université PARIS-SACLAY

Ingénierie du logiciel
Software engineering

Une méthode de conception pour la programmation orientée objet basée sur UP, UML & JAVA

2024-2025, Patrick RUIZ

Critères d'évaluation

- Evaluation de la démarche (20 points):

  Voir le document "UML conception orientée objet AAV".
- Soutenance finale et production (20 points)
  - Présentation marketing (5 points)
  - Présentation technique (5 points)
  - Application (10 points) :
    - Complexité technique
    - Originalité
    - Démonstration
    - Vidéo

3 étapes fondamentales pour atteindre ces objectifs:

- *spécifier* le problème à résoudre (description précise des fonctionnalités)
- *concevoir* une solution (méthode d'analyse)
- *implémenter* (mettre en œuvre, réaliser, construire) cette solution (choix du matériel, choix du langage)

Spécifier et concevoir

implémenter

Les ancêtres:

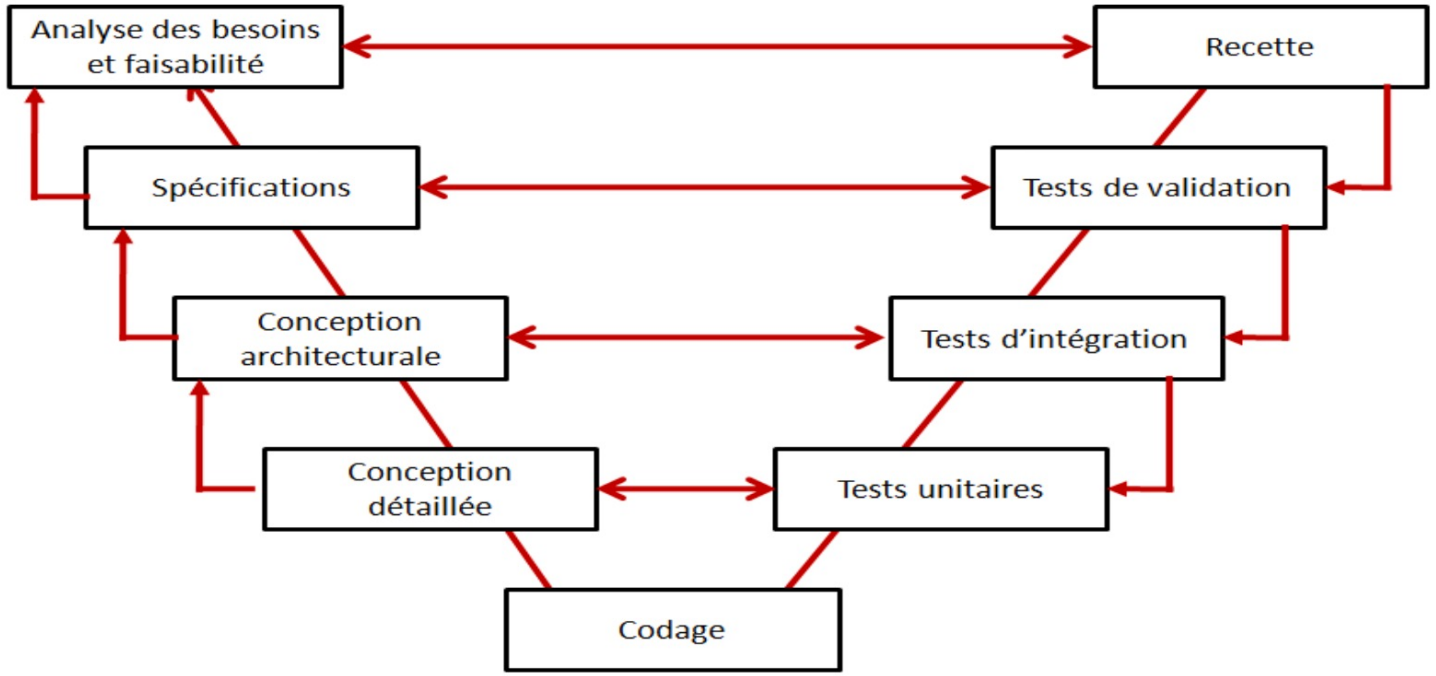
- Le cycle en V (1976):

Cascade

+

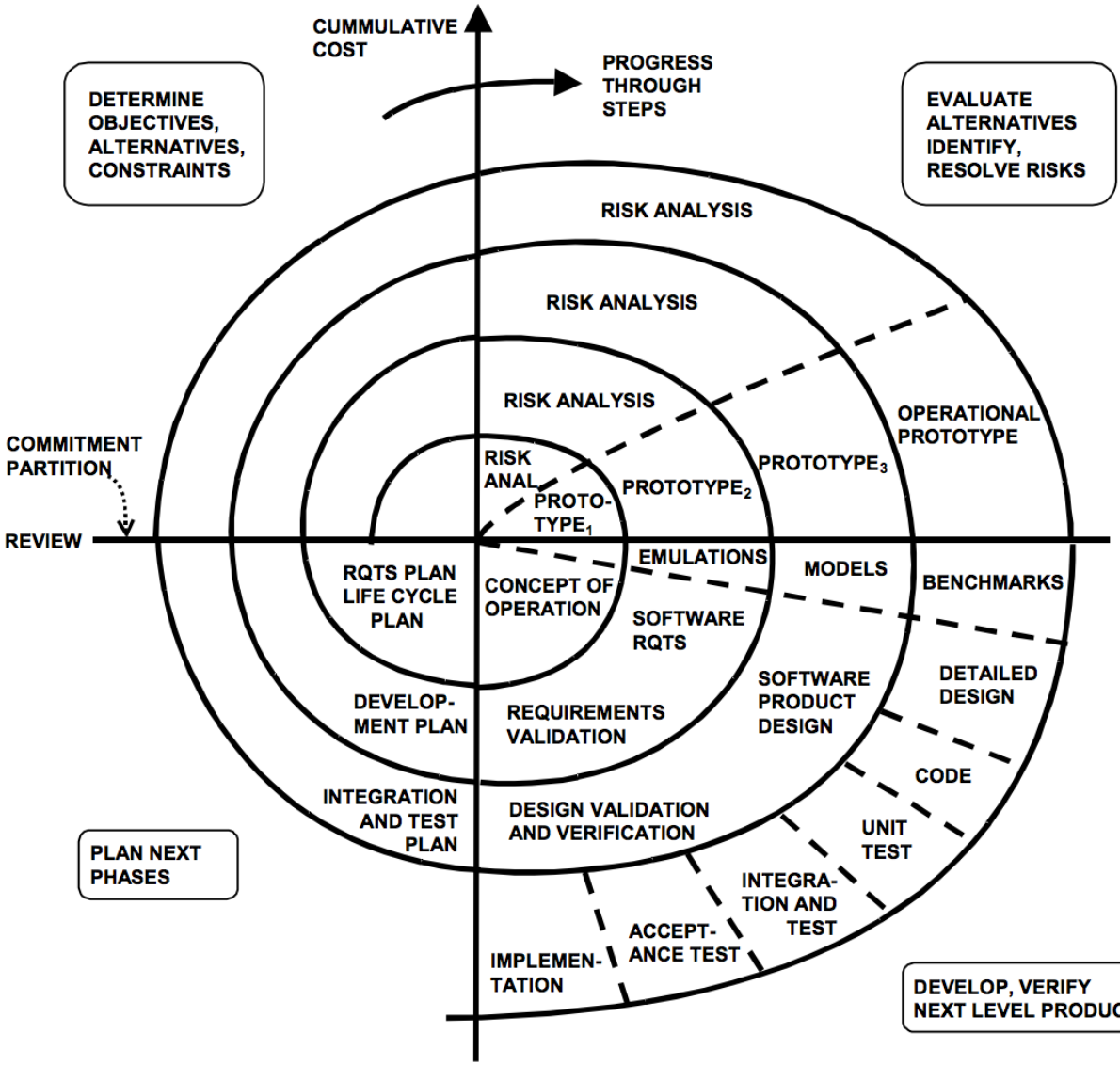
Tests rapides

- La spirale (1986)
- Merise (années 1980)



Les ancêtres:

- Le cycle en V (1976)
- La spirale (1986):
  4 phases itératives pilotées par le risque
- Merise (années 1980)



Les ancêtres:

- Le cycle en V (1976)
- La spirale (1986):
- Merise (années 1980): Modèle Conceptuel de Données (BDD relationnelles)

Un besoin qui s'accentue dans les années 1990:

POURQUOI ?

Résultat d'une étude sur 8 380 projets de 1995:

- Succès : 16 % ;
- Problématique : 53 % (budget ou délais non respectés, défaut de fonctionnalités)
- Échec : 31 % (abandonné)

CAUSES

- *Généralisation de l'informatique dans tous les domaines de l'entreprise*
- *Accroissement de la puissance et de la mémoire des ordinateurs*
- *Mise en concurrence mondiale des solutions logicielles*

Critères de qualité d'un logiciel *par ordre d'importance*:

- Adéquation entre le logiciel et les besoins des utilisateurs (fonctions, ergonomie)
- Fiabilité (stabilité)
- Performance (fluidité)
- Portabilité (multi-plateforme)
- Evolutivité (simplicité pour rajouter des fonctionnalités)
- Facilité de maintenance (identification/résolution aisée des bugs même une fois les développeurs initiaux partis !)

