Modélisation orientée objets avec UML

Guillaume Laurent

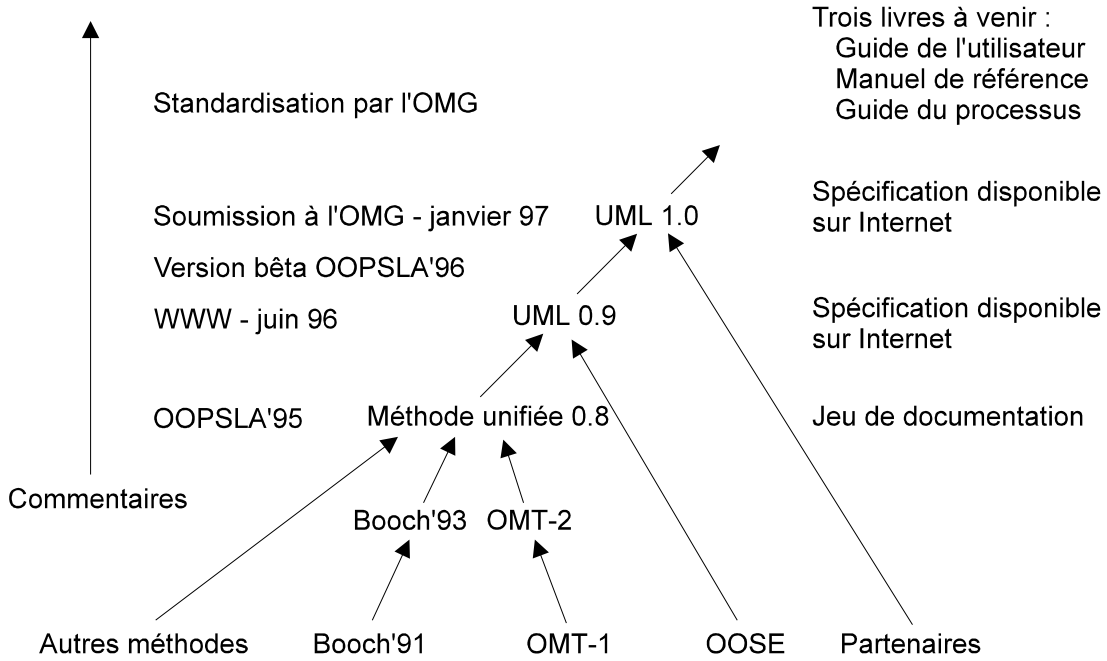
ENSMM

2007

Plan du cours

Genèse de UML (Unified Modeling Langage)

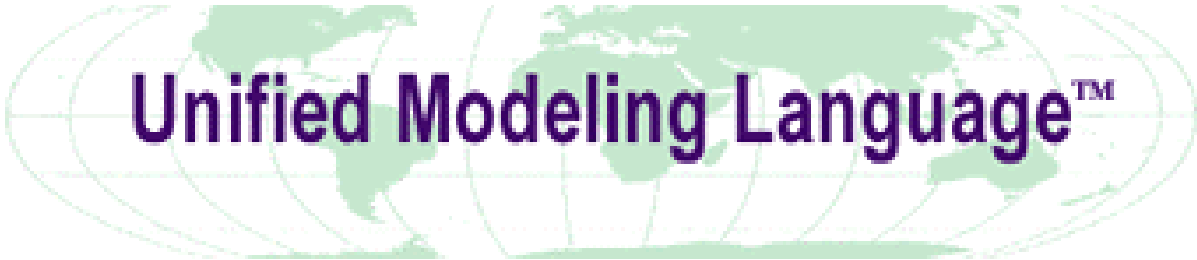
- Jusqu'au milieu des années 1990, de nombreux modèles objets sont proposés :
  - Booch'91 de Grady Booch
  - Object Modeling Technique (OMT) de James Rumbaugh en 1991 puis 1994
  - Object-Oriented Software Engineering (OOSE) de Ivar Jacobson en 1992



Source P.A. Muller

- Unification des méthodes par Rumbaugh, Jacobson et Booch et naissance d'UML 1.0 en 1997
- UML 2.0 en septembre 2004

Qu'est-ce que UML



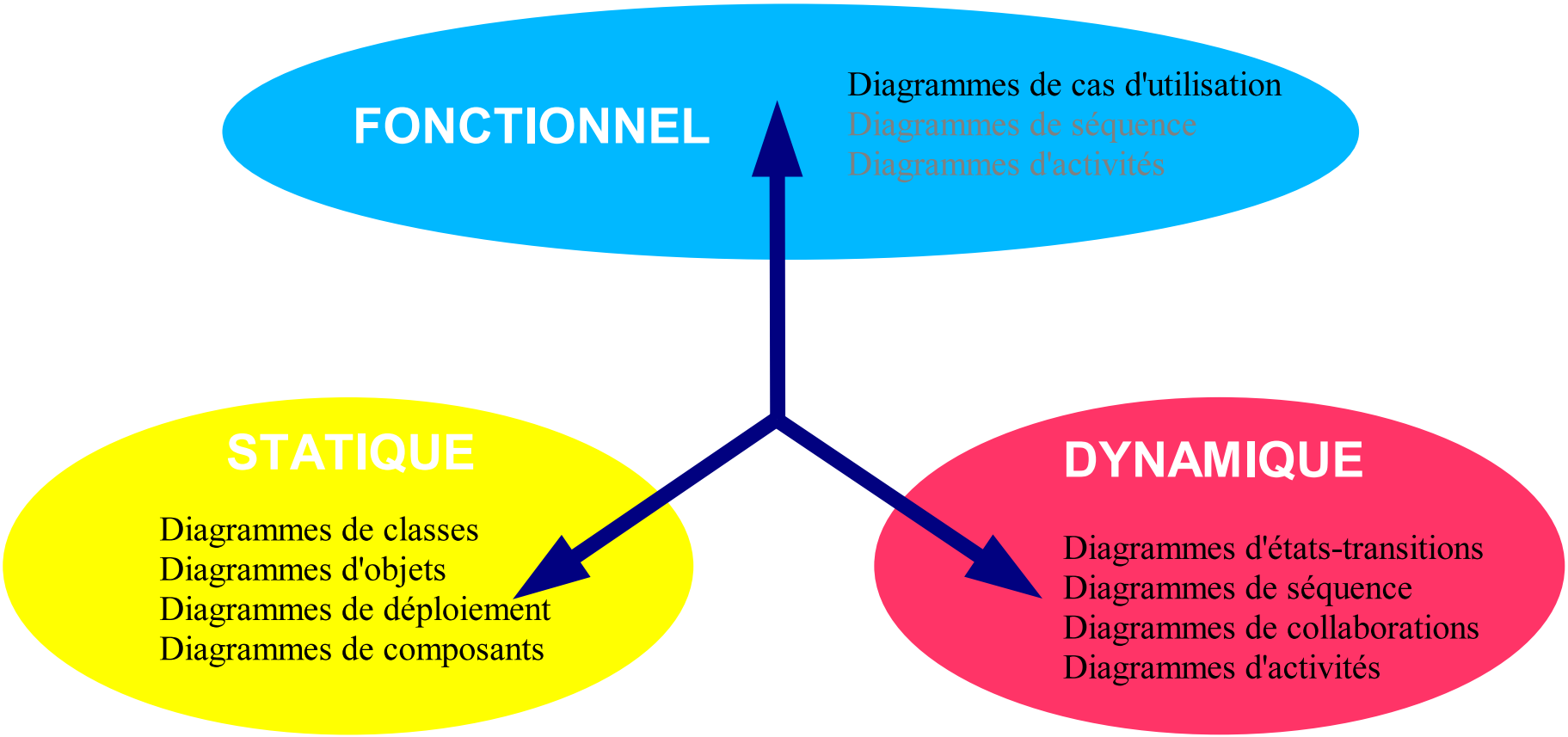
- UML est un langage universel de modélisation objets
- UML est une notation, un outil de communication visuelle (diagrammes)
- UML est une norme maintenue par l'organisation à but non lucratif OMG (Object Management Group) et disponible gratuitement sur *www.uml.org*

- UML n'est pas un processus de développement
- UML n'est pas un langage de programmation

Les diagrammes UML

- Diagrammes de cas d'utilisation : décrivent les services rendus par le système du point de vue de l'utilisateur
- Diagrammes de classes : décrivent les classes d'une application et leurs relations statiques
- Diagrammes d'objets : montrent l'état d'une application à un instant donné
- Diagrammes de composants : sont une vue modulaire de l'application en regroupant les classes qui collaborent
- Diagrammes de déploiement : modélisent l'aspect matériel de l'application
- Diagrammes de collaboration : sont une représentation spatiale des interactions entre objets
- Diagrammes de séquence : sont une représentation temporelle des interactions entre objets
- Diagrammes d'états-transitions : représentent le comportement d'un objet sous la forme d'un automate à états
- Diagrammes d'activités : représentent le comportement d'une opération

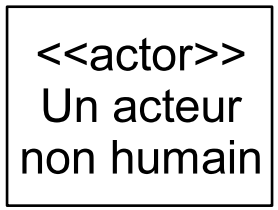
Interventions des diagrammes sur les axes de modélisation

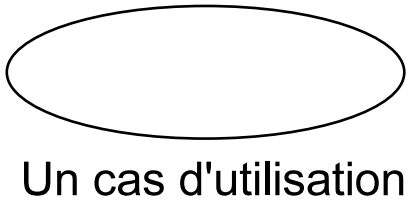


2.1 Diagrammes de cas d'utilisation

Diagrammes de cas d'utilisation

- Identifier les acteurs
  - Un acteur représente un rôle joué par une entité extérieure au système (humain ou autre système) et qui interagit directement avec le système étudié
  - Un utilisateur peut être représenté par plusieurs acteurs
  - Plusieurs utilisateurs peuvent être représentés par le même acteur

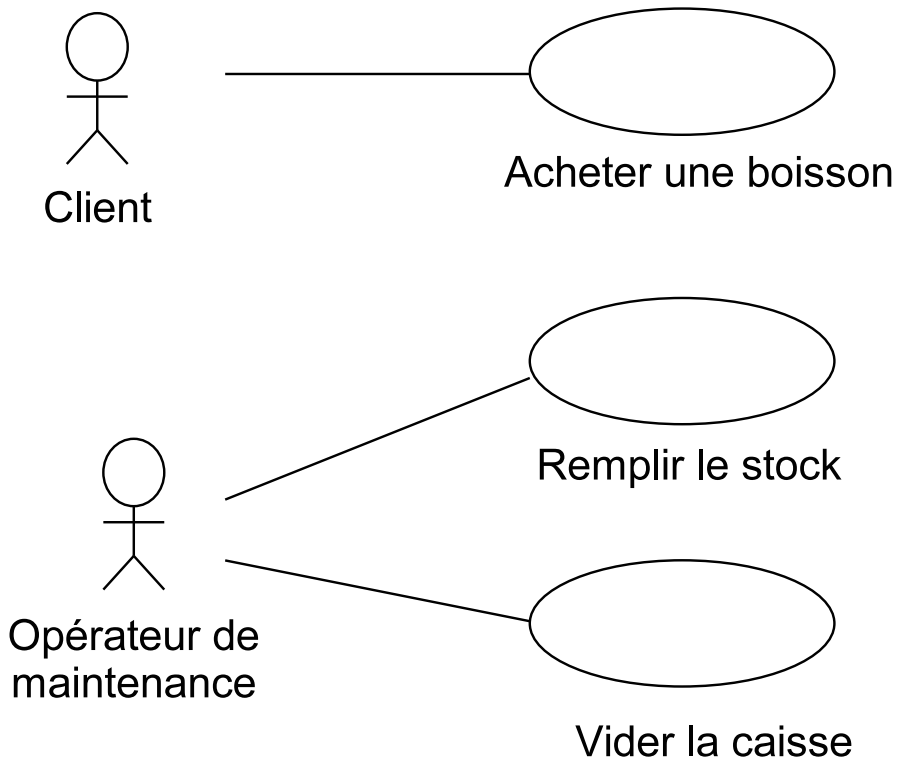




- Identifier les cas d'utilisation
  - Un cas d'utilisation représente un service complet attendu du système
  - Un cas d'utilisation a un début et une fin clairement identifiés
  - Un cas d'utilisation est décrit par une forme verbale

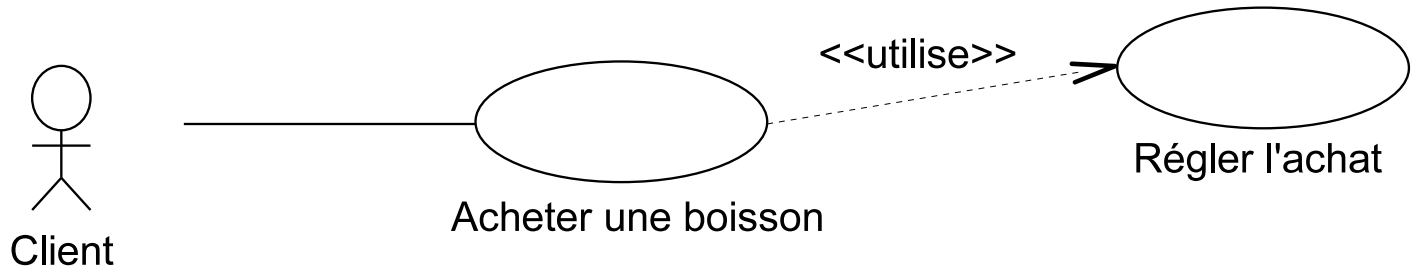
Diagrammes de cas d'utilisation

- Exemple des cas d'utilisation d'un distributeur de boissons

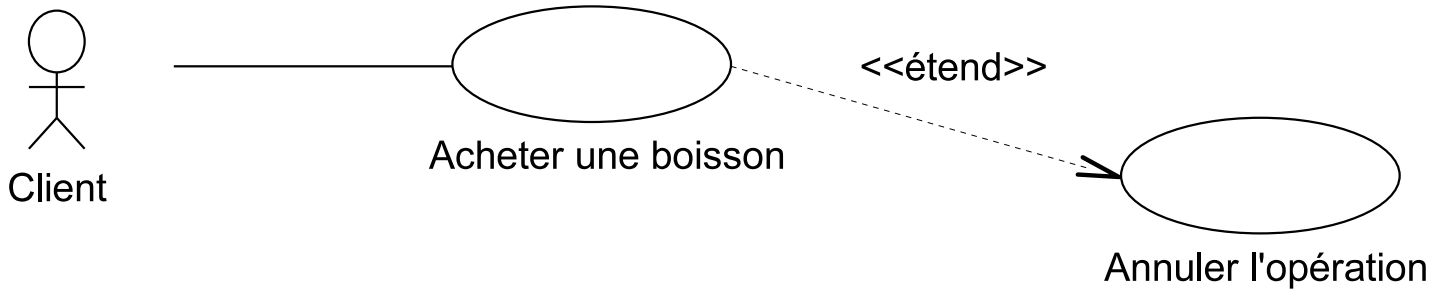


Diagrammes de cas d'utilisation

- Relation d'utilisation (include)

