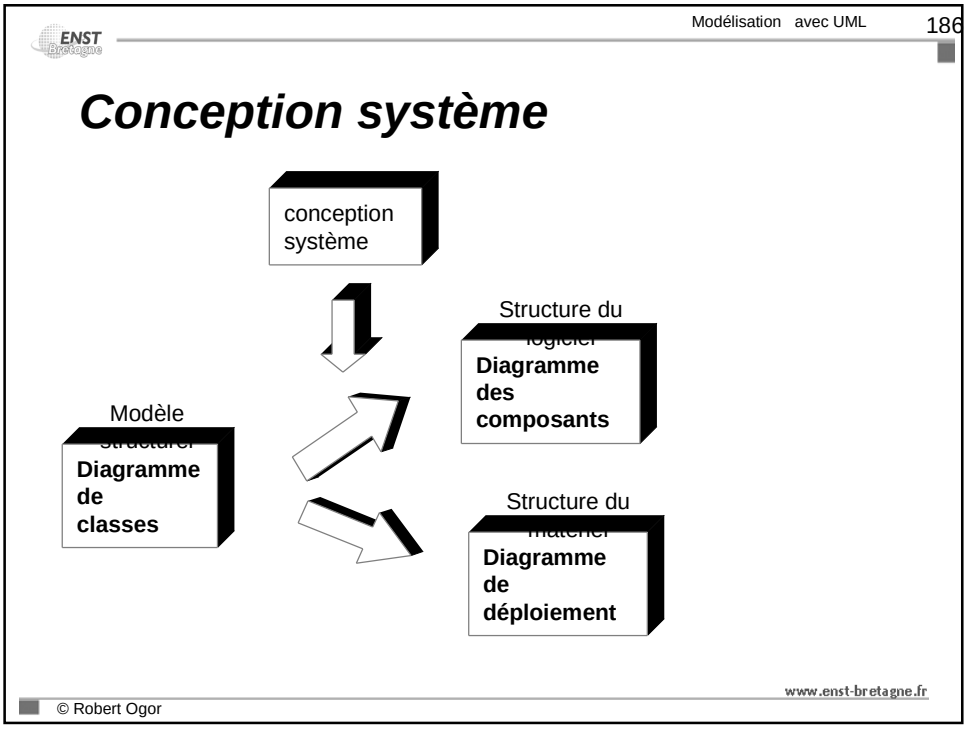
Les diagrammes de déploiement de l'application