UML & UP

- **Unified Modeling Language** est un langage *graphique* de modélisation des données et des traitements pour la POO

  ➔ le plus connu et le plus utilisé

- **Unified Process** est un patron de processus de développement logiciel, ancêtre des méthodes agiles
  - adaptable à une large classe de systèmes logiciels, de domaines d'application, de niveaux de compétences.
  - UP s'appuie sur la notation UML pour la production de documents
  - Autres méthodes Agile: XP, Scrum, Kanban…

Historique d'UML

https://www.omg.org

OMG (Object Management Group) est une association dont l'objectif est de standardiser le modèle objet

Standardisation par l'OMG

Soumission à l'OMG

Version bêta OOPSLA'96

OOPSLA'95

UML 2.5.1 — 2017

UML 2.4 — 2011

UML 2.0 — 2005

UML 1.0 — Janvier 1997

UML 0.9 — Juin 1996

https://www.youtube.com/watch?v=vAHHdnIV8rU

Méthode unifiée 0.8 — Octobre 1995

Booch'93 OMT-2

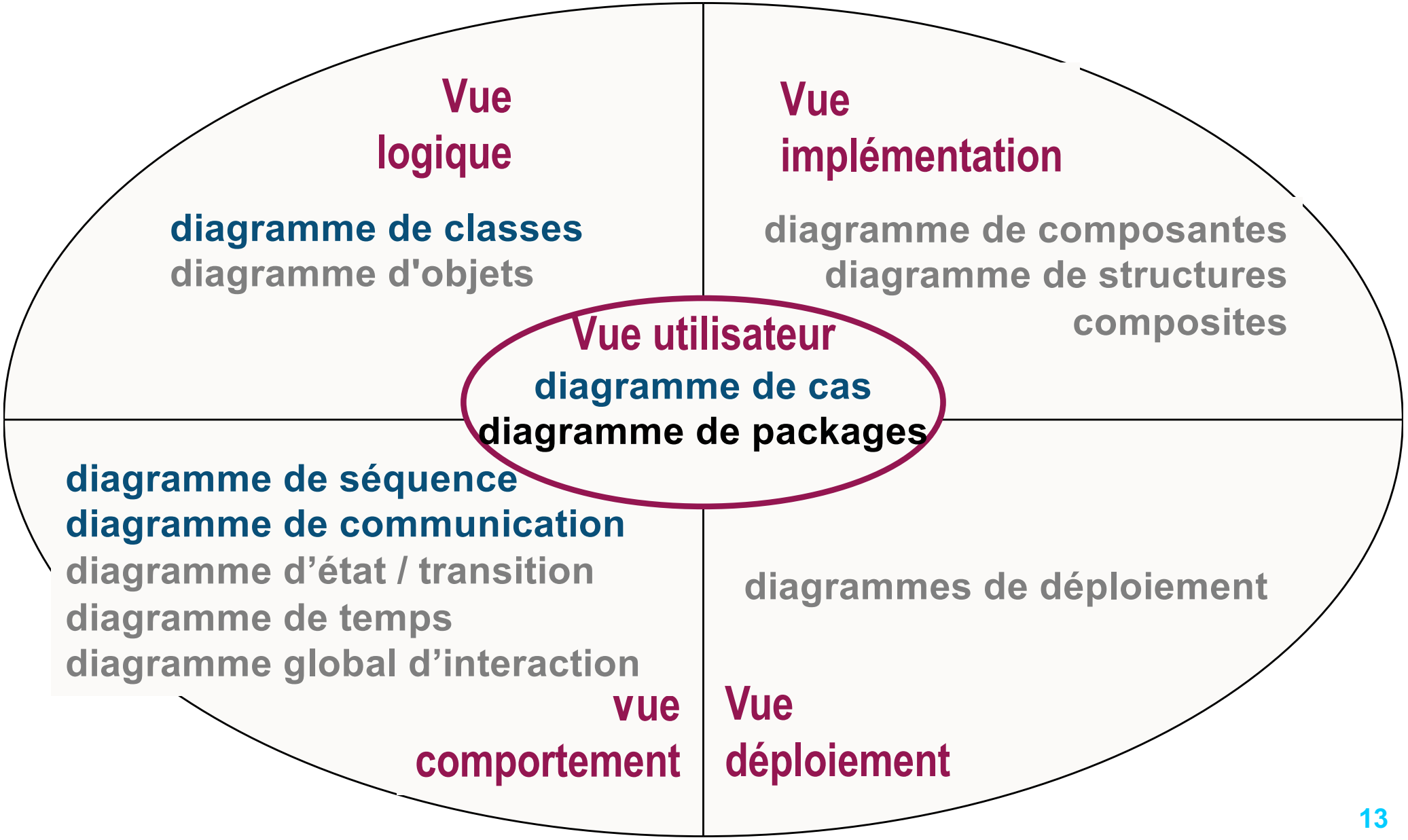
Autres méthodes Booch'91 OMT-1 OOSE Partenaires

Présentation du *Unified Process* et du *Unified Modeling Language*

https://www.omg.org/spec/UML/2.5.1/PDF

université PARIS-SACLAY

UML… beaucoup de diagrammes !



université PARIS-SACLAY

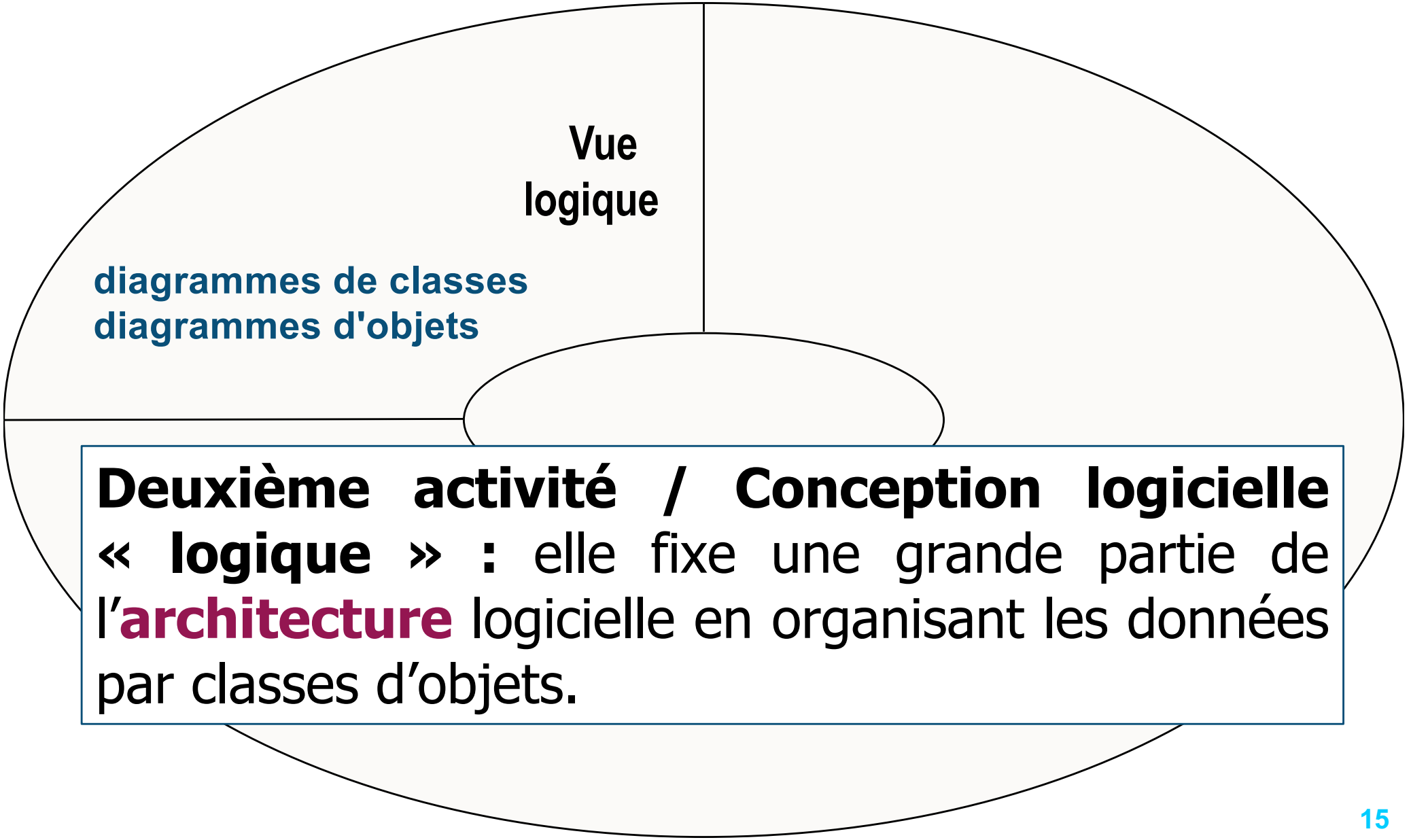
UML… beaucoup de diagrammes !

Vue utilisateur
diagrammes de cas

Première activité / expression des besoins: description du système comme un ensemble de transactions du point de vue de l'utilisateur.

université PARIS-SACLAY

UML… beaucoup de diagrammes !



Deuxième activité / Conception logicielle « logique » : elle fixe une grande partie de l'**architecture** logicielle en organisant les données par classes d'objets.

université PARIS-SACLAY

UML… beaucoup de diagrammes !