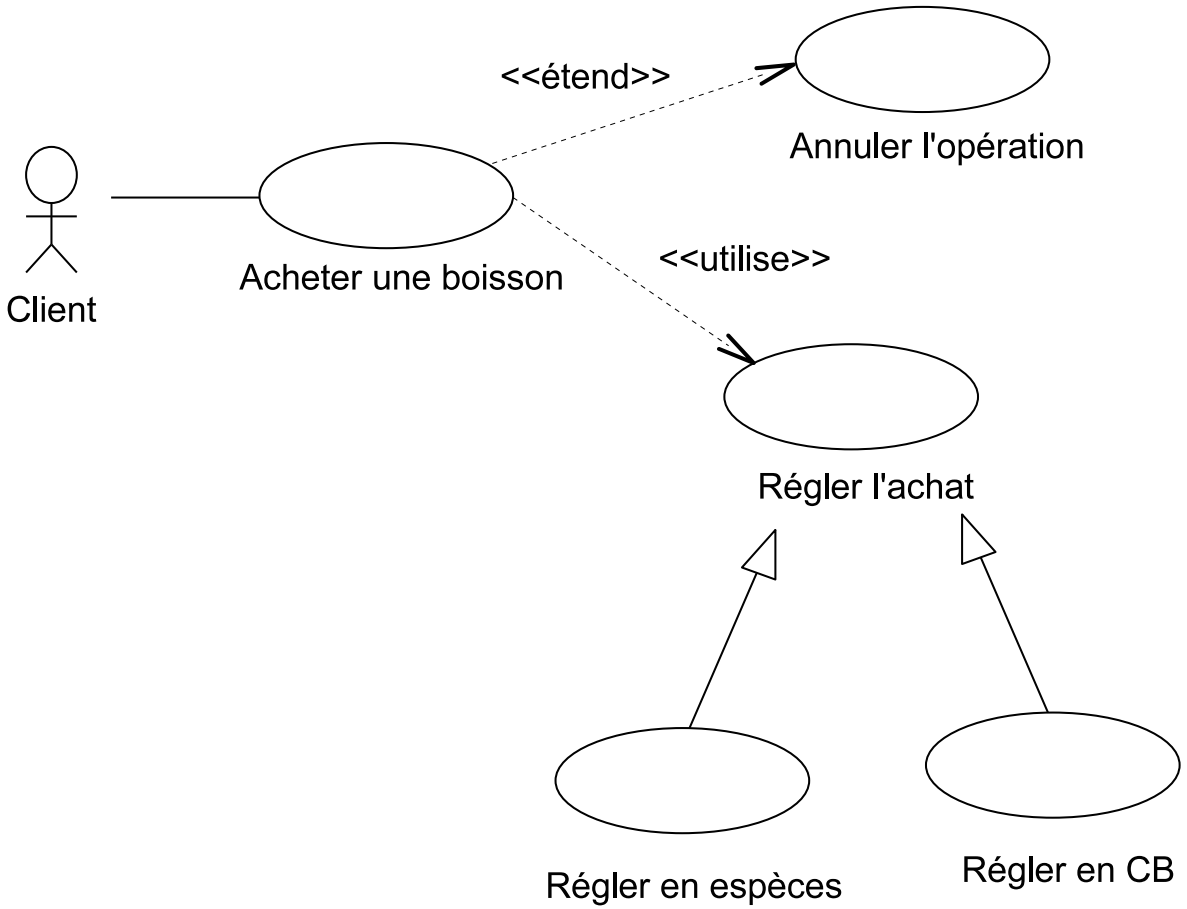


- Relation d'extension (extend)



Diagrammes de cas d'utilisation

- Relation de spécialisation



Exercices

Ex.1 Modéliser les cas d'utilisation d'un logiciel de jeu d'échecs

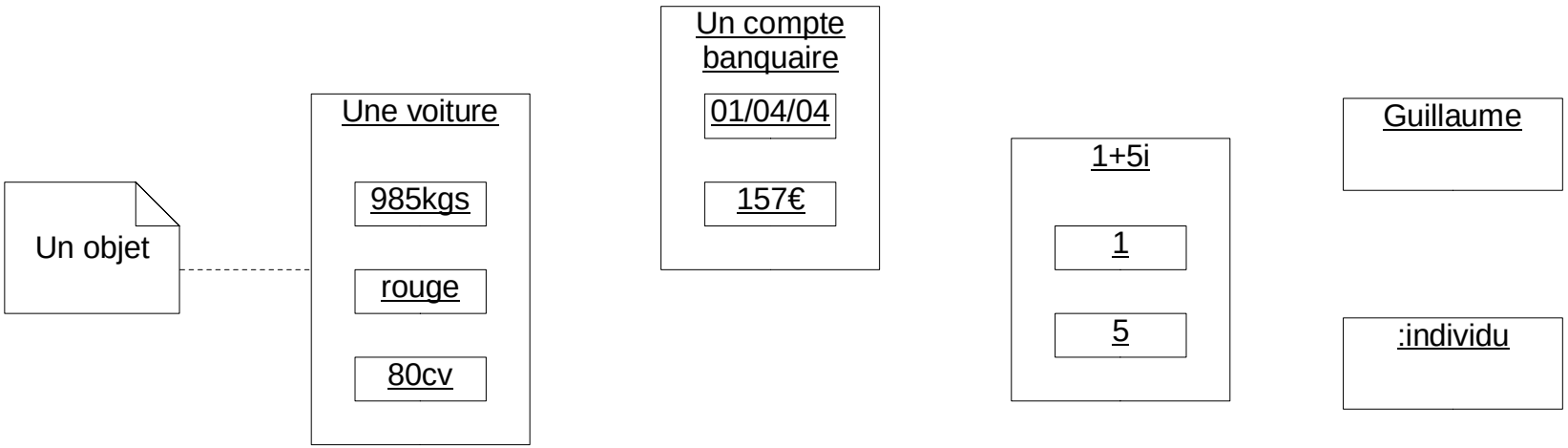
Ex.2 Modéliser les cas d'utilisation d'une caisse enregistreuse (extrait de *UML par la pratique*, Pascal Roques, Eyrolles)

- Le déroulement normal d'utilisation de la caisse est le suivant :
  - Un client arrive à la caisse avec des articles à payer
  - Le caissier enregistre le numéro d'identification de chaque article, ainsi que la quantité si elle est supérieure à un
  - La caisse affiche le prix de chaque article et son libellé
  - Lorsque tous les achats sont enregistrés, le caissier signale la fin de la vente
  - La caisse affiche le total des achats
  - Le client choisit son mode de paiement : liquide (le caissier encaisse l'argent reçu, la caisse indique la monnaie à rendre au client), chèque , carte de crédit (la caisse transmet une demande d'autorisation à un centre d'autorisation)
  - La caisse enregistre la vente et imprime un ticket.
  - Le caissier donne le ticket de caisse au client.
- Lorsque un paiement est terminé, la caisse transmet les informations sur le nombre d'articles vendus au système de gestion de stocks.
- Tous les matins, le responsable du magasin initialise la caisse pour la journée.

3.1 Diagrammes d'objets

Concept d'objet

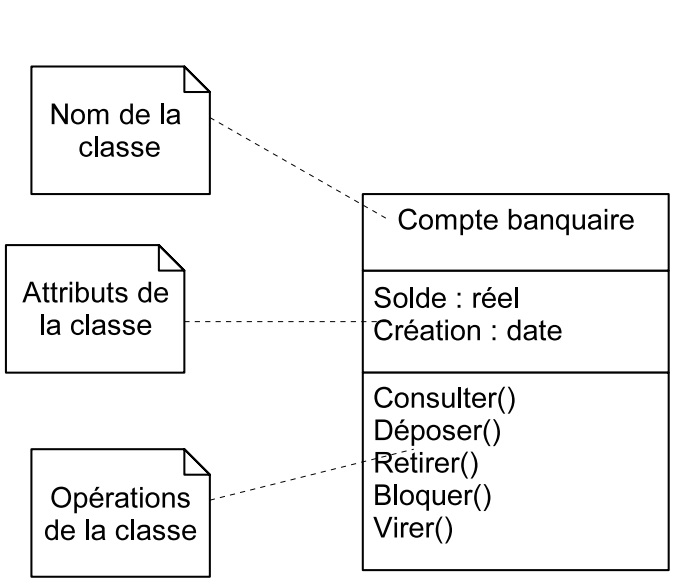
- Un objet est une entité atomique définie par l'union de :
  - Etat : attributs (données membres, champs)
  - Comportement : façon dont un objet évolue, décrit sous formes d'opérations (méthodes, fonctions membres)
  - Identité : propriété qui permet de distinguer tout objet des autres (référence en java, adresse en C++)
- Un objet peut représenter une entité physique (par ex. une voiture), virtuelle (par ex. un compte en banque) ou conceptuelle (par ex. un complexe)

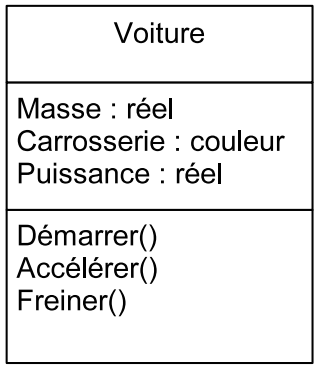


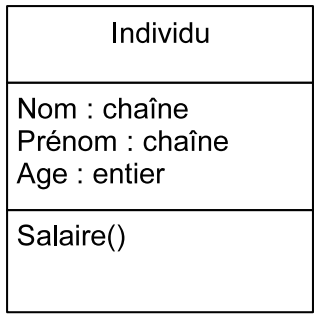
3.2 Diagrammes de classes

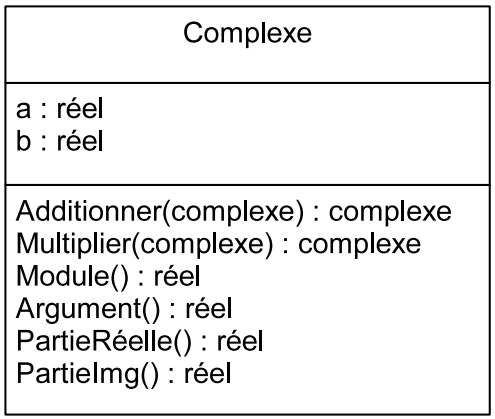
Concept de classe

- Notion de catégorie ou de type d'objets
- Factorise les caractéristiques communes à une catégorie d'objets
- Un objet est une *instance* d'une classe



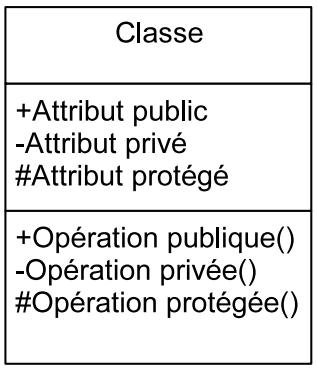


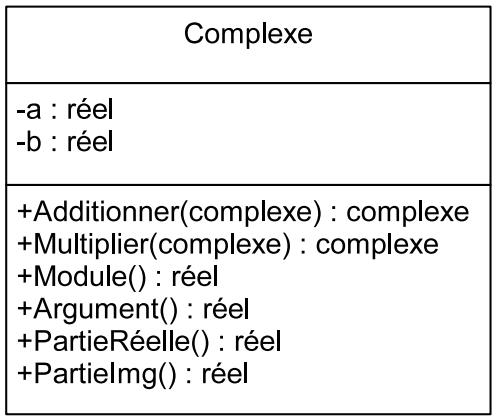


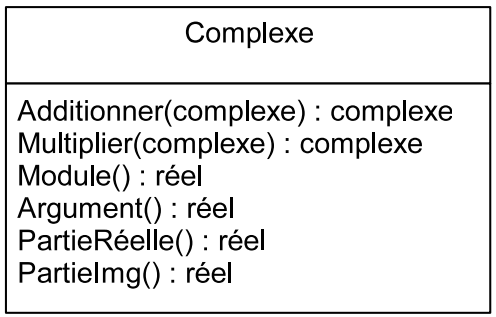


Encapsulation et visibilité des attributs et des opérations

- Intérêt de l'encapsulation
  - Permet de cacher les détails d'implantation des attributs d'un objet
  - Empêche un client de modifier directement l'état d'un objet (i.e. ses attributs), obligation d'utiliser les opérations
- 3 niveaux
  - *Public* : attributs accessibles et modifiables par n'importe quelle autre classe, opérations utilisables par n'importe quelle autre classe,
  - *Privé* : attributs inaccessibles pour toute autre classe, opérations inutilisables pour toute autre classe,
  - *Protégé* : attributs accessibles et modifiables uniquement par les classes dérivées, opérations utilisables uniquement par les classes dérivées.

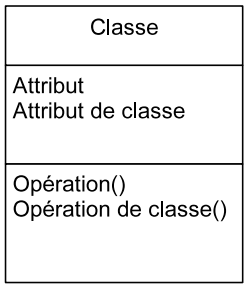


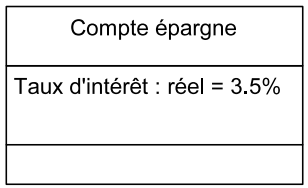


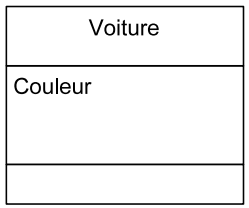


Attributs de classe et classes génériques

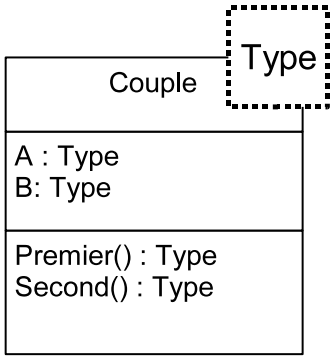
- Attributs (ou opérations) communs à l'ensemble des instances d'une classe (attributs *static* en C++)

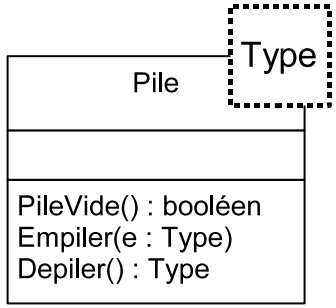






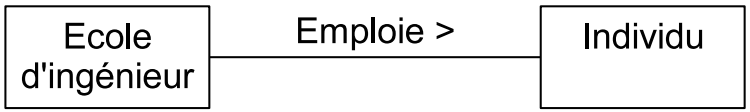
- Classes génériques avec type paramétrable (classes *template* en C++)

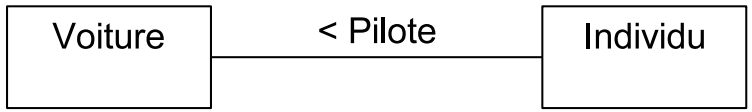




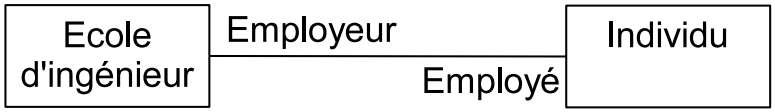
Associations de classes

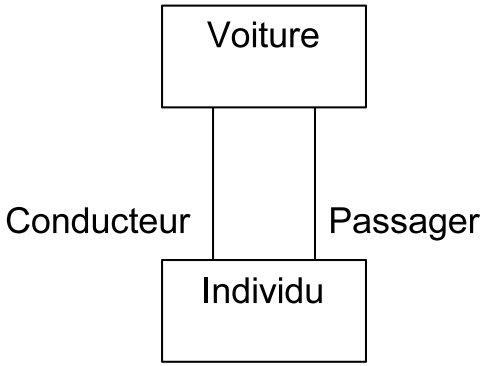
- Reflet d'une relation bidirectionnelle entre deux classes (ou plus)
- Pour pouvoir envoyer des messages (i.e. utiliser les opérations), la classe cliente doit être associée avec la classe serveur