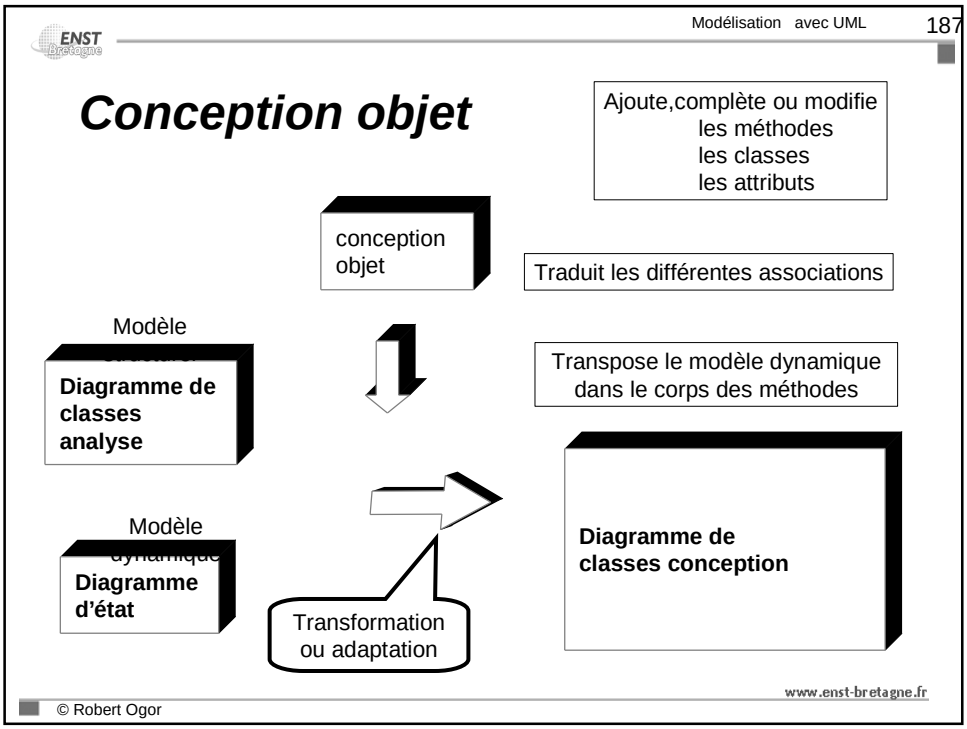


Conception

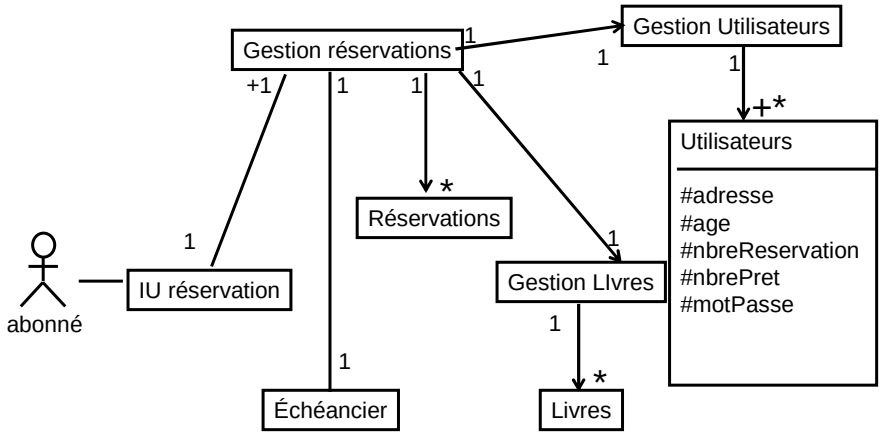


- La phase de conception a pour objectif de rechercher comment le système va être réalisé, contrairement à l'analyse qui rechercher ce qui doit être fait (quoi).
- Elle élabore les différentes parties du système et leurs interactions d'abord à un niveau général puis à un niveau de plus en plus détaillé.
- Elle tient compte des contraintes matérielles et logicielles : langages, bases de données, processeurs, périphériques, etc..
- C'est une étape où ne peuvent intervenir que des informaticiens spécialisés dans les différentes technologies utilisées.


Modélisation avec UML
186
Conception système
conception système
Structure du logiciel
Diagramme des composants
Modèle structurel
Diagramme de classes
Structure du matériel
Diagramme de déploiement
www.enst-bretagne.fr
© Robert Ogor


Modélisation avec UML
187
Conception objet
Ajoute,complète ou modifie
les méthodes
les classes
les attributs
conception objet
Traduit les différentes associations
Modèle structurel
Diagramme de classes analyse
Transpose le modèle dynamique dans le corps des méthodes
Modèle dynamique
Diagramme d'état
Transformation ou adaptation
Diagramme de classes conception
www.enst-bretagne.fr
© Robert Ogor

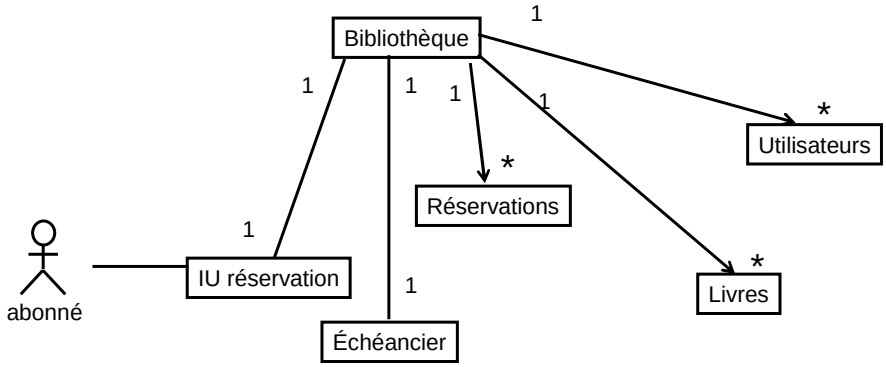
Conception



Mise en œuvre de la **navigabilité**, choix de qui accède à qui.
Mise en œuvre de la **protection** des attributs et des relations

www.enst-bretagne.fr

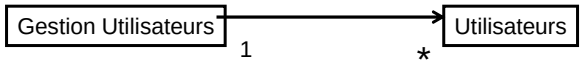
Conception du diagramme de classes



Les différentes **classes de contrôle** sont regroupées dans une seule que l'on appelle Bibliothèque.
Cette solution n'est pas possible si les livres et les utilisateurs sont sur des serveurs différents
Classe frontière (interface utilisateur).
Classe entité (base des livres, base des utilisateurs …)

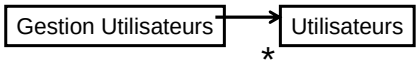
www.enst-bretagne.fr

Conception : choix de persistance

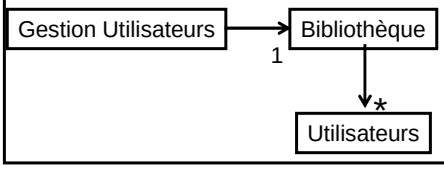


Choix de conception :

Utilisateurs est une table dans une base de données
Ce choix est guidé par la persistance.



Ou bien, il est défini, par exemple en Java, par une liste d'utilisateurs, au programmeur d'assurer la persistance par des fichiers.



Conclusions

- Méthode pour appréhender la réalisation d'un système informatique
- Diagrammes qui permettent d'aider à la réflexion, de permettre la discussion (clients, développeurs)
- Pas de solution unique mais un ensemble de solutions plus ou moins acceptables suivant les contraintes du client et les logiciels et matériels disponibles
- Une solution acceptable sera obtenue avec des itérations successives.