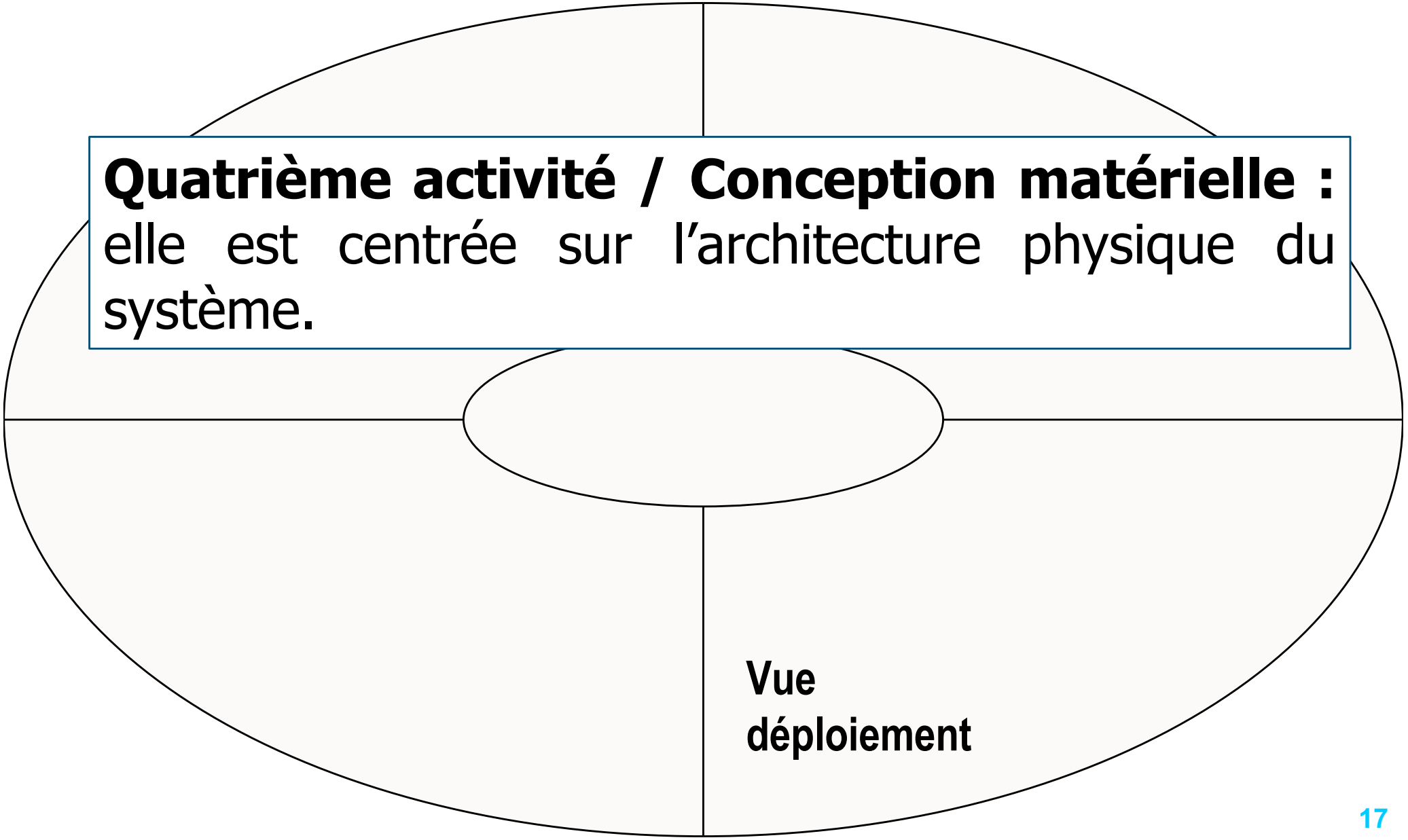
Troisième activité / Conception logicielle
« **processus** » **:** il s'agit de définir les **interactions** entre les différents objets constitutifs du programme.

diagramme de séquence
diagramme de communication
diagramme d'état / transition

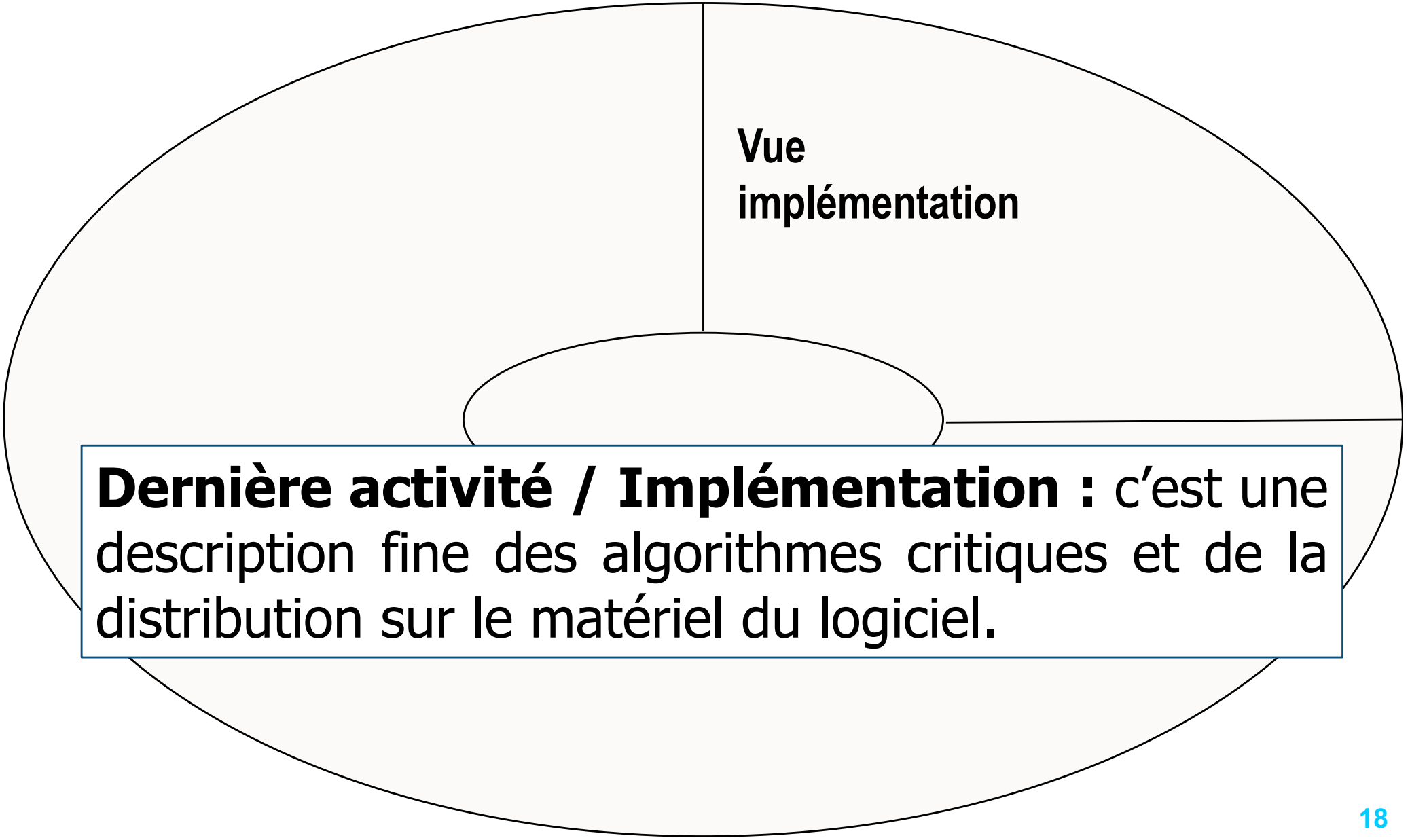
Vue
Processus

UML… beaucoup de diagrammes !

Quatrième activité / Conception matérielle : elle est centrée sur l'architecture physique du système.



UML… beaucoup de diagrammes !

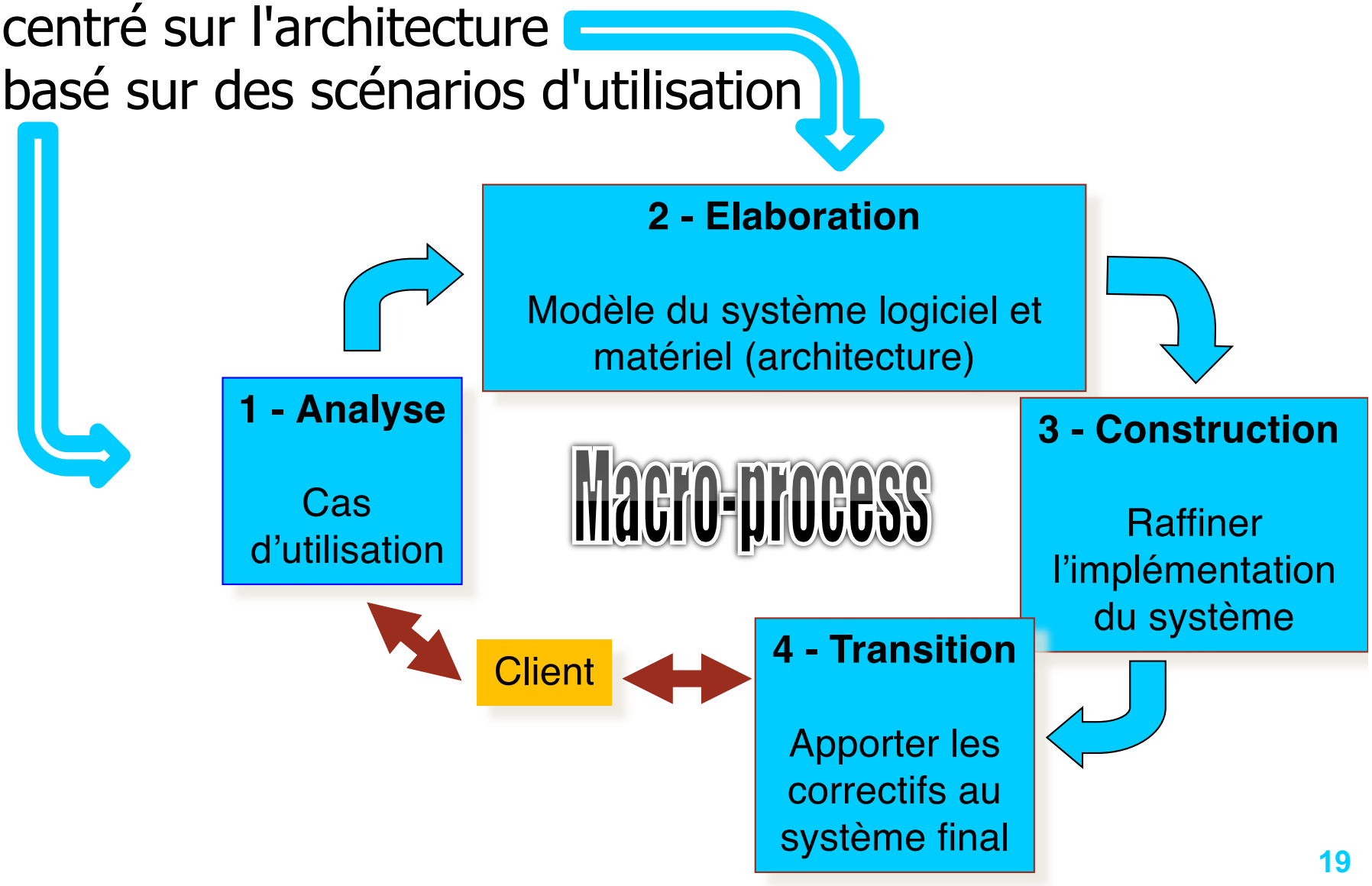


Dernière activité / Implémentation : c'est une description fine des algorithmes critiques et de la distribution sur le matériel du logiciel.