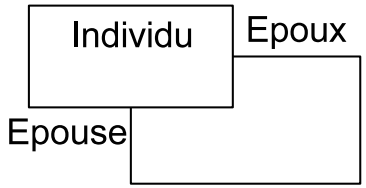




- Possibilité de décrire les rôles

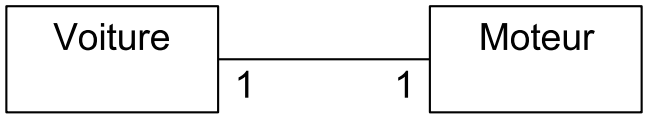


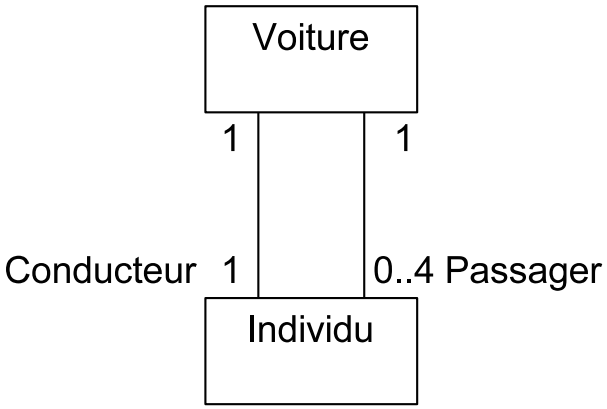


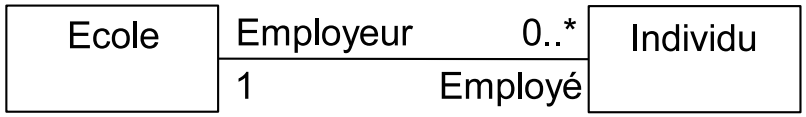


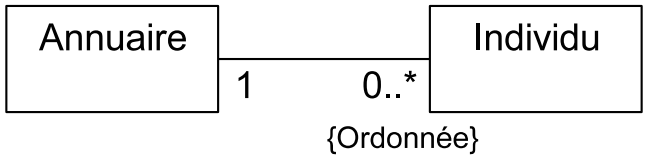
Associations de classes

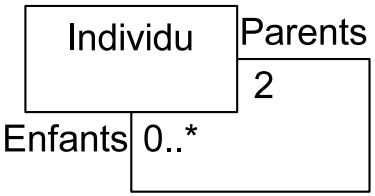
- Multiplicité des associations





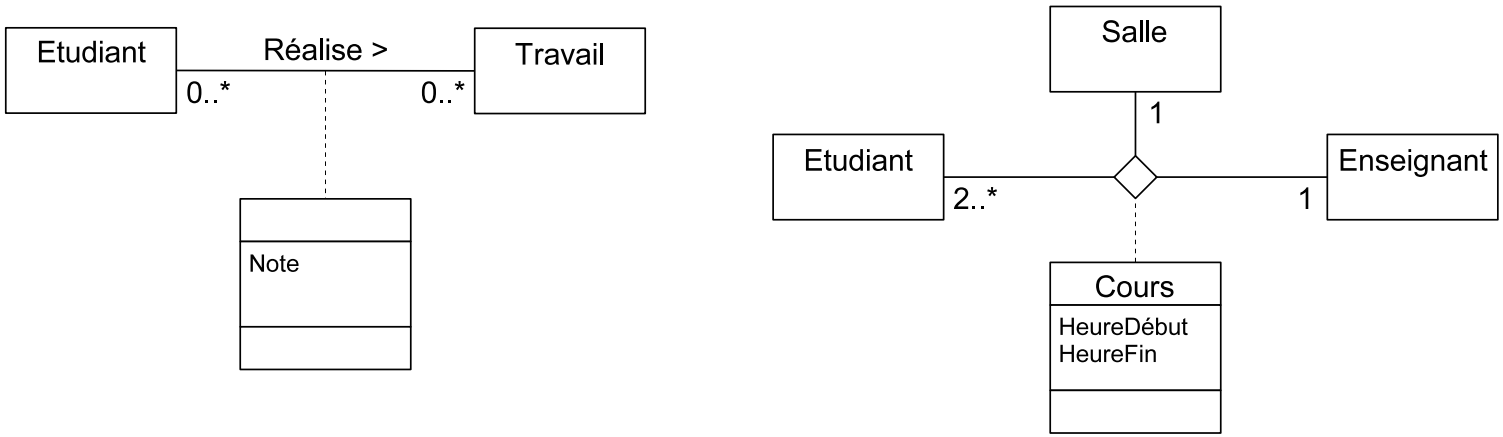




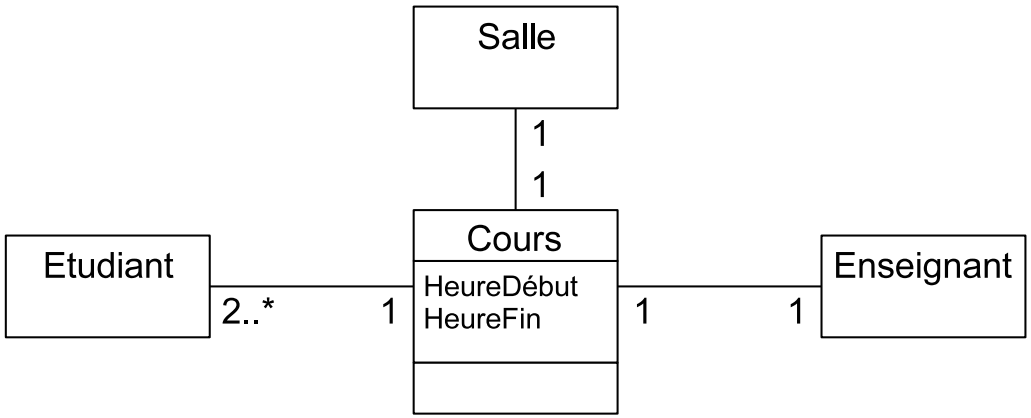


Classes d'association

- Une association peut être représentée par une classe (pour être enrichie par exemple)

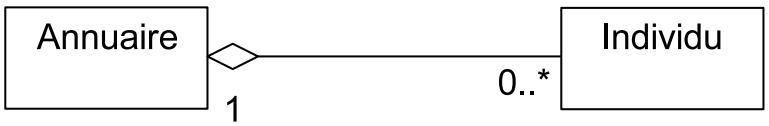


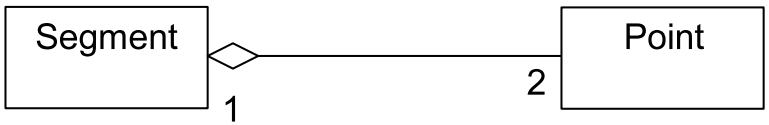
- Les associations n-aires peuvent souvent être réduites

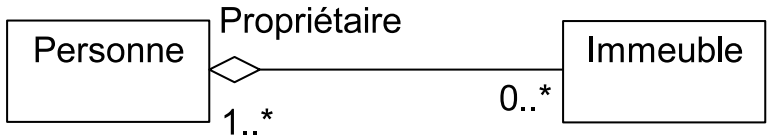


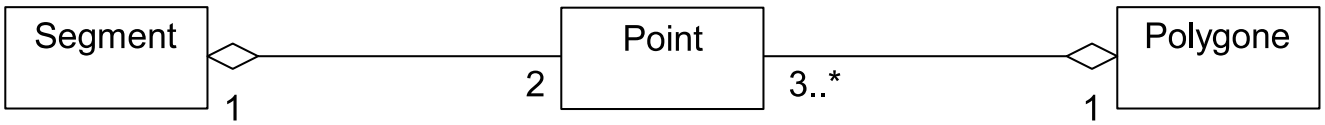
Agrégations de classes

- L'agrégation est une association non symétrique :
  - Relation de dominance et de subordination
  - Une classe fait partie d'une autre classe
  - Une action sur une classe implique une action sur une autre classe
- Une classe peut appartenir à plusieurs agrégats



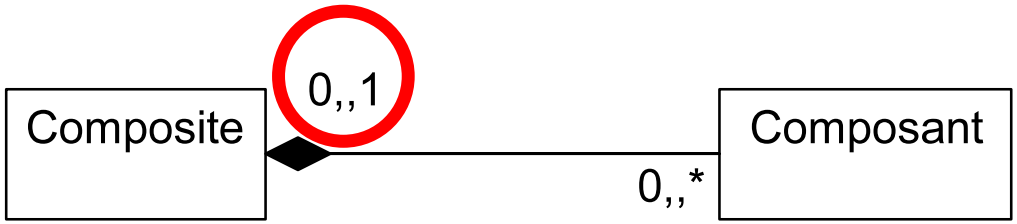


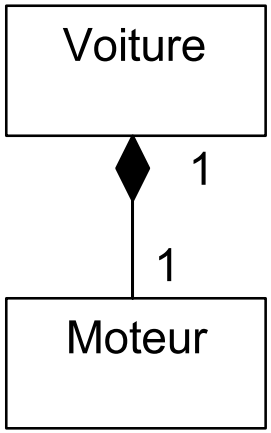




Compositions de classes

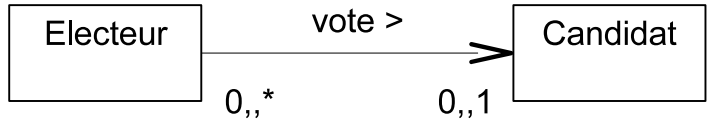
- La composition est une agrégation *forte*
- Relation d'appartenance stricte équivalente à l'attribution
- Une classe ne peut pas appartenir à plusieurs compositions
- Une classe composée ne peut exister sans son propriétaire



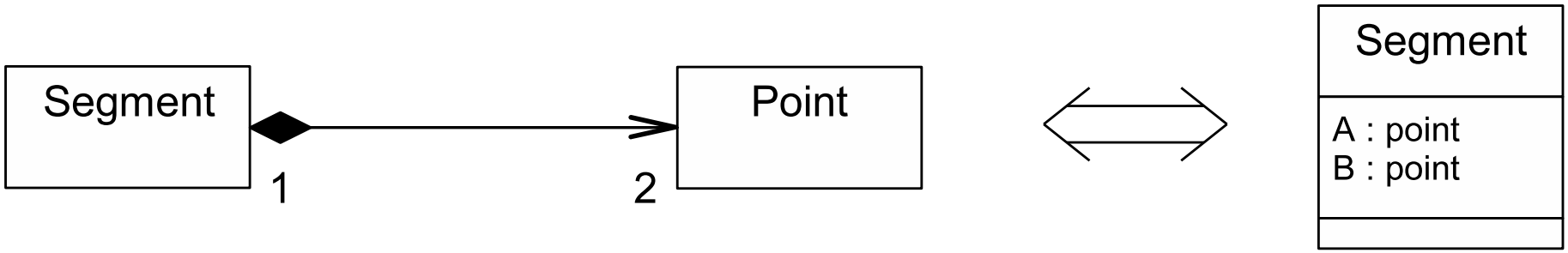


Navigation

- Par défaut, les associations sont navigables dans les deux directions
- Une association peut ne nécessiter qu'un seul sens de navigation :

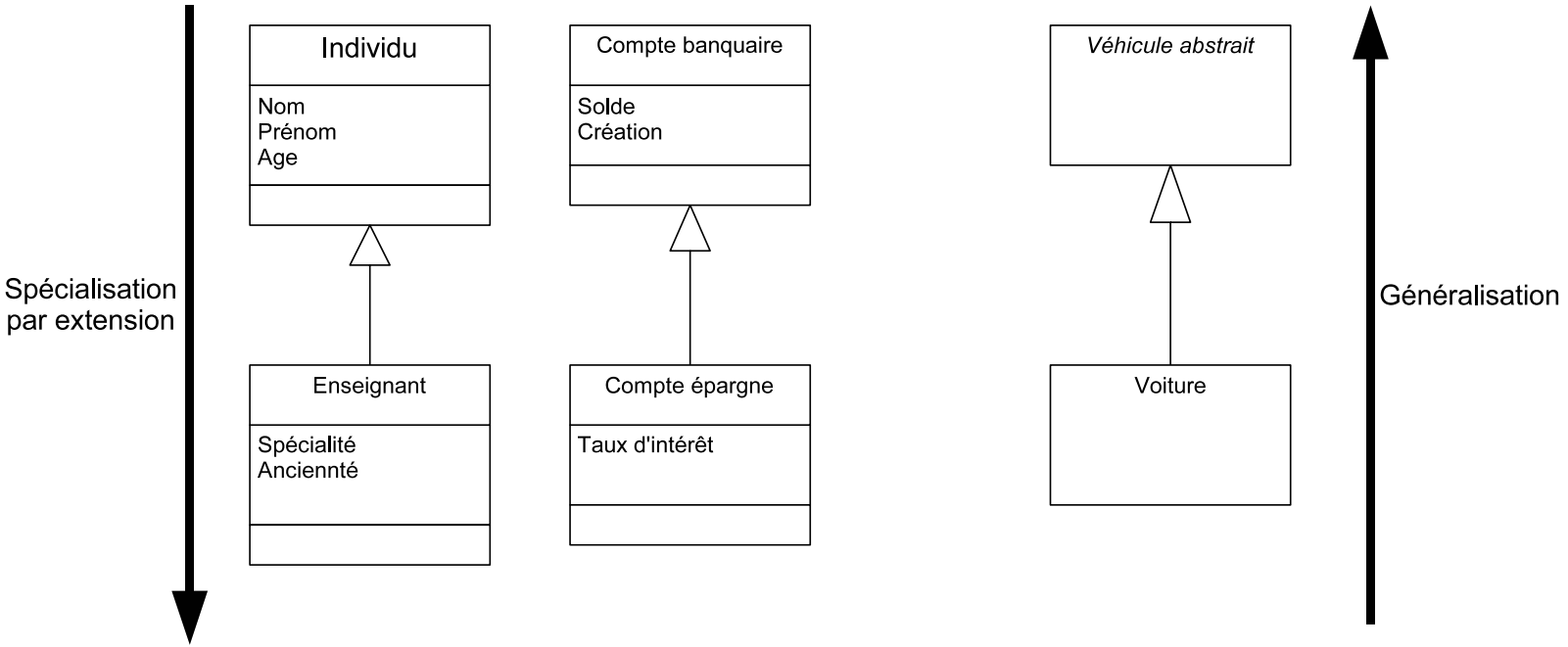


- B ne ≪ voit ≫ pas A
- B ne peut envoyer de message à A



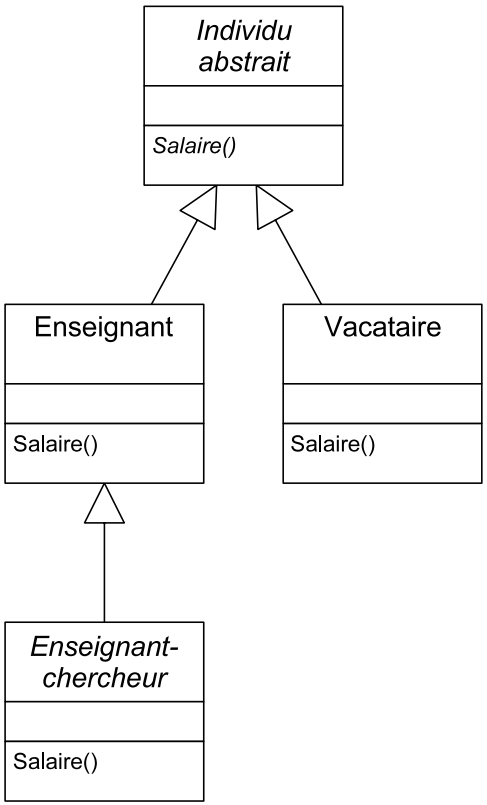
Généralisation/spécialisation

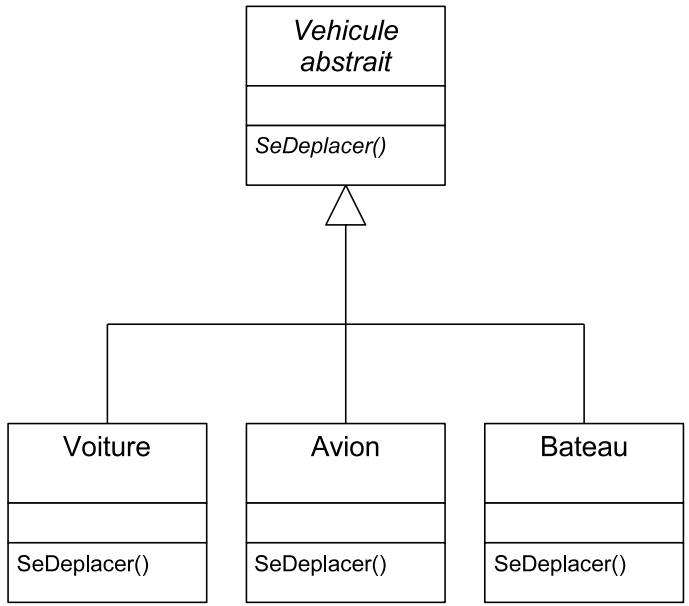
- Spécialisation de la classe existante par ajout d'attributs et/ou ajout/redéfinition de méthodes
- Héritage, dérivation par extension
- Relation « EST UN »