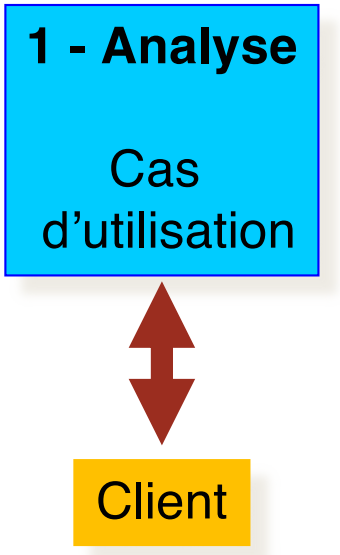
Principe du UP : Le Macro-process

le Processus Unifié (UP): 4 ETAPES du Macro-Process

- Il est centré sur l'architecture
- Il est basé sur des scénarios d'utilisation



4 ETAPES du Macro-Process



1ère ETAPE: Analyse des besoins

L'analyse des besoins donne une *vue* du projet **sous forme de produit fini**: prototype de l'interface et étude préliminaire des risques majeurs.

- Définir l'architecture générale du système
- Évaluer les risques majeurs (causes d'échec)
- Déterminer les délais (planning)
- Déterminer les coûts (justification économique du projet)

4 ETAPES du Macro-Process

2 - Elaboration

Modèle du système logiciel et matériel (architecture)

2ème ETAPE: Elaboration

L'élaboration reprend les éléments de la phase d'analyse des besoins et les précise pour arriver à **une *spécification* détaillée** de la solution à mettre en œuvre.

- préciser la plupart des ***cas d'utilisation*** (80%)
- en déduire l'**architecture logicielle et matérielle** du système
- un ***prototype*** qui s'attaque aux **risques** : mise en œuvre l'ensemble des ***fonctionnalités critiques***
- ***délais : planification*** *fine* de la phase de construction
- ***coûts : estimation précise*** **des ressources** nécessaires à l'achèvement du projet.

4 ETAPES du Macro-Process

3ème ETAPE: Construction

L'architecture se métamorphose en **produit complet**.

- A l'issu le prototype contient **tous les cas d'utilisation** pour cette version,
- une première version de la documentation complète est disponible

3 - Construction

Raffiner l'implantation du système

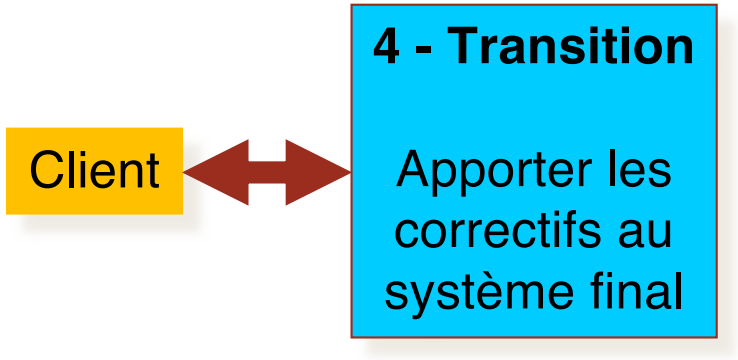
4 ETAPES du Macro-Process

4ème ETAPE : Transition

Le produit est en ***version bêta :*** Un groupe d'utilisateurs essaye le produit et détecte les anomalies et défauts.

Cette phase suppose des activités :

- **formation** des utilisateurs clients
- mise en œuvre d'un **service d'assistance**
- correction des anomalies constatées.



université PARIS-SACLAY

Principe du UP : le Micro-process

A l'intérieur de chaque étape le **Processus Unifié** **est itératif : il répète le même micro-process**

Pourquoi ?

Pour intégrer au plus vite les demandes de modification du client et l'associant étroitement aux étapes du développement

Pour éviter le risque de perte d'investissement en identifiant tôt les difficultés qui feront que l'objectif annoncé au client ne pourra être atteint dans les délais ou en respectant la marge

Notion clé: éviter de vouloir décrire entièrement le problème avant de commencer à implanter du code et à le tester

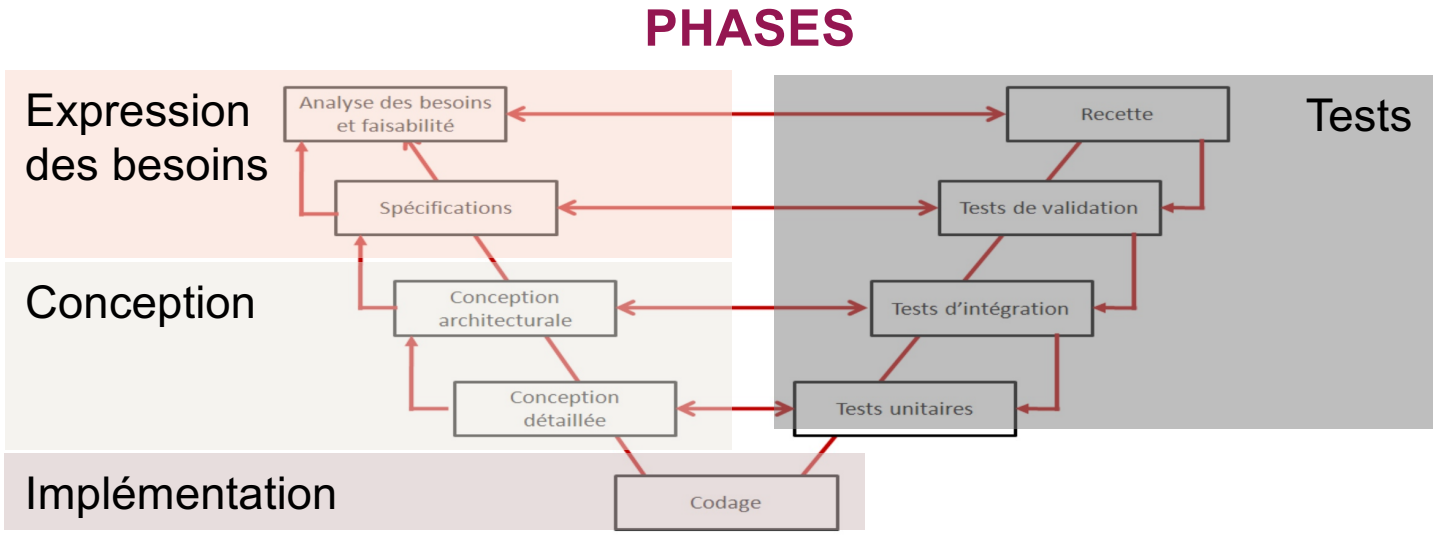
- Itération 1: interaction avec le client pour décider des fonctionnalités: vérification de la compétence de l'entreprise sur les fonctionnalités les plus risquées / incertaines
- Itération 2: implémentation de fonctionnalités critiques

  etc …
- Itération finale: implémentation des fonctionnalités les plus évidentes

Principes du UP

Chaque itération suit un *micro*processus du type « cycle en V » visant les fonctionnalités souhaitées pour l'itération. Le cycle en V est donc répété pour chaque itération, le programme s'enrichie au fur et à mesure.

Microprocessus



Microprocess

Expression des besoins (UML-VUE UTILISATEUR)

① Recenser les **besoins fonctionnels** (du point de vue de l'utilisateur) qui conduisent à l'élaboration des **modèles de *cas d'utilisation***

② Issus des cas d'utilisation, des *scénarios* racontent (graphiquement) ce que fait l'utilisateur

③ appréhender les **besoins non fonctionnels** (techniques) et en déduire une première estimation du coût et délais

Analyse (UML-VUE conception LOGIQUE)

① Un prototype d'analyse (non-fonctionnel) livre une spécification complète des besoins utilisateur

② Produire un modèle d'analyse UML, première ébauche des processus internes: *premier jet du* ***diagramme de classe***

université PARIS-SACLAY

Microprocess

Conception (UML-VUE conception PROCESSUS)

① faire les choix technologiques logiciels et matériels permettant l'implémentation du modèle d'analyse

② acquérir une compréhension approfondie des contraintes liées au langage de programmation, à l'utilisation des bibliothèques et au système d'exploitation.

③ proposition plus précise d'architecture logicielle et matérielle: ***diagramme de séquences***

Implémentation (UML-VUE IMPLEMENTATION)

- produire les classes et les sous-systèmes sous formes de codes sources
- la retouche du modèle de conception est possible

Tests:

Créer des cas de **tests unitaires et d'intégration**, effectuer ces tests et prendre en compte le résultat de chacun.

université PARIS-SACLAY

Unified-Process: Vue d'ensemble

Macroprocess

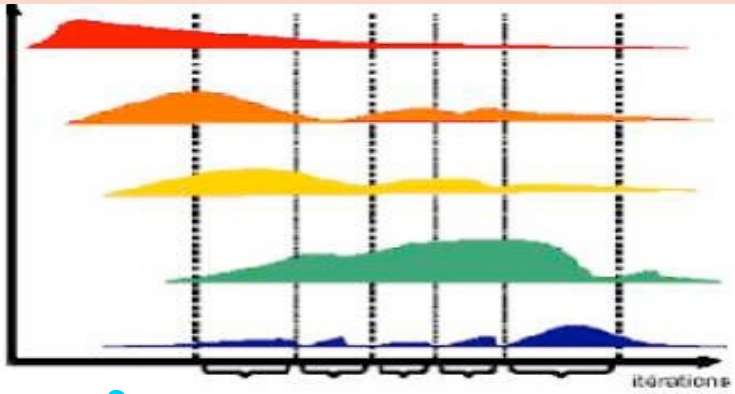
4 ETAPES

1 - Analyse besoins | 2- Elaboration | 3- Conception | 4- Transition

Microprocess

5 PHASES

- Expression des besoins
- Analyse
- Conception
- Implémentation
- Tests



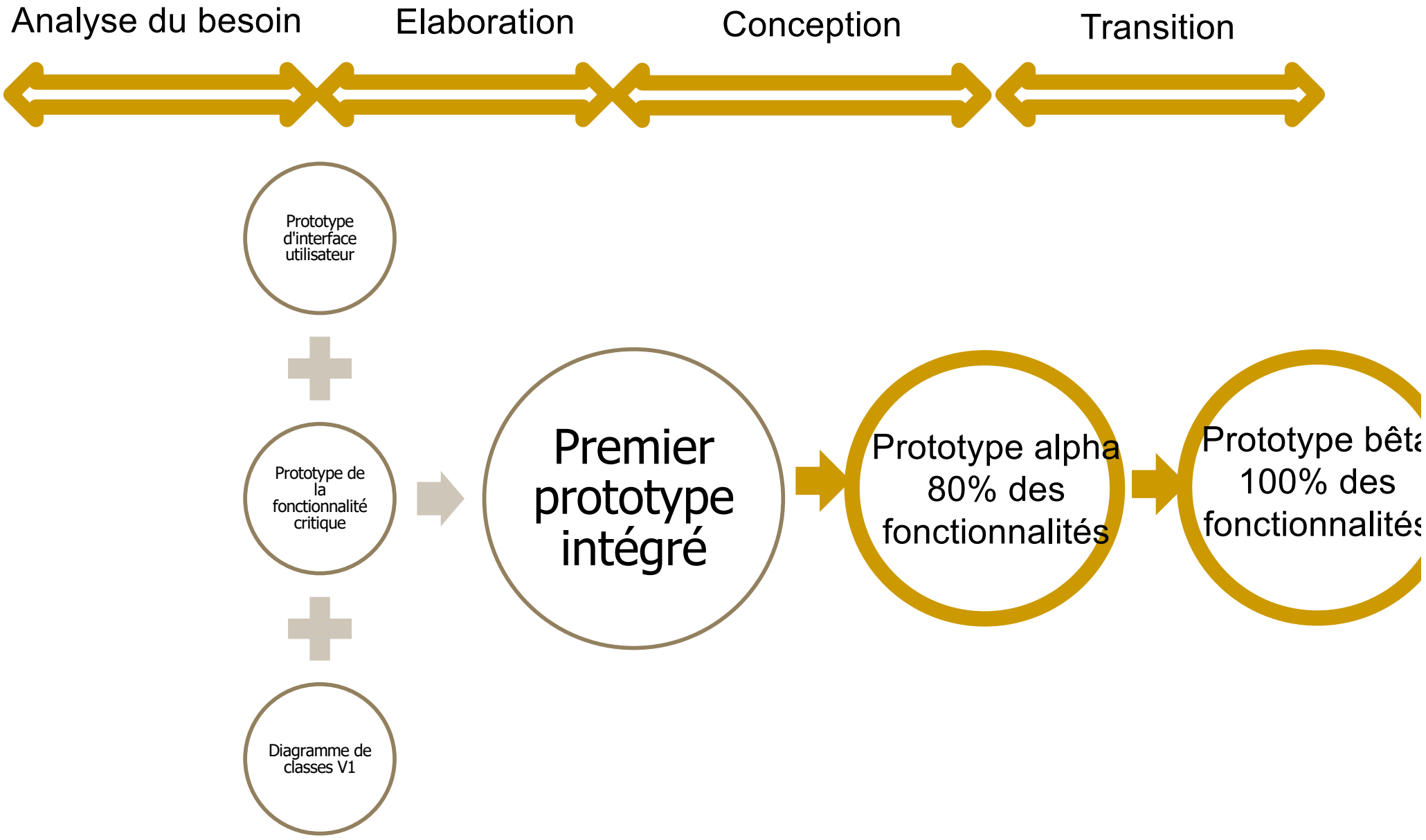
Itérations n°1 2 3 4 5 6 7

- Itération 1: interaction avec le client pour décider des fonctionnalités: vérification de la compétence de l'entreprise sur les fonctionnalités les plus risquées / incertaines
- Itération 2: implémentation de fonctionnalités critiques

  etc …
- Itération 6: implémentation des fonctionnalités les plus évidentes
- Itération 7: correction de bugs de la version bêta

université PARIS-SACLAY

Synthèse des étapes modèle View-Logic-Data



université PARIS-SACLAY

Planning du cours

A – Evaluation, contenu, logiciels (1H)

B – Présentation de UML & UP (C) (1H)

C – **UP « Expression du besoin » :**

- diagramme de cas d'utilisation UML
- diagramme de classe UML (C - TD)
- *première application à votre projet de ces deux diagrammes* (3H)

D – Première analyse des fonctionnalités critiques

- Identification de la fonctionnalité wireless : *sitographie*
- Identification de la fonctionnalité critique : *sitographie (2H)*

E – **UP « Elaboration » :**

- TP de découverte de la programmation Androïd (6H).

- Codage des classes limitées aux attributs (1H)

- Codage d'un prototype non fonctionnel du projet (2H)

>> JALON 1 « engagement » pour le 4 novembre au plus tard: déposer sur eCampus *1- le prototype* ***non-fonctionnel*** de l'application 2 - le *diagramme* ***de cas***, 3 - le *diagramme de* ***classes*** *sans les méthodes 4 - le prototype de la* ***fonctionnalité critique***

université PARIS-SACLAY

Planning du cours

F – **UP « Conception »** :

- Le diagramme de séquence
- *application à l'architecture du projet*
- *codage des associations.* (4H)

>> JALON 2 « version alpha » pour le 11 novembre :

Déposer sur eCampus le diagramme de **séquence**, le diagramme de **classes** *incluant les méthodes,* un ***prototype fonctionnel partiel*** *fusionnant le programme d'interface et celui de la fonctionnalité critique*

G – **UP « Implémentation »** :

Passage de la version alpha à la version bêta qui assure parfaitement 80% des fonctionnalités de la version finale (4H).

>> JALON 3 « version bêta » pour le 18 novembre : déposer la ***version bêta.***

Planning du cours

H – **UP « Transition »**

>> JALON 4 pour le 21 novembre :

Déposer les éléments suivants **AVANT le début des soutenances**

- **Diaporama de l'application pour *au client* :** raison d'être marketing et mode d'emploi *de ce qui marche uniquement*.

- **Diaporama pour *la maintenance technique*** donnant des détails sur la mise en œuvre de la fonctionnalité critique, **en s'appuyant sur un diagramme de séquence,** et éventuellement de classes.

- **Vidéo** démonstrative de la **version bêta** de l'application (ce qui marche uniquement !)

- **Diagramme de cas *final***

- **Diagramme de séquence *final***; cette vue processus doit représenter le scénario le plus complexe de l'application de la démonstration.

- **Diagramme de classes avec *100% des attributs et méthodes de la démonstration***

Le formalisme du Unified Modeling Language

Vue utilisateur – diagramme de Cas

université PARIS-SACLAY

Diagramme de cas d'utilisation (ou de besoins)

Les diagrammes de **cas d'utilisation** décrivent, sous la forme d'**actions** et de réactions, le comportement d'un système du point de vue de l'utilisateur, encore appelé **acteur (correspond à un et un seul rôle)**.

Pour établir ce diagramme:

1.Recenser chaque acteur (peuvent être des machines: acteurs « secondaire »)

2.Recenser l'ensemble des fonctionnalités d'un système en examinant les besoins fonctionnels de chaque acteur