Polymorphisme

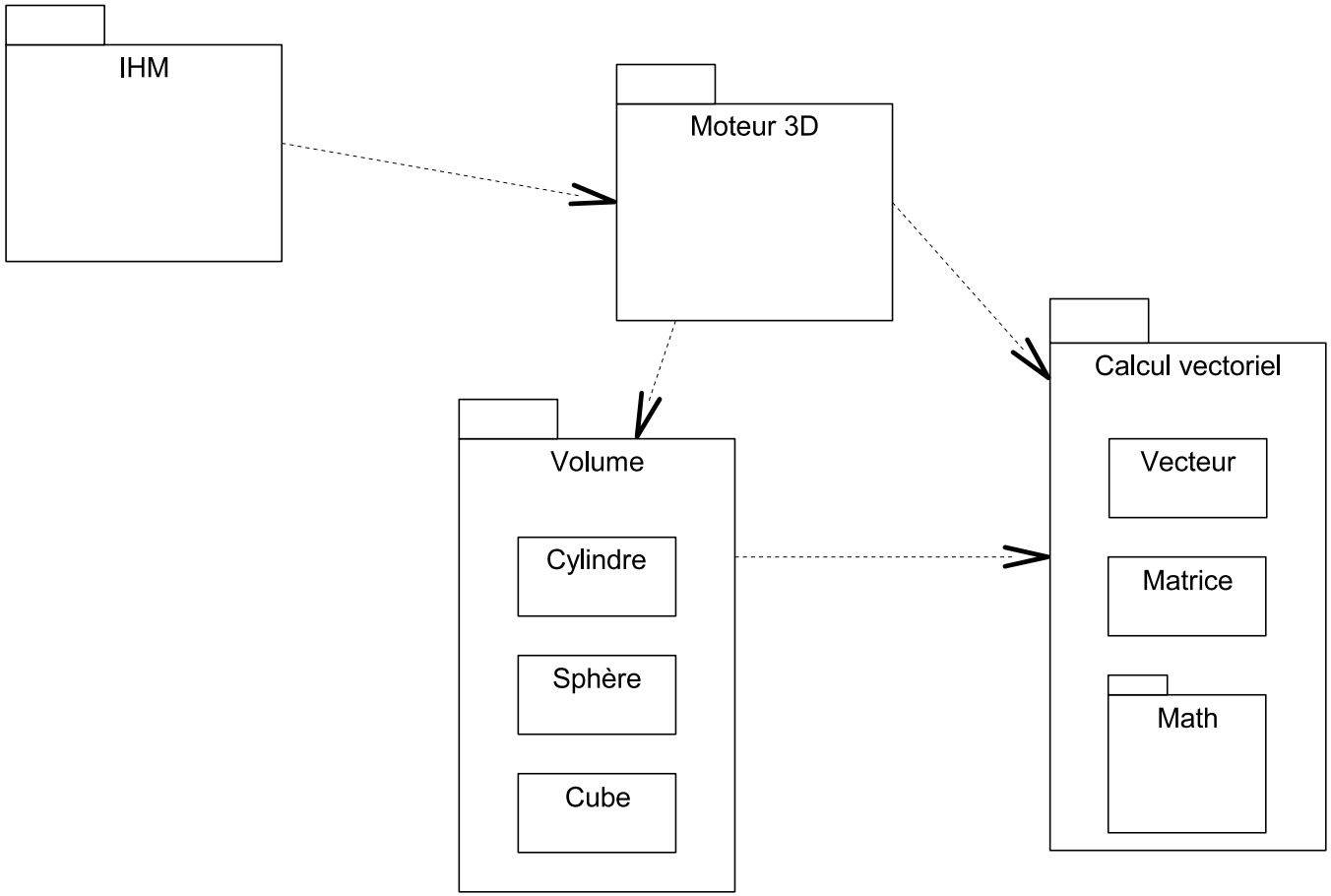
- Permet d'utiliser un nom unique pour désigner des instances de classes différentes issues d'une même arborescence
- Chaque objet désigné par ce nom réagit de manière propre à la réception d'un même message





Notion de paquetages (package)

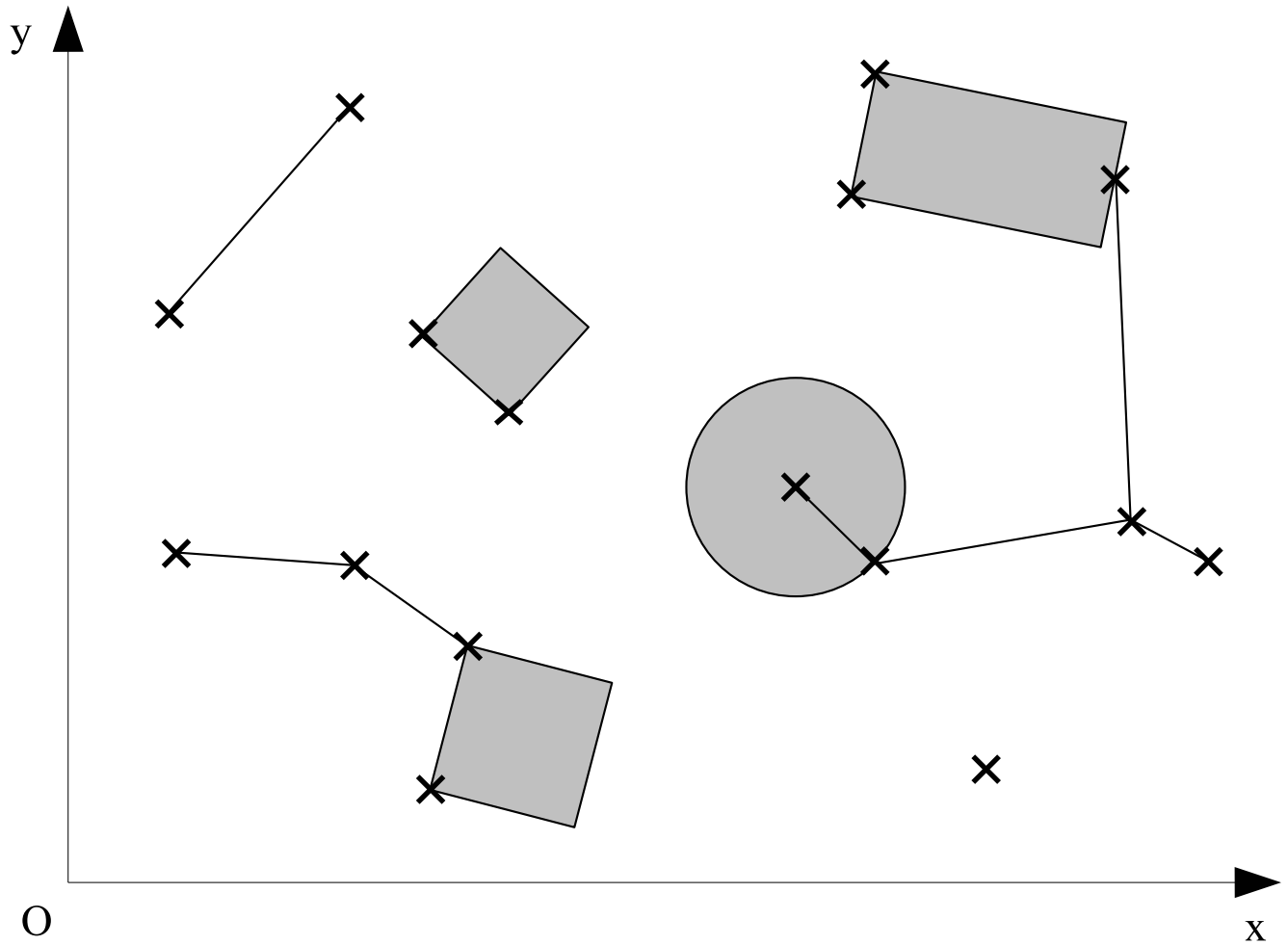
- Regrouper les classes (ou autres diagrammes) liées entre elles
- Faire apparaître (et minimiser !) les dépendances entre paquetages



Exercice

Ex.1 Soit le dessin géométrique ci-dessous :

- Modéliser cette situation par un diagramme d'objets
- Modéliser un dessin géométrique par un diagramme de classes



Exercice

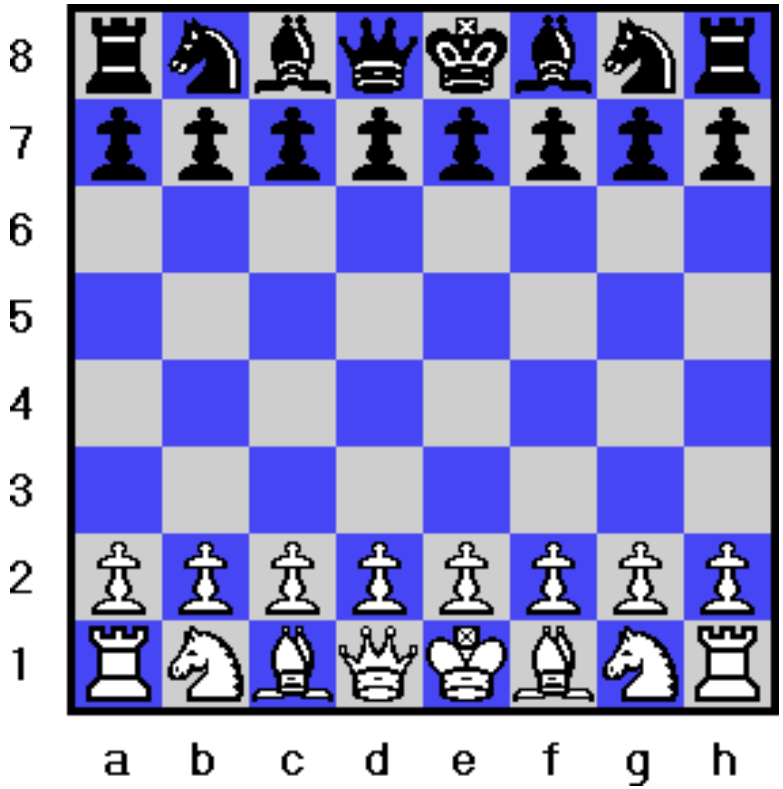
Ex.2 Modéliser les phrases suivantes par un diagramme de classes

- Un répertoire contient des fichiers
- Une pièce a des murs
- Une transaction bancaire est un achat ou une vente
- Une enveloppe a une adresse et un timbre
- Les lapins, les hirondelles, les lions, les requins, les autruches, les vautours, les truites, les pingouins sont des animaux qui se déplacent de manières différentes.
- Des poupées russes (gigognes)
- Un pays possède plusieurs villes et une seule capitale

Exercice

Ex.3 Modéliser la structure statique d'un logiciel de jeu d'échecs en se focalisant successivement sur :

- L'échiquier et les pièces
- Les joueurs et leurs pièces
- Une partie

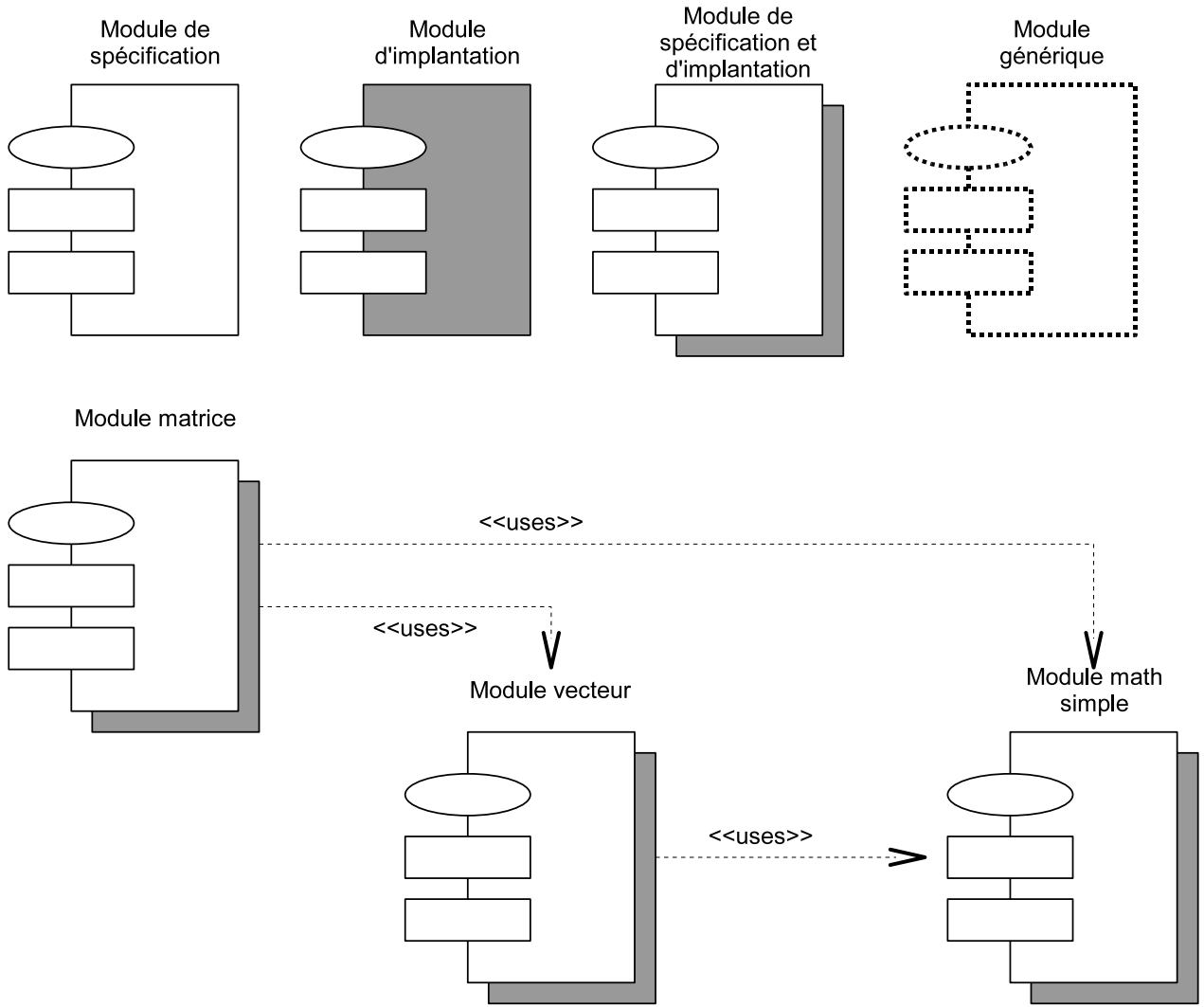


(extrait de *UML par la pratique*, Pascal Roques, Eyrolles)

3.3 Diagrammes de composants

Diagrammes de composants

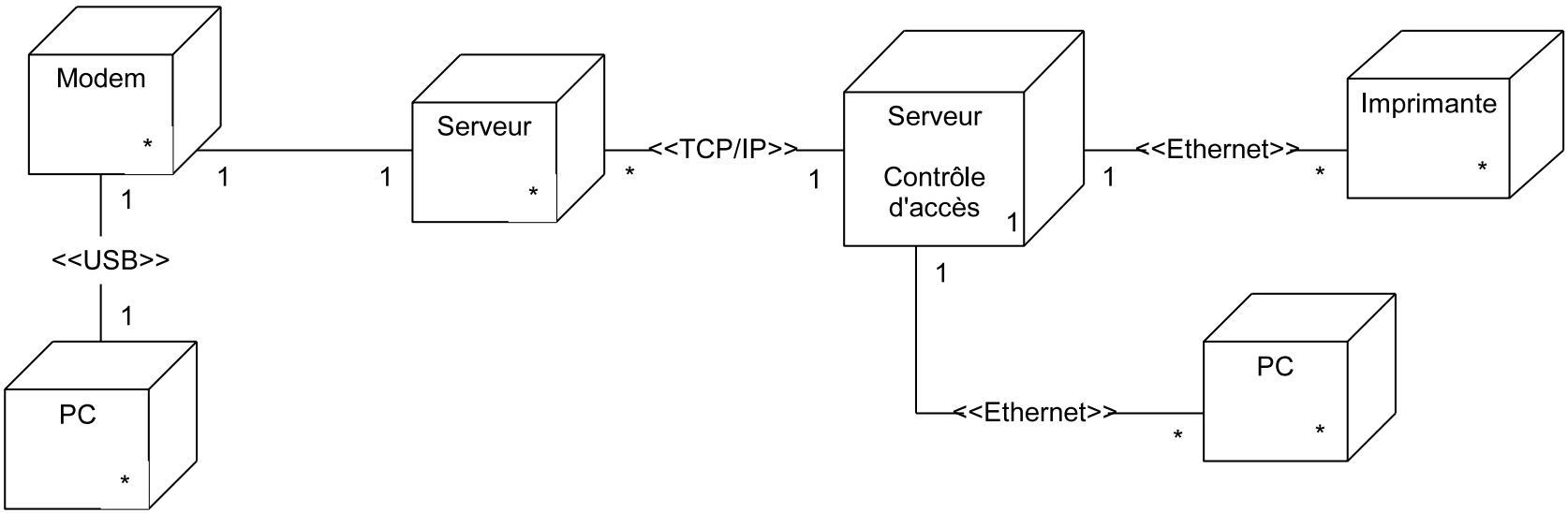
- Permet de décrire l'architecture statique d'une application en terme de modules (fichiers sources, fichiers compilés, etc.)
- Les dépendances entre modules permettent d'identifier les contraintes de compilation et de mettre en évidence la réutilisation des composants



3.4 Diagrammes de déploiement

Diagrammes de déploiement

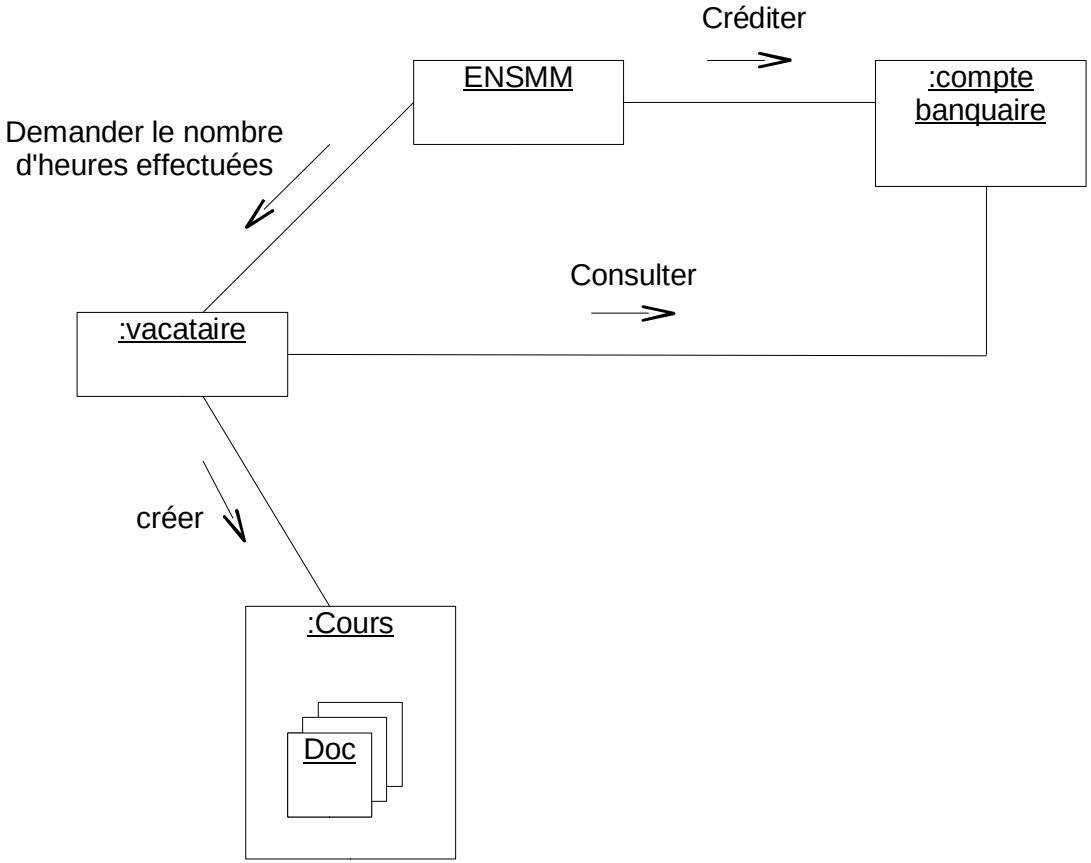
- Montre la disposition physique des différentes ressources matérielles (noeuds), leurs interconnexions et la répartition des programmes sur ces matériels
- Un système est généralement décrit par un petit nombre de diagrammes de déploiement (généralement un seul suffit)



4.1 Diagrammes de collaboration

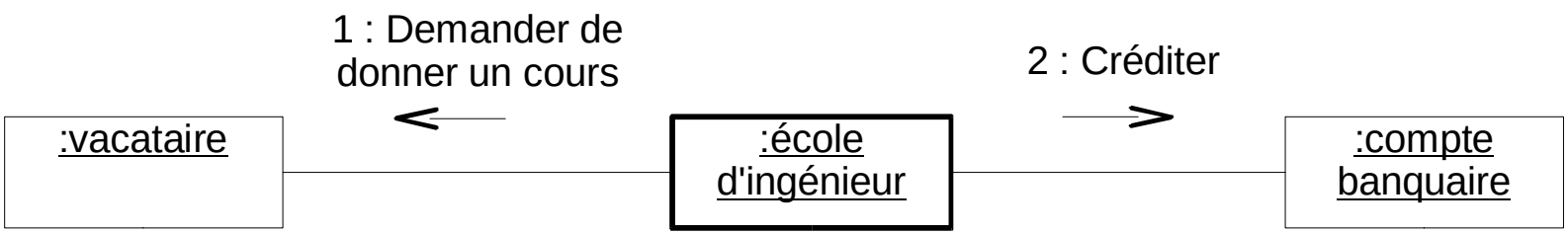
Concept de message

- Message : quand un objet (le client) demande à un autre objet (le serveur) d'effectuer une de ses opérations, on parle de message



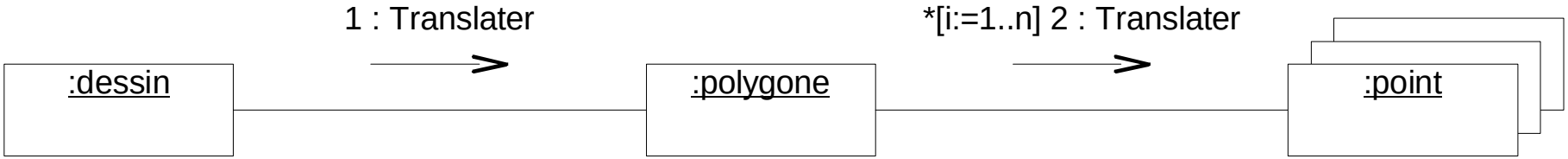
Diagrammes de collaboration

- Représentent les interactions (échanges de messages) entre objets selon un point de vue spatial
- Extension des diagrammes d'objets par l'ajout de la représentation des messages échangés
- Vues généralement partielles et focalisées sur une collaboration de quelques objets
- Les messages sont numérotés pour indiquer l'ordre des envois

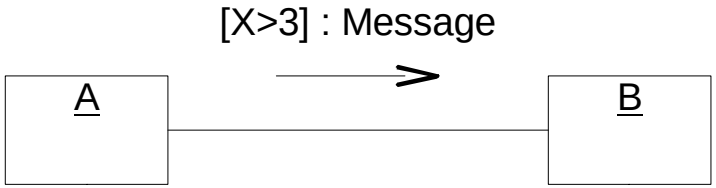


Diagrammes de collaboration

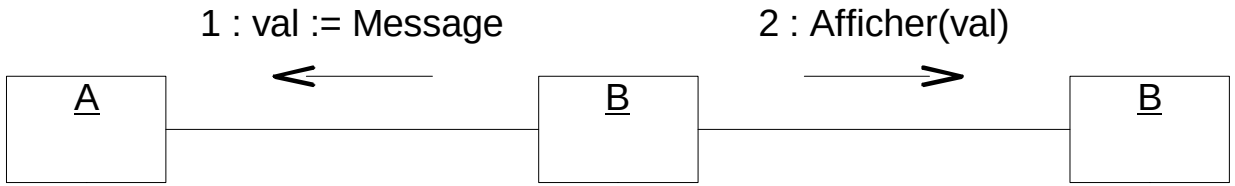
- Itérations



- Messages conditionnels

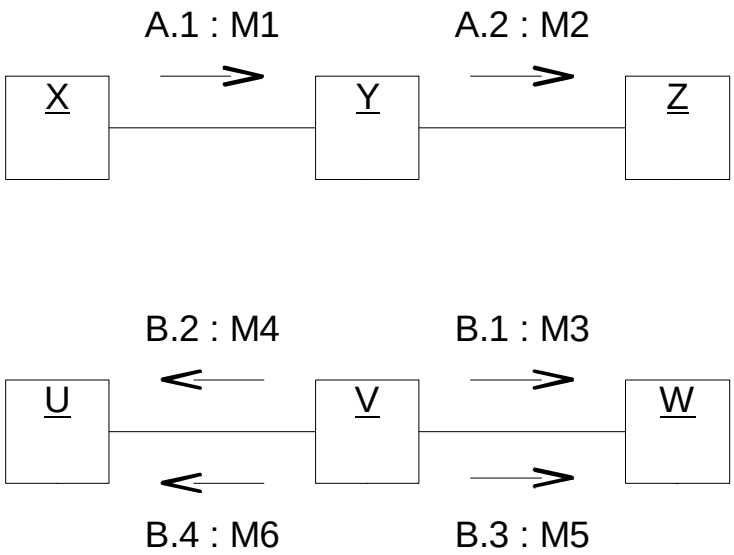


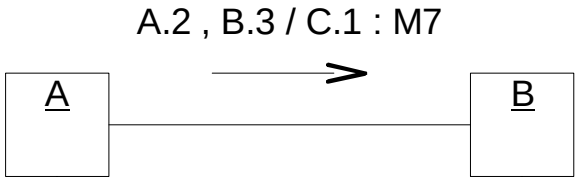
- Résultats et paramètres (ou arguments)



Diagrammes de collaboration

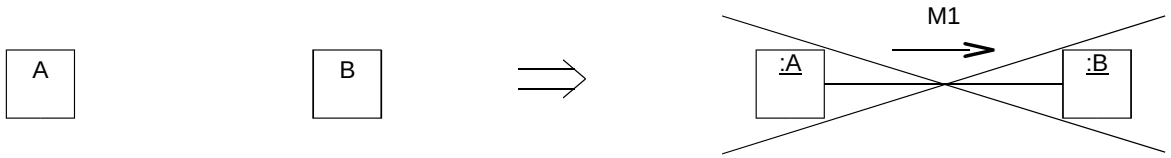
- Exécutions parallèles et synchronisation



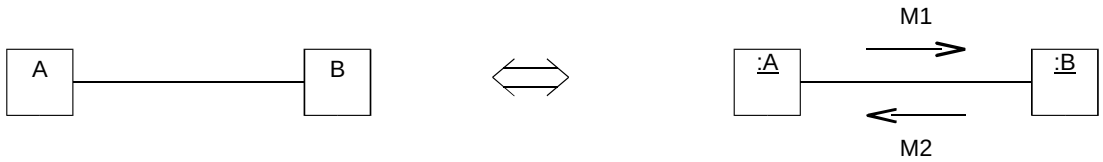


Diagrammes de collaboration

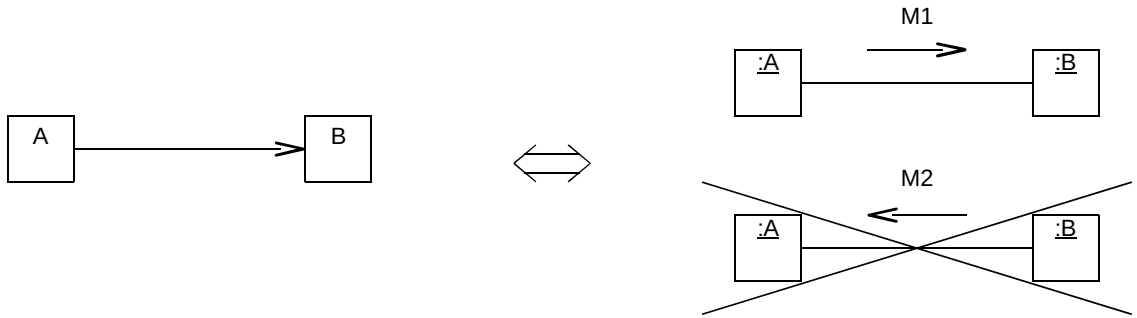
- Deux objets peuvent échanger un message ssi les classes de ces objets sont en relation (association, agrégation ou composition)



- Le sens des messages entre deux objets est bidirectionnel ssi aucun sens de navigation n'est précisé

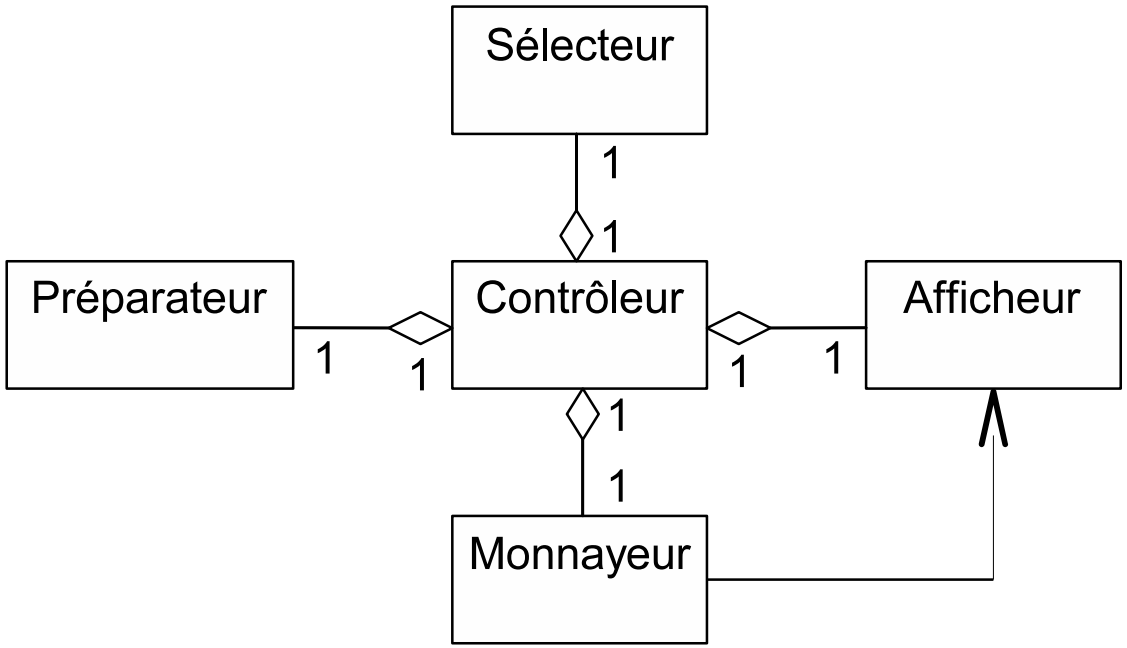


- Le sens des messages entre deux objets est unidirectionnel ssi un sens de navigation est précisé