3.Eviter les redondance et les micro-cas d'utilisation

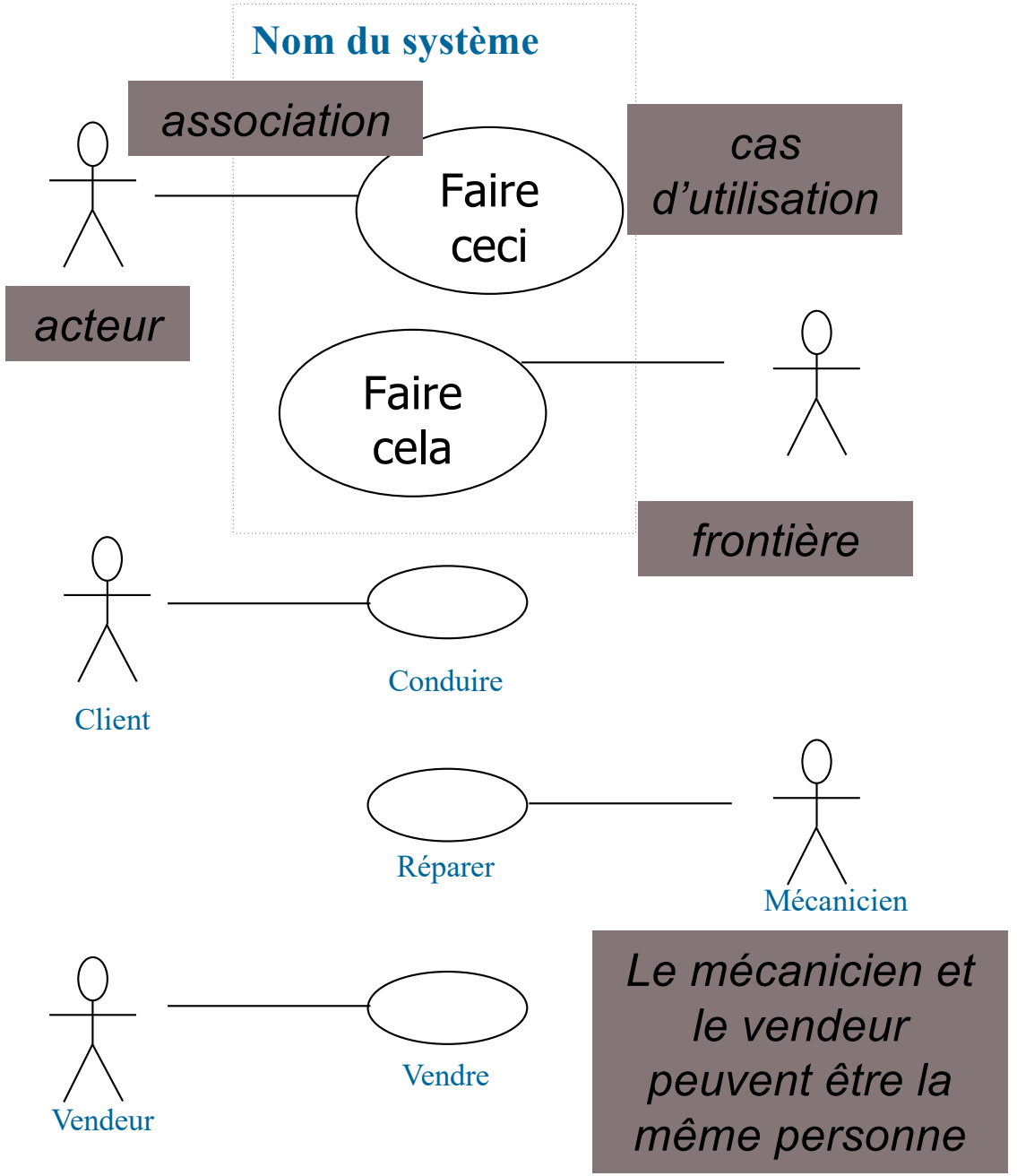
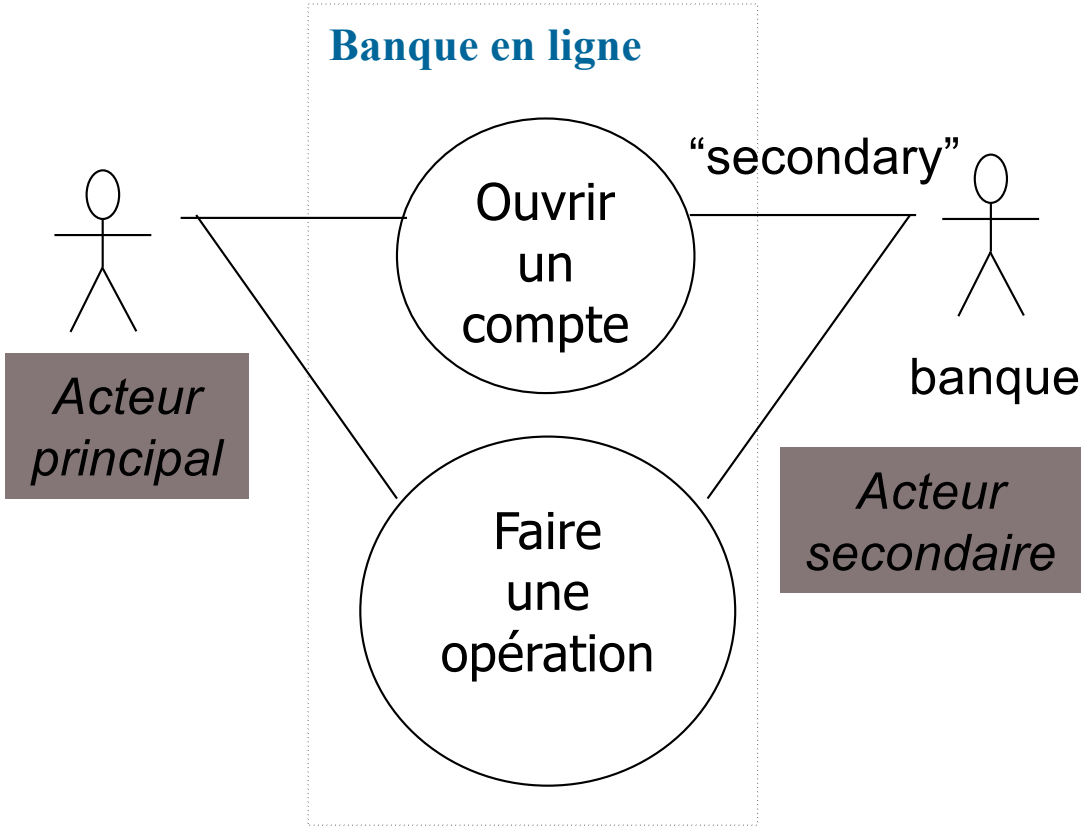


Diagramme de cas d'utilisation (ou de besoins)

On distingue deux types d'acteurs:

- les **acteurs principaux**. Cette catégorie regroupe les personnes qui utilisent les fonctions principales du système.

- les **acteurs secondaires**. Cette catégorie regroupe les personnes qui effectuent des tâches administratives, de maintenance. Ce sont parfois des systèmes informatiques communiquant.



université PARIS-SACLAY

Diagramme de cas d'utilisation (ou de besoins)

Description textuelle des cas d'utilisation

IDENTIFICATION:

- Nom du cas:
- Objectif:
- Acteurs:
- Version:
- Auteur:

SEQUENCEMENTS:

- Prérequis:
- Enchaînement :
- Suite:

Exemple: le client léger callshop VOIP

Ci-dessous quelques notes prises suite à une demande téléphonique d'un client

- Le prix d'une communication vers les îles Fidji est de 0,9€ / min.
- Après l'introduction de la monnaie dans le système cashless, l'utilisateur a 2 mn pour composer son numéro (ce délai est décompté par le standard).
- La ligne peut être libre ou occupée.
- Le correspondant peut raccrocher le premier.
- Le système cashless consomme de l'argent dès que l'appelé décroche et à chaque unité de temps (UT) générée par le standard.
- On peut ajouter des pièces à tout moment.
- Lors du raccrochage, le solde de monnaie est rendu.

Vous avez un rendez-vous avec le client pour éclaircir les choses… utilisez un diagramme de cas d'utilisation pour aider à l'analyse du besoin

Un exemple: le client léger callshop VOIP

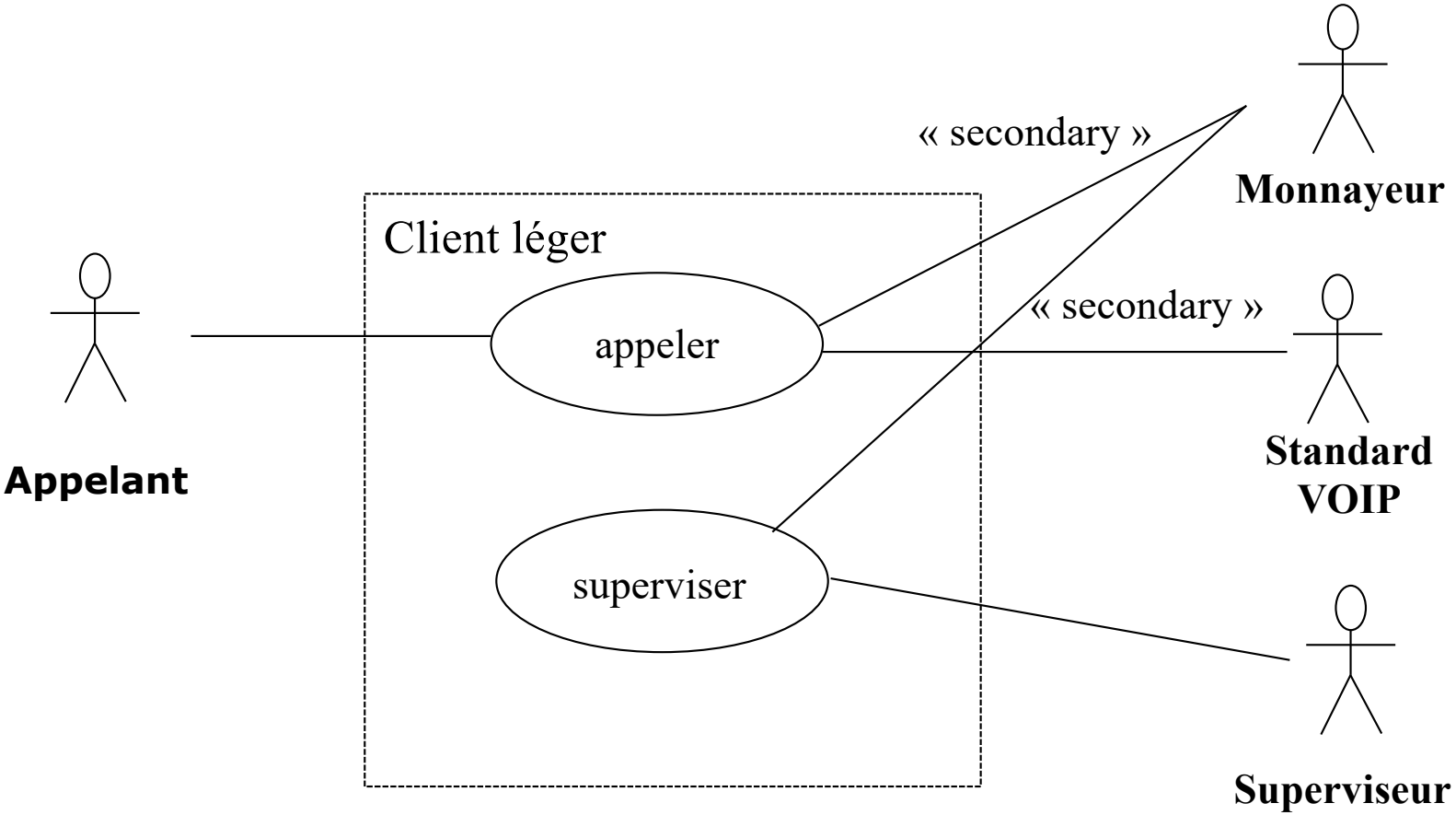


Diagramme de cas : texte

Description textuelle des cas d'utilisation

IDENTIFICATION:

- Nom du cas:
- Objectif:
- Acteurs:
- Version:
- Auteur:

SEQUENCEMENTS:

- Prérequis:
- Enchaînement :
- Suite:

Diagramme de cas: texte

Description textuelle des cas d'utilisation

IDENTIFICATION:

- Nom du cas:
- Objectif:
- Acteurs:
- Version:
- Auteur:

SEQUENCEMENTS:

- Prérequis:
- Enchaînement :
- Suite:

Le formalisme du Unified Modeling Language

Vue logique : diagramme de Classes diagramme d'Objets

université PARIS-SACLAY

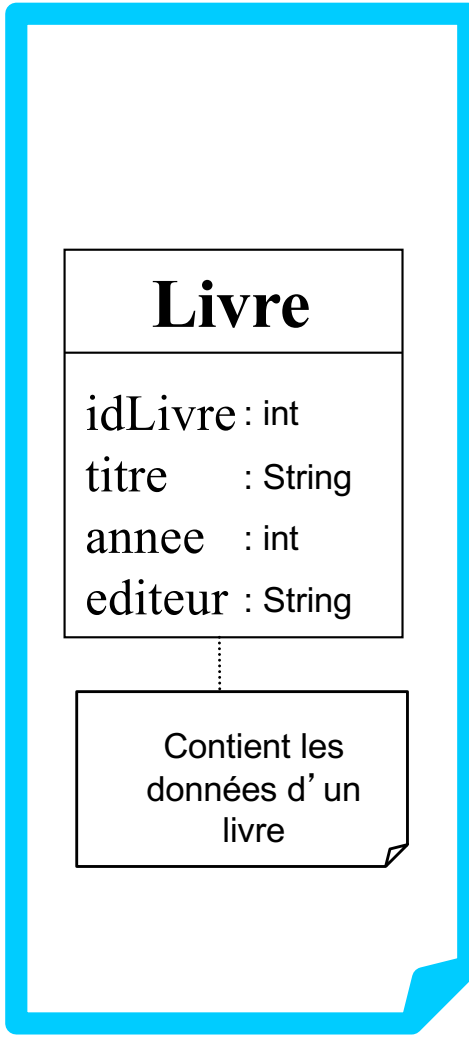
Diagramme de classes

- Le diagramme de classe permet de d'organiser les données du système par classes d'objets
- Méthode pour des projets de petite taille:
  1. Interviewer les utilisateurs et faire une liste des données, utiliser éventuellement comme support des exemples de documents attendus en sortie.
  2. Remplacer les données calculées ou composées (adresse) par les données élémentaires.
  3. Rechercher les dépendances fonctionnelles: c'est-à-dire tout attribut étant lié naturellement à d'autres attributs, cela fait apparaître les classes.
  4. Un attribut ne doit apparaître que dans une classe
  5. Une classe ne possède qu'un seul rôle

Représentation d'une classe

Les exemples correspondent à la création d'une base de donnée pour une Bibliothèque

- Collecter les données:
  - Quelles sont toutes les **familles** de données nécessaires à la réalisation du projet?
    - *classe:* correspond à une famille de données similaires.
    - *Instance ou objet:* réalisation d'une classe.
    - Elle porte un nom dont la première lettre est majuscule
      exemple: Livre.
  - Quelles sont toutes les données utiles pour une classe donnée?
    - *Attribut*: propriété ou donnée d'une classe.
    - Il faut préciser la structure de chaque attribut: type simple, composé ou autres classes.
  - Quelle est la *responsabilité* de la classe ?
    - Une classe devrait avoir peu voire une seule responsabilité.



Autres éléments descriptifs d'une classe

- Intègre un système de **traitement** des données

  Méthode (ou opération): il s'agit des fonctions permettant de travailler sur les attributs

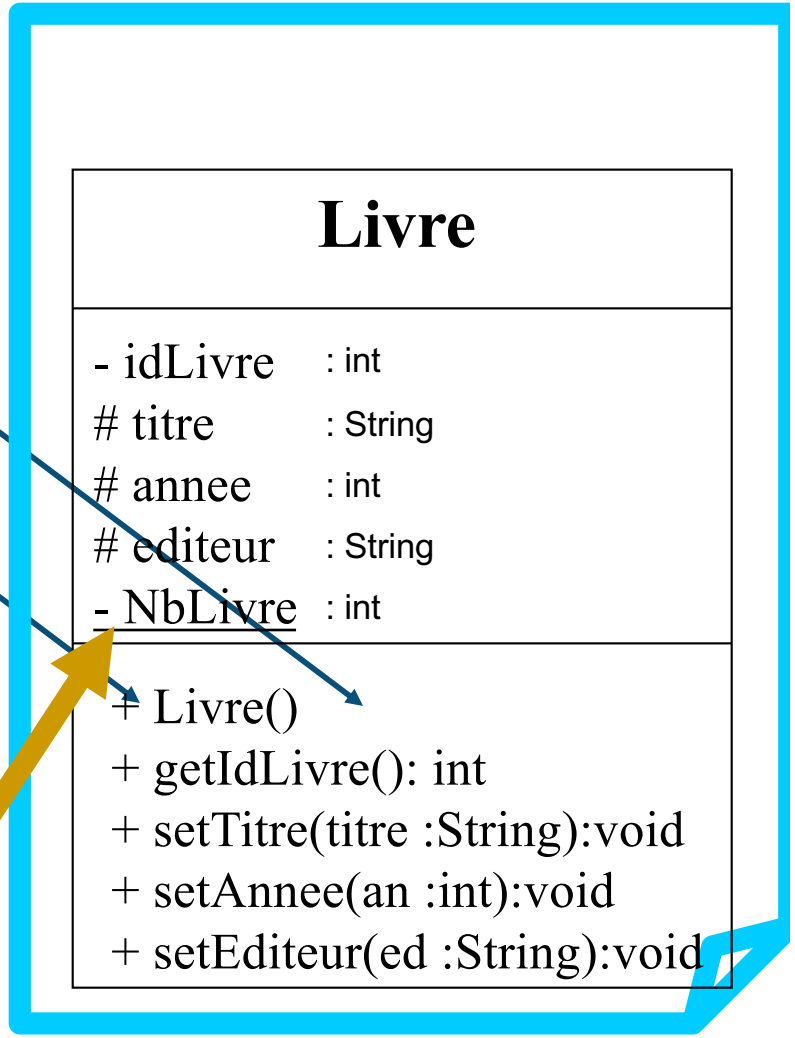
- Dispose d'un **constructeur**

  les constructeurs ressemblent à des méthodes , mais ils ne sont utilisés qu'à la création des objets de la classe. Ils servent à l'initialisation de l'objet.

- Vocabulaire:

  On appelle **membres** d'une classe l'ensemble des attributs et des méthodes d'une classe

  Les attributs et méthodes **de classe (pas d'instances)** sont soulignés



université PARIS-SACLAY

Rappel: signature et le concept de surcharge

- Définition:

  Plusieurs méthodes ou constructeurs d'une même classe peuvent posséder le même nom mais pas la même signature.

- Définition:

  La **signature** d'une méthode est le type de la valeur retournée et le type des paramètres qu'on trouve dans la déclaration (En C , c'est ce qu'on appelle le prototype d'une fonction)

- Intérêt : Permet l'**abstraction** , c'est-à-dire qu'une même action peut être implémentée de plusieurs manières tout en conservant le même nom.

Rappel sur le concept d'encapsulation

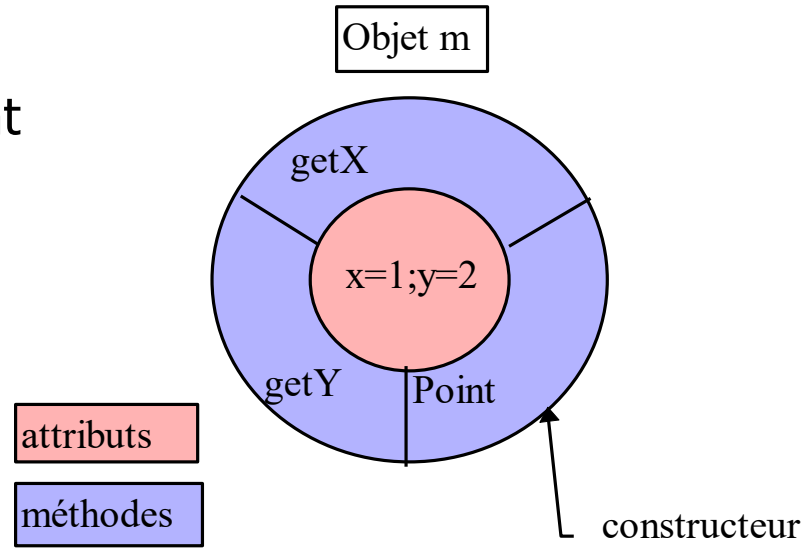
- Définition:

  Impossibilité pour un objet d'une classe d'accéder directement (sans passer par des méthodes) à un attribut appartenant à un objet d'une autre classe.

- C'est le concept qui évite :
  - **La corruption des données contenues dans l'objet**

    (erreur de programmation courante lorsque plusieurs développeurs travaillent sur le même projet).
  - La **nécessité de connaître l'implémentation** de **l'objet lorsqu'on l'utilise**.