Exercice

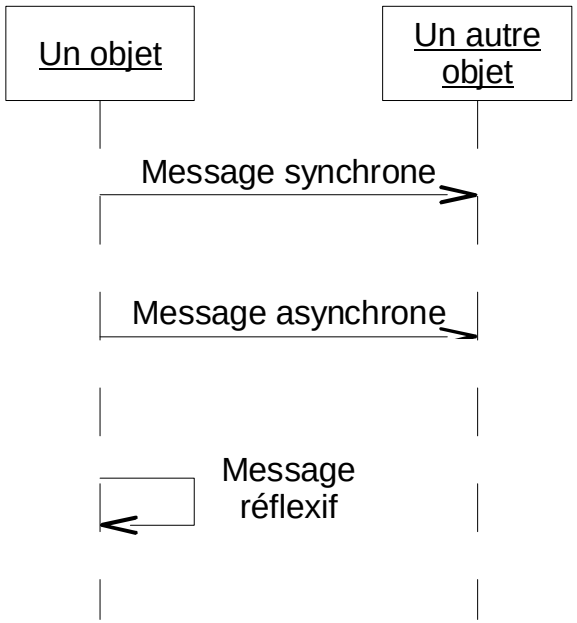
Ex.1 Décrire à l'aide d'un diagramme de collaboration l'enchaînement nominal de distribution d'une boisson pour un distributeur de boissons dont la structure est définie par le diagramme de classes ci-dessous

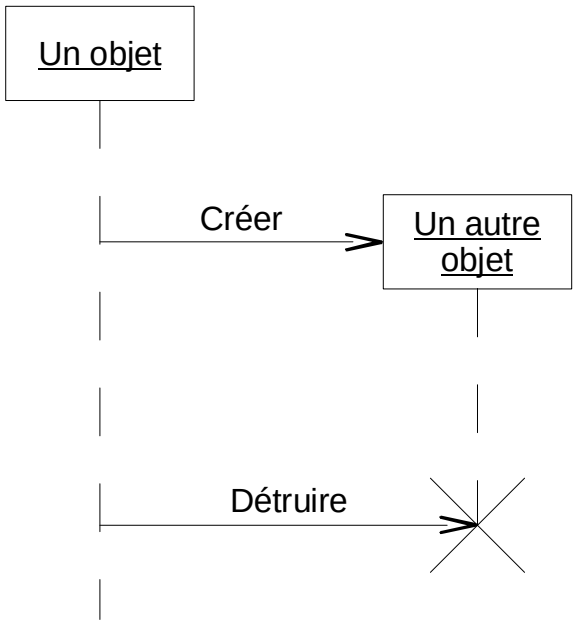


4.2 Diagrammes de séquence

Diagrammes de séquence

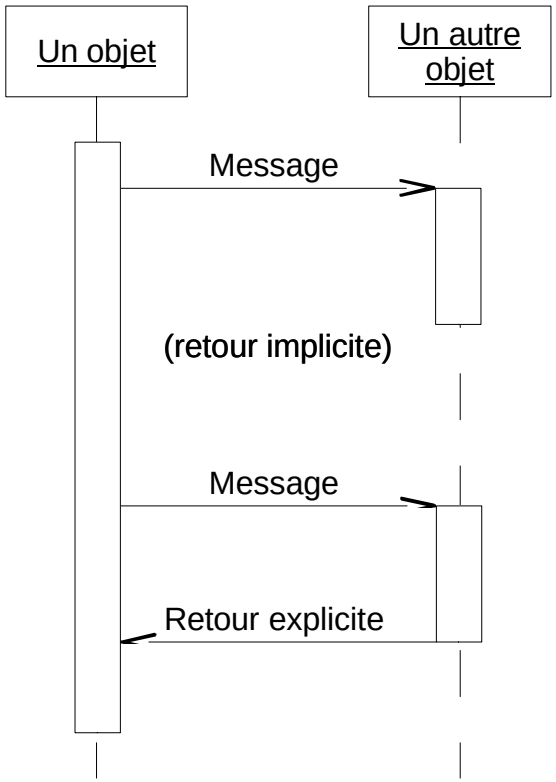
- Représentent les interactions (échanges de messages) entre objets selon un point de vue temporel
- Le contexte des objets n'est pas représenté

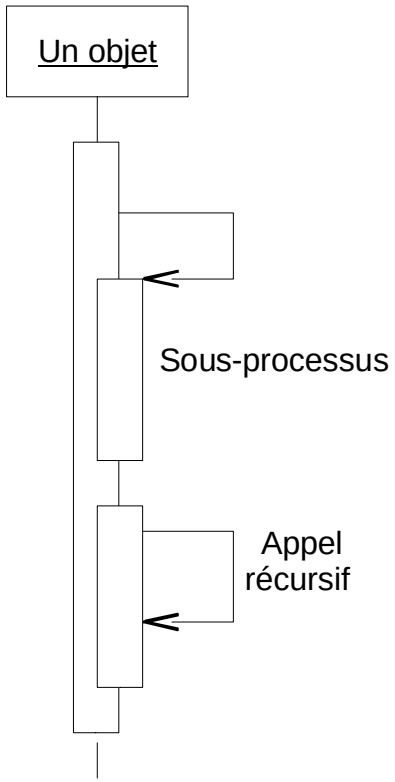




Diagrammes de séquence

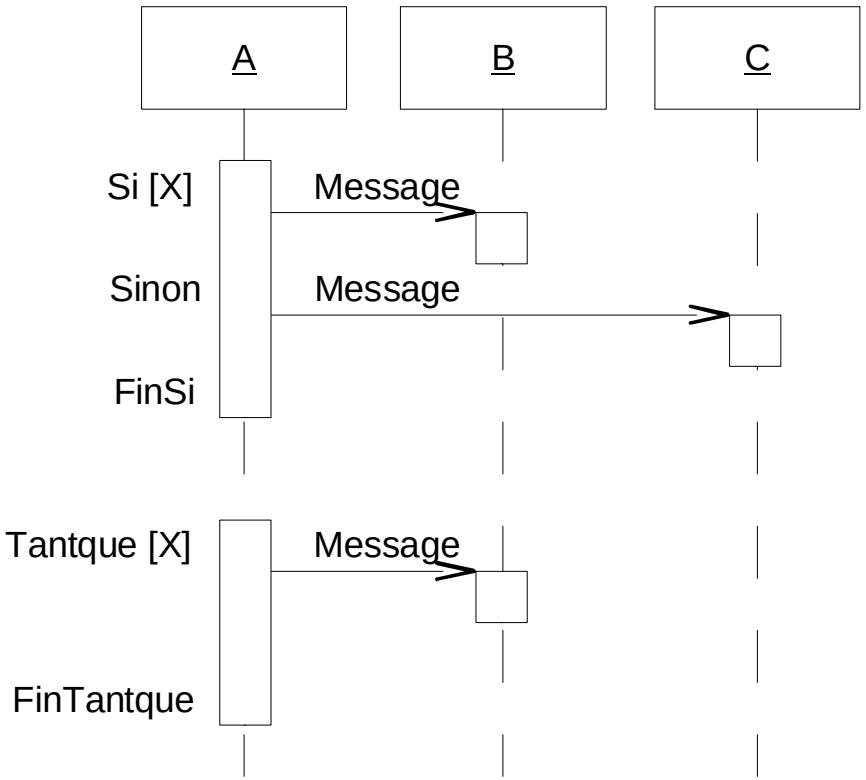
- Activation

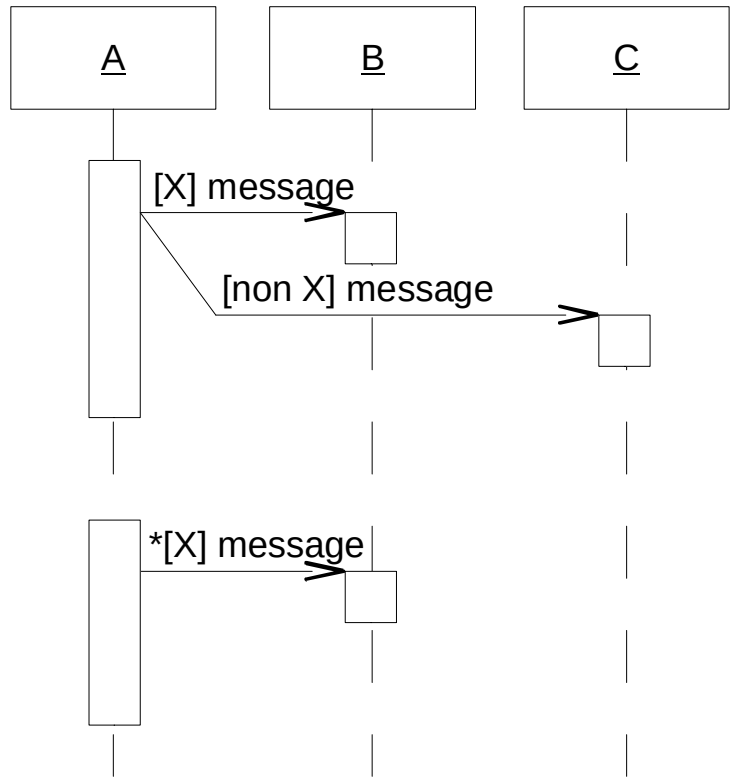




Diagrammes de séquence

- Structures de contrôles

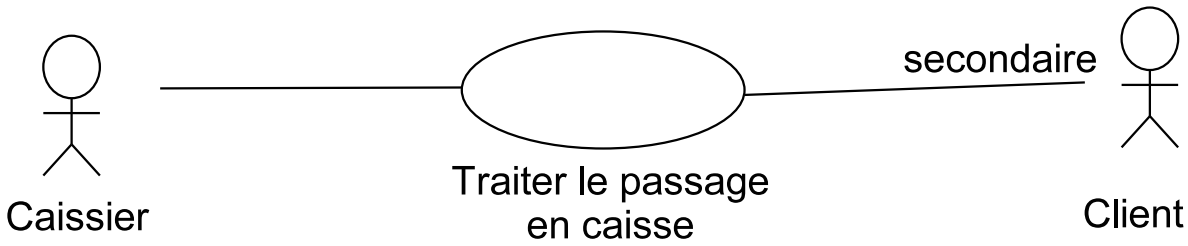




Exercices

Ex.1 Réaliser le diagramme de séquence correspondant au diagramme de collaboration du distributeur de boissons (cf. T41)

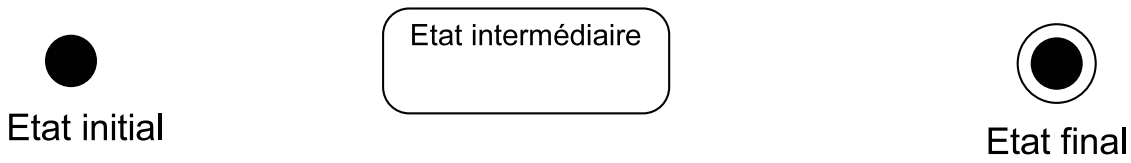
Ex.2 Réaliser le diagramme de séquence système du cas d'utilisation « traiter le passage en caisse » de l'exercice sur la caisse enregistreuse (cf. T12)



4.3 Diagrammes d'états-transitions

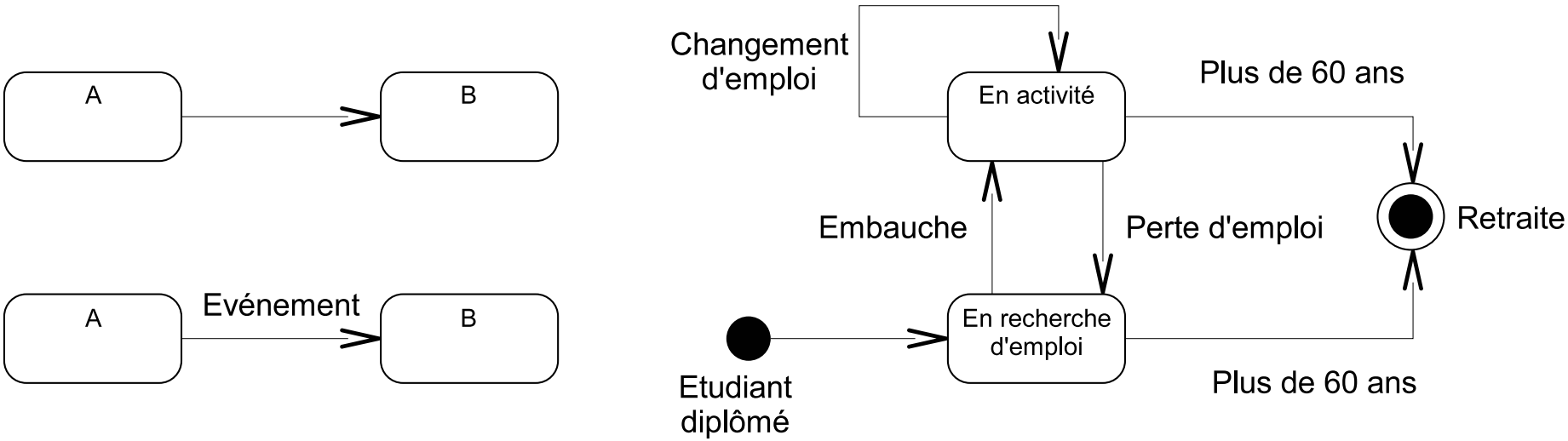
Diagrammes d'états-transitions

- Permettent de modéliser des automates d'états déterministes
- Représentent le comportement *interne d'un objet* (évolution de son état), en réponse aux interactions avec d'autres objets
- Représentation des états :

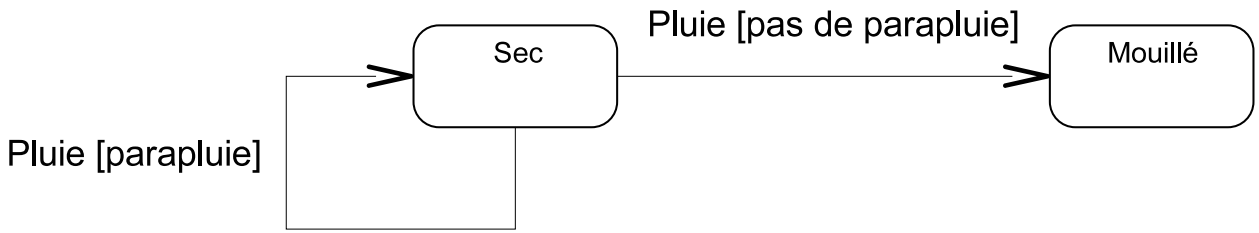


Diagrammes d'états-transitions

- Transitions et événements

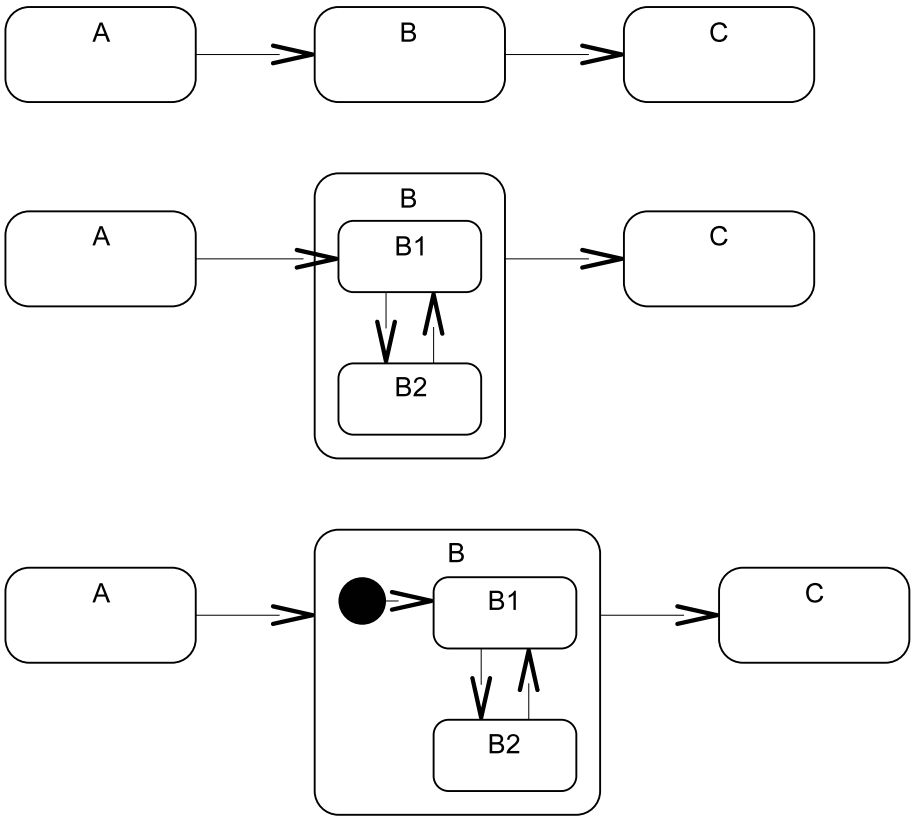


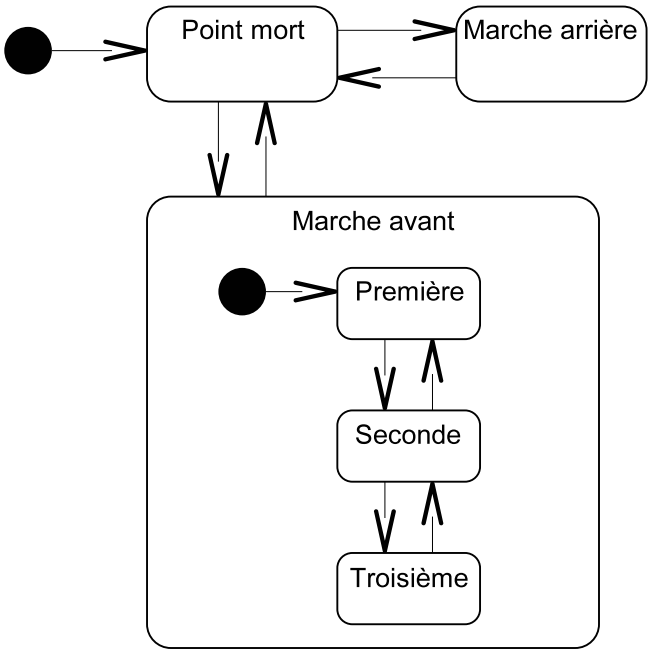
- Gardes (événements conditionnels)



Diagrammes d'états-transitions

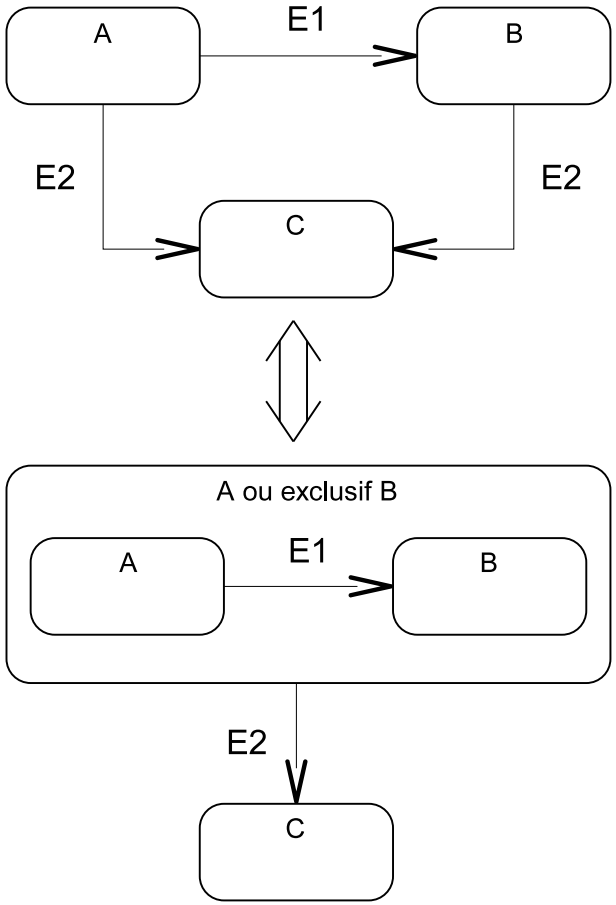
- Généralisation d'états
  - Super-états décomposés en plusieurs états disjoints (décomposition disjonctive)
  - Même démarche que la spécialisation/généralisation des classes, les sous-états héritent des caractéristiques de leur super-état

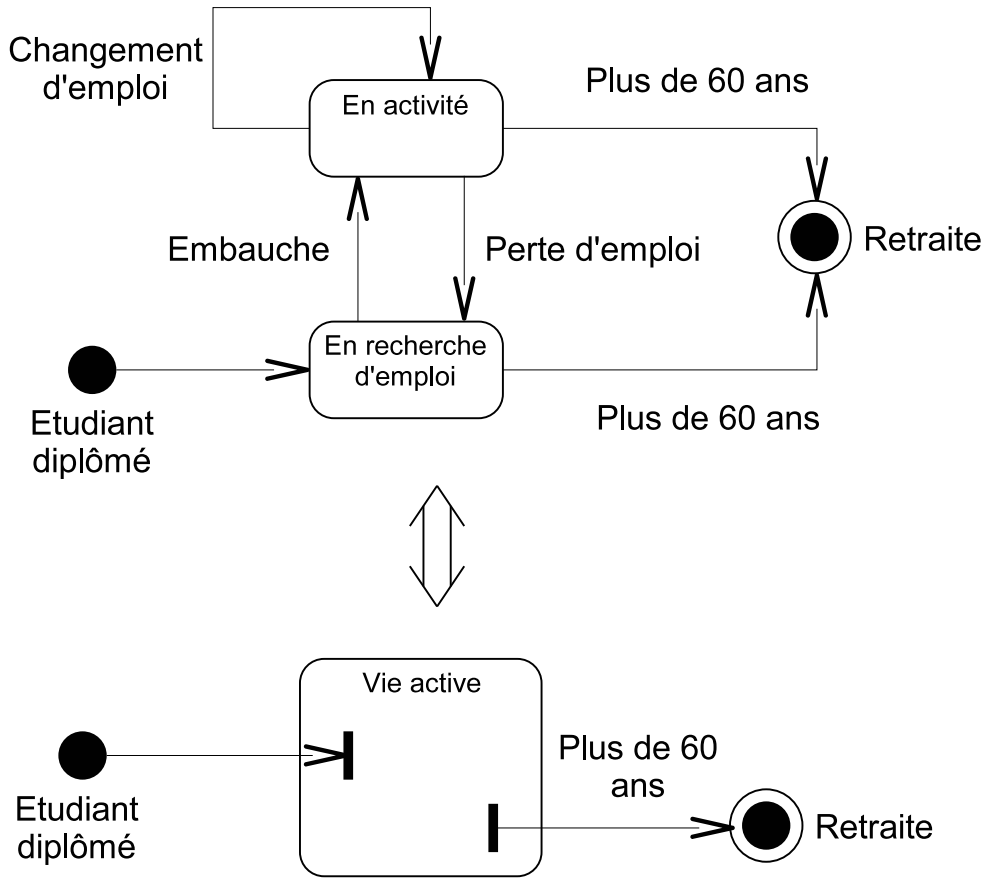




Diagrammes d'états-transitions

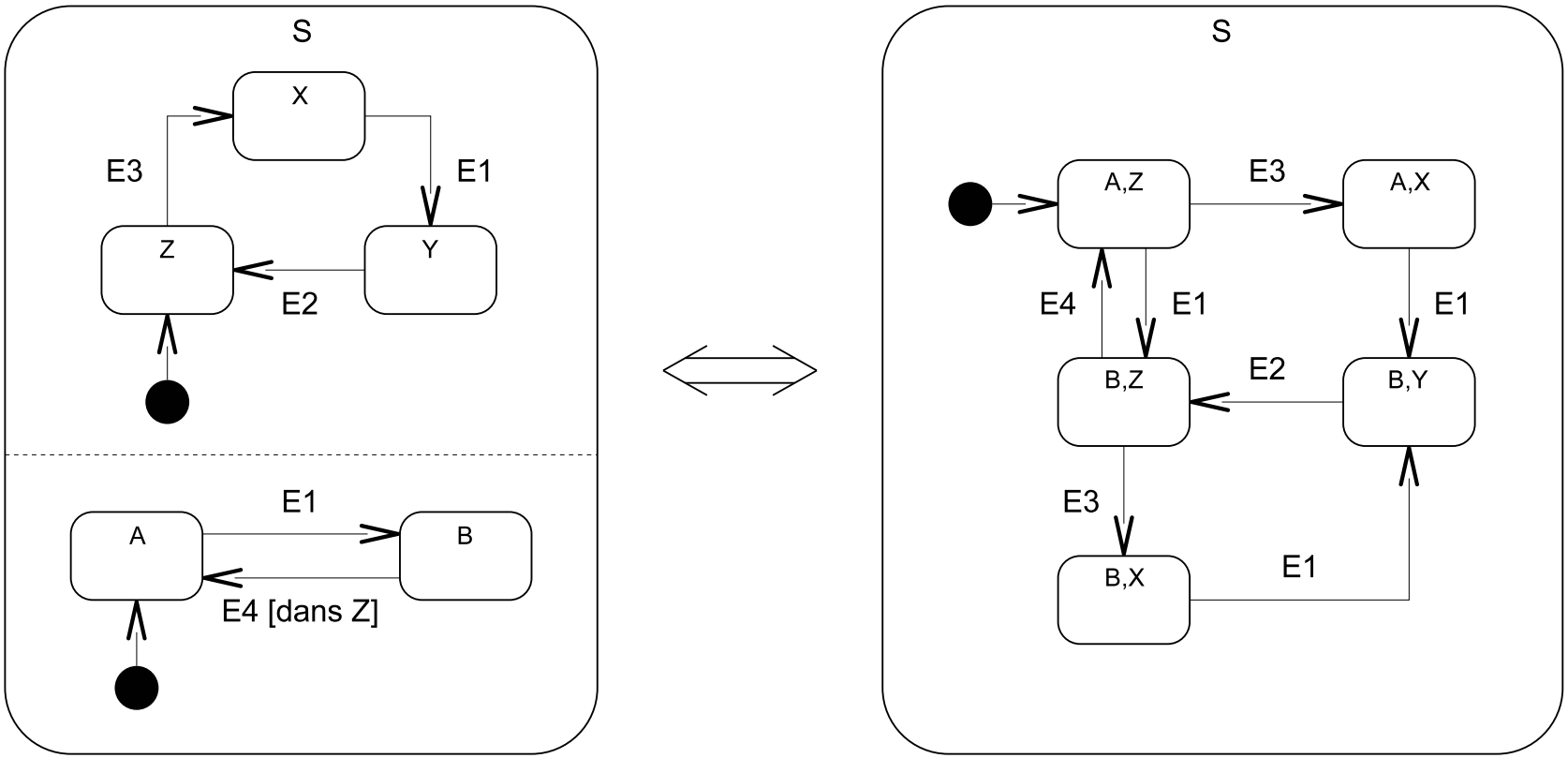
- Généralisation d'états
  - Facilite la représentation et permet d'occulter les détails





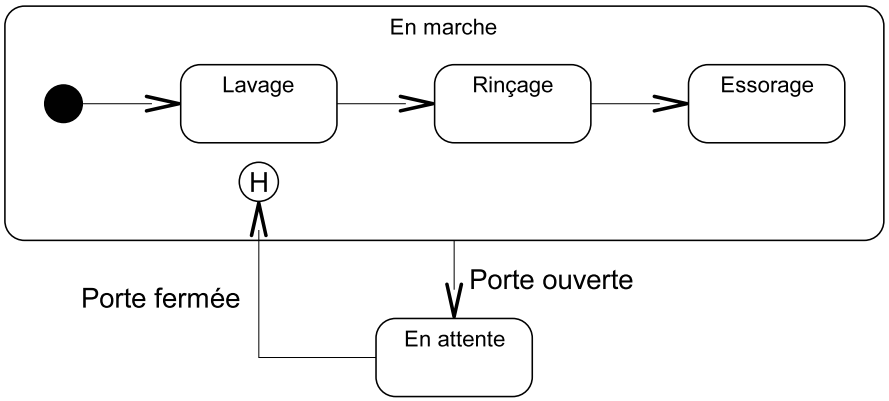
Diagrammes d'états-transitions

- Agrégation d'états
  - Composition d'un état à partir de plusieurs autres états indépendants (composition conjonctive)
  - Permet de décrire des automates parallèles

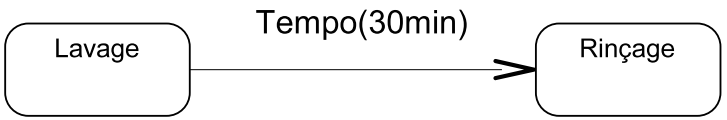


Diagrammes d'états-transitions

- Historique
  - Mécanisme pour mémoriser le dernier sous-état visité
  - H* indique une mémorisation pour tous les états imbriqués

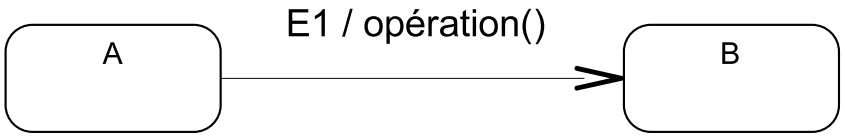


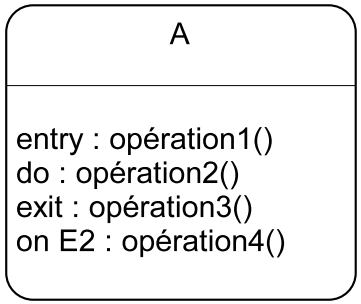
- Déclenchement d'un événement au bout d'un certain temps passé dans un état (transition temporisée)



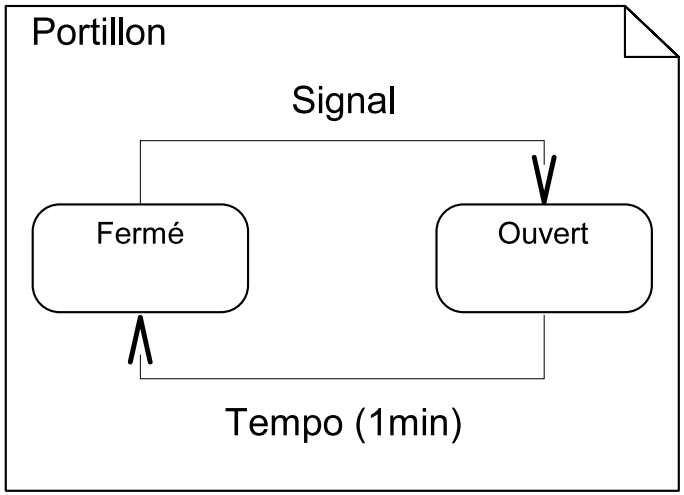
Diagrammes d'états-transitions

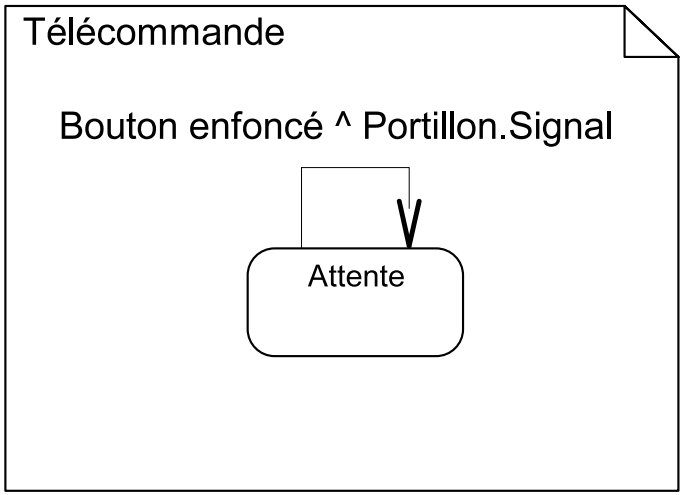
- Déclenchements d'opérations





- Déclenchements d'événements relatifs à un autre objet





Exercices

Ex.1 Modéliser le fonctionnement d'un distributeur de boissons à l'aide d'un diagramme d'états-transitions

Ex.2 Modéliser le fonctionnement d'un téléphone public à pièces à l'aide d'un diagramme d'états-transitions

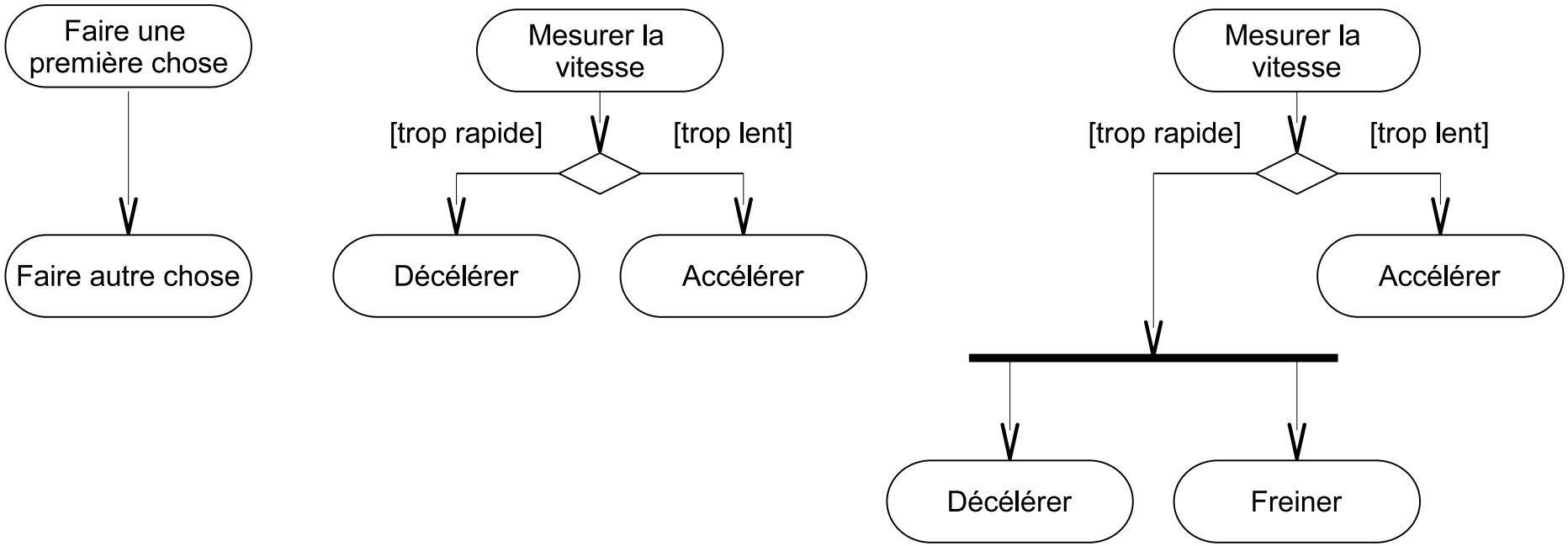
- Le prix minimal d'une communication est de 0,2 euros
- Après l'introduction de la monnaie, l'utilisateur a 2 minutes pour composer son numéro
- La ligne peut être libre ou occupée
- Le correspondant peut raccrocher le premier
- Le téléphone consomme de l'argent dès que l'appelé décroche et à chaque unité de temps
- On peut ajouter des pièces à tout moment
- Lorsque l'on raccroche, le solde de monnaie est rendu

(extrait de *UML par la pratique*, Pascal Roques, Eyrolles)

4.4 Diagrammes d'activités

Diagrammes d'activités

- Représentent le comportement *interne d'une opération*

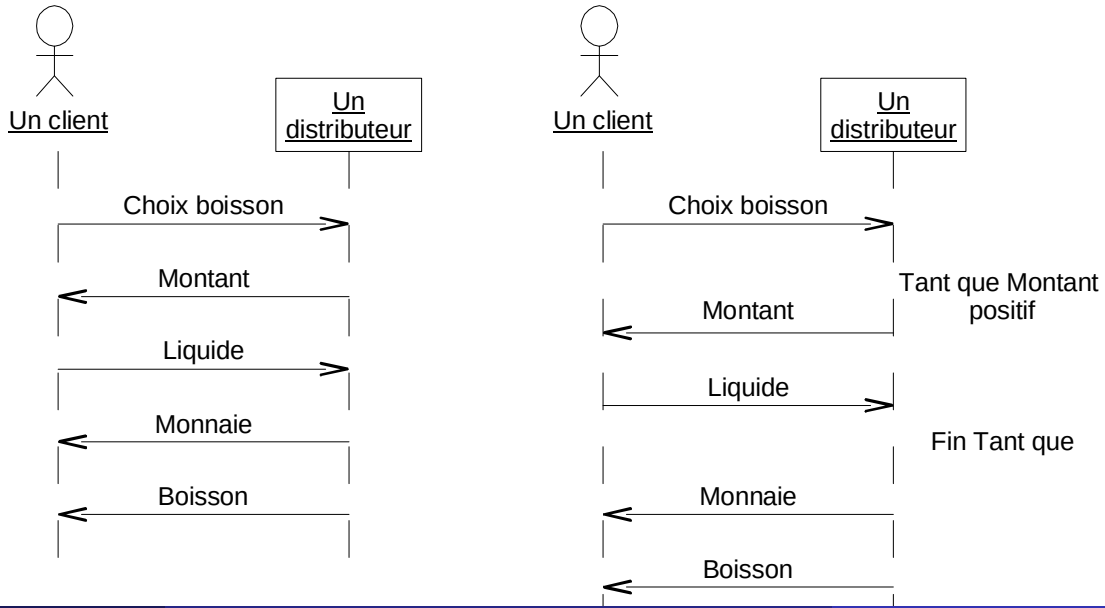


Exercice

Ex.1 Modéliser le comportement de l'opération « PréparerBoisson() » de l'objet « Préparateur » d'un distributeur de boissons

Capture des besoins – Cahier des charges

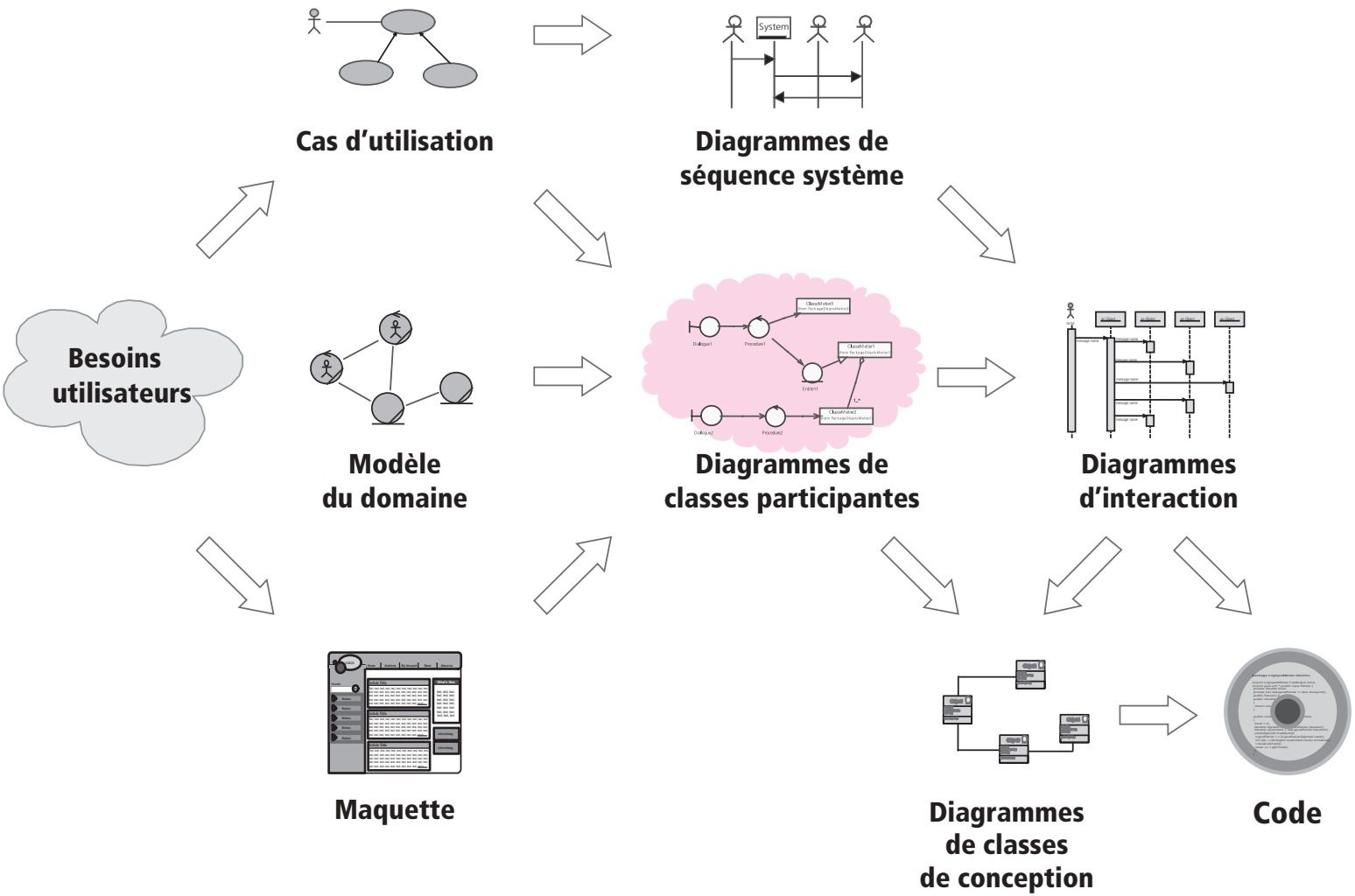
- Diagrammes de cas d'utilisation
- Utilisation de fiches textuelles (non UML)
  - Identification (titre du cas, acteurs concernés, responsable, date, etc.)
  - Préconditions
  - Enchaînements nominaux
  - Enchaînements alternatifs
  - Enchaînements d'exceptions
  - Postconditions
- Utilisation de diagrammes de séquence système (diagrammes de séquence simplifiés)



Démarche de modélisation objet

- Ordonner les cas d'utilisation
- Identifier les classes principales à partir des spécifications
- Élaborer des diagrammes de classes
- Faire ≪fonctionner≫ les diagrammes de classes sur des diagrammes de séquence et de collaboration
- Itérer et enrichir ! !

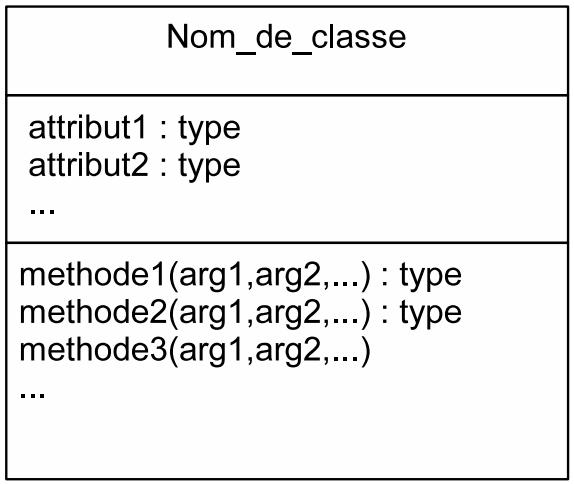
Synthèse de la démarche



(source : *Modéliser un site e-commerce*, Pascal Roques, Eyrolles)

Spécification d'une classe (.h)

```
class Nom_de_classe
{
 // Attributs
 type attribut1;
 type attribut2;
 ...

 // Constructeur
 Nom_de_classe(arg1,arg2,...);

 // Destructeur
 ~Nom_de_classe(void);

 // Méthodes
 type methode1(arg1,arg2,...);
 type methode2(arg1,arg2,...);
 void methode3(arg1,arg2,...);
 ...

};
```



Implantation d'une classe (.cpp)

```
Nom_de_classe::Nom_de_classe(arg1,arg2,...) {
 ...
}

Nom_de_classe::~Nom_de_classe(void) {
 ...
}

type Nom_de_classe::methode1(arg1,arg2,...) {
 ...
 return ...;
}

type Nom_de_classe::methode2(arg1,arg2,...) {
 ...
 return ...;
}

void Nom_de_classe::methode3(arg1,arg2,...) {
 ...
}
```

Instanciation, envoie de messages et destruction

- Instanciation

```
Nom_de_classe Objet1,Objet2;

Nom_de_classe Objet3(x,y,...);

Nom_de_classe *RefObjet4;

RefObjet4=new Nom_de_classe(x,y,...);
```

- Envoie de messages

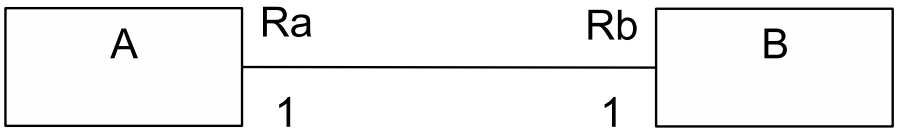
```
x=Objet1.Methode1(x,y,...);

RefObjet4->Methode3(x,y,...);
```

- Destruction

```
// La destruction des objets 1, 2 et 3 est automatique
// en fin de bloc

delete RefObjet4;
```

Association 1/1

```
class A
{
  ...
  B *Rb;
  ...
};
```




```
class B
{
  ...
  A *Ra;
  ...
};
```

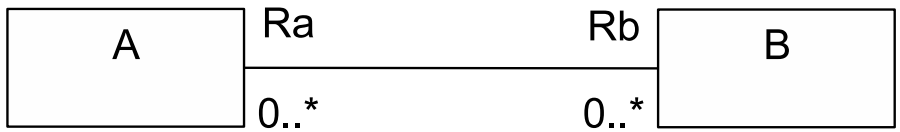
Association 1/1 unidirectionnelle

```
class A
{
  ...
  B *Rb;
  ...
};
```


```
class B
{
  ...
};
```

Association 0..*/0..*

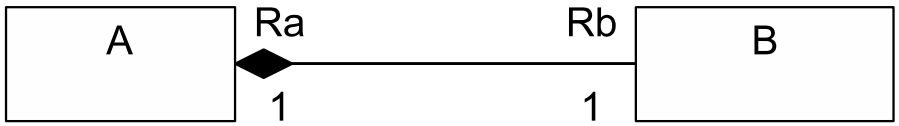
```
class A
{
 ...
 set<B*> Rb;
 ...
};

class B
{
 ...
 set<A*> Ra;
 ...
};
```



Composition 1/1

```
class A
{
 ...
 B Rb;
 ...
};
```

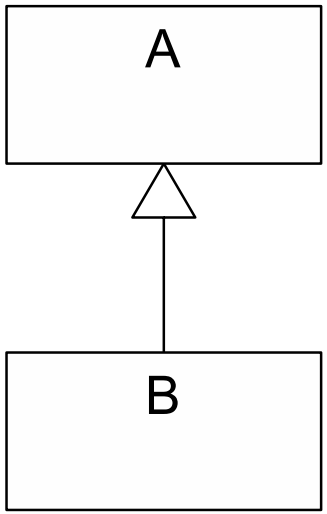
```
class B
{
 ...
 A *Ra;
 ...
};
```



Généralisation/spécialisation

```
class A
{
 // Attributs
 type attribut1;
 type attribut2;

 // Constructeur
 A(...);

 // Méthodes
 type methode1(...);
 type methode2(...);
};
```

```
class B : public A
{
 // La classe B hérite de tous les
 // attributs et méthodes de la
 // classe A

 // Nouveaux attributs propres à B
 type attribut3;
 type attribut4;

 // Constructeur (obligatoire)
 B(...)

 // Méthodes de A redéfinies pour B
 type methode2(...);

 // Nouvelles méthodes propres à B
 type methode3(...);
};
```



Bibliographie


Grady Booch, James Rumbaugh, and Ivar Jacobson.
The Unified Modeling Language User Guide.
Addison Wesley, 2005.


Object Management Group (OMG).
Unified Modeling Language (UML), version 2.0.


Pierre-Alain Muller and Nathalie Gaertner.
Modélisation Objet avec UML.
Eyrolles, 2000.


Pascal Roques.
UML en action.
Eyrolles, 2003.


Pascal Roques.
UML 2 par la pratique, études de cas et exercices corrigés.
Eyrolles, 2004.