    (En effet les méthodes suffisent à contrôler les données)

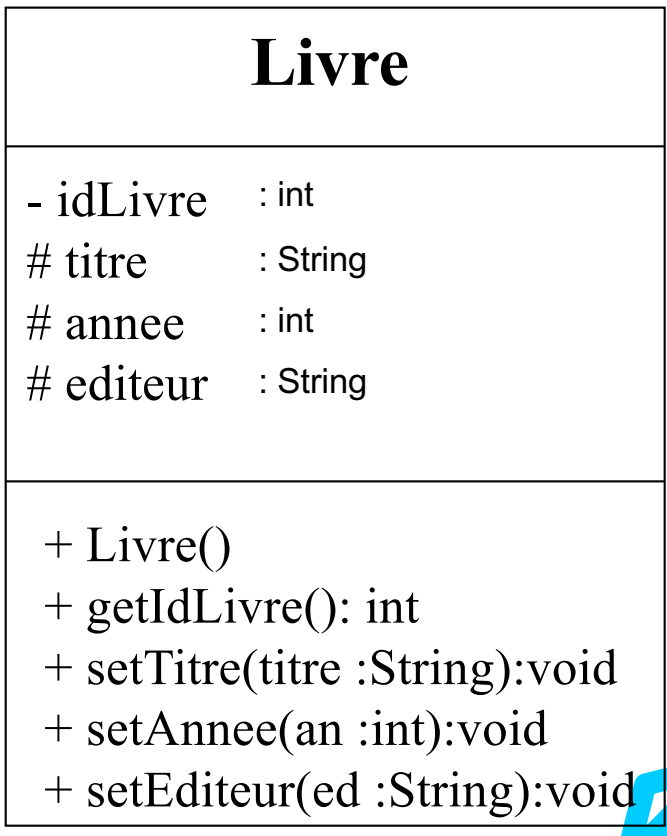


Notation UML de l'encapsulation: visibilité

La visibilité est:

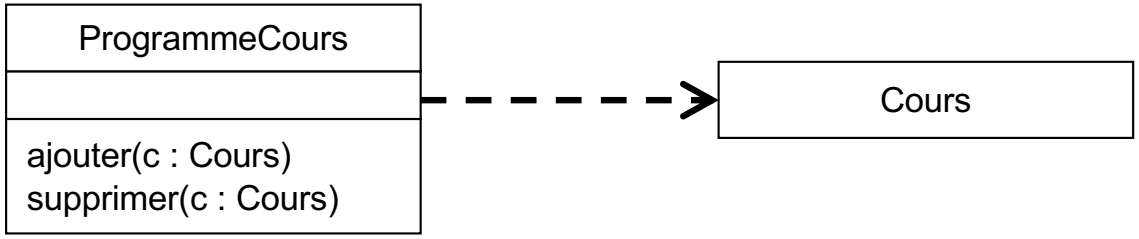
- **public** (UML '+'): Accès autorisé à tous les objets.
  Utilisation : A éviter pour les attributs. Conseillé *pour les méthodes*.
- **protected** (UML '#'): Accès autorisé aux objets de la classe , des classes filles et des classes du même paquetage.
  Utilisation : Pour les éléments non publics *a transmettre par héritage*.
- **private** (UML '-'): Accès autorisé seulement aux objets de la même classe.
  Utilisation : Pour des éléments "secrets" dont la modification risque de rendre l'objet inutilisable. Attention , pas de transmission par héritage. Usage rare.

Modélisation par schéma UML



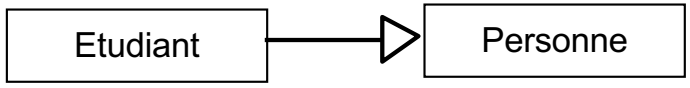
Relations entre classes

- Différentes collaborations
  - ***dépendances (utilise/uses)***: relation d'utilisation entre deux classes, c'est une relation **indicative, sans pendant dans le code** (peu utilisée dans la pratique)



Se lit: programmeCours ***utilise*** *cours*

  - ***héritage*** ou *généralisation (hérite de/ inherits)*: c'est une relation classe mère/classe fille.



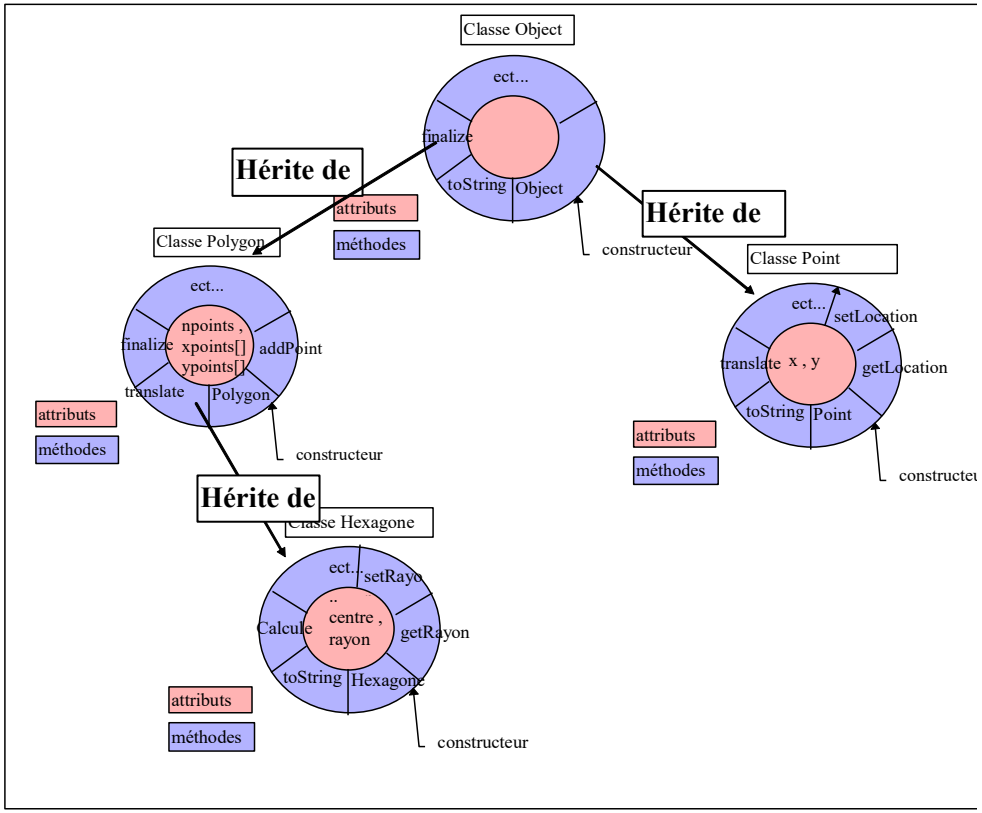
Se lit: Etudiant ***hérite de*** *Personne*

Rappel: héritage

Définition:
Lorsqu'une sous-classe (classe fille) hérite d'une super-classe (classe mère) elle possède d'emblée tous les membres de sa super-classe (sauf ceux qui ne lui sont pas accessibles par encapsulation: visibilité private).

Remarques:

- Toute modification apportée à la classe mère **se répercute** sur la classe fille: C'est le concept qui permet une récupération facile du code déjà écrit.
- L'héritage est une propriété des **classes** pas des objets !
- Les constructeurs ne sont pas hérités : ce ne sont pas des membres de la classe à part entière.



Relations entre classes

- Différentes collaborations
  - ***associations***: matérialisable par un **verbe**, indique que les attributs d'une classe sont reliées à ceux d'une autre classe à travers une **action**.

Conséquence: à partir de l'instance d'une classe (objet) on peut passer à l'instance d'une autre classe.

C'est la collaboration la plus courante

Comment trouver les associations ?

Les associations proviennent de règles de gestion exprimant le rôle joué par les données, elle proviennent de l' interview des utilisateurs.

Règles spécifiques des associations

- **Multiplicité** (cardinalité des bases de données):

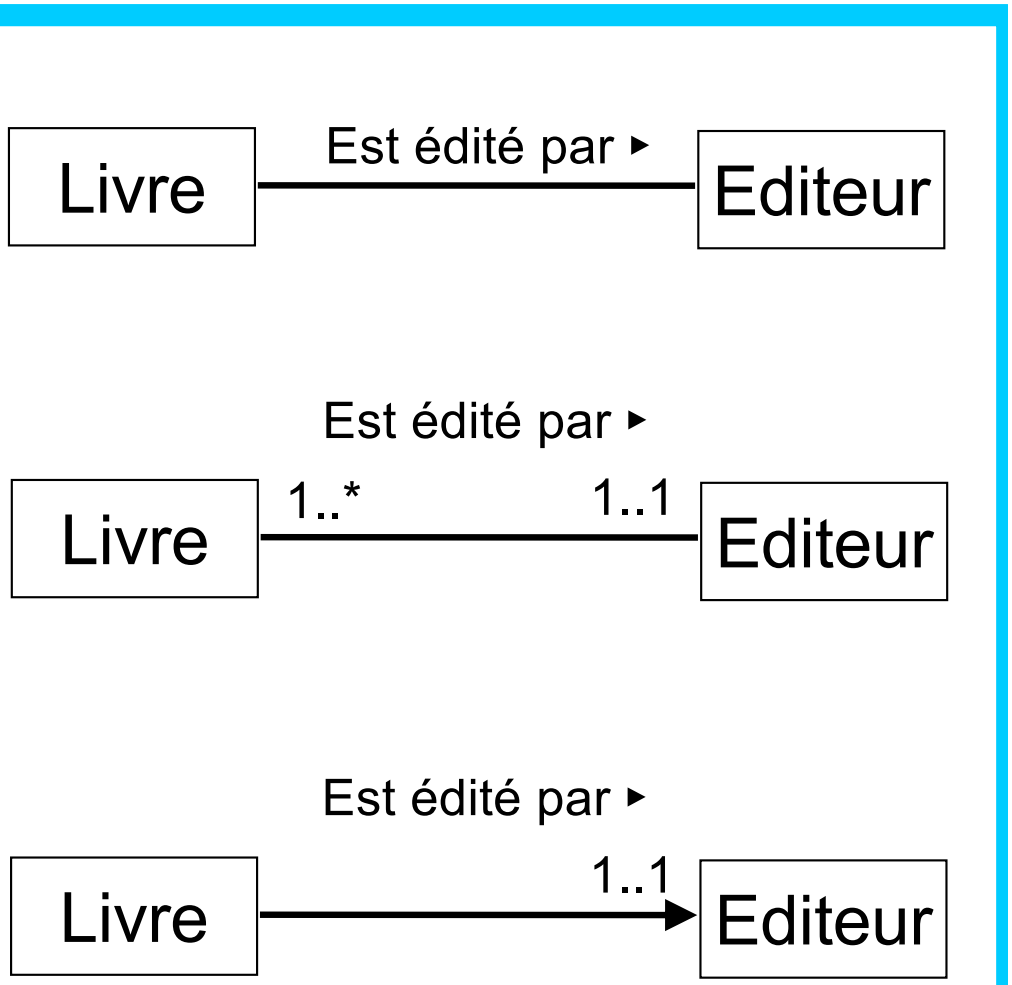
C'est le nombre de représentants (minimum..maximum) de chaque structure qui peuvent être liés par l'association. * = plusieurs

Exemple:

- Un livre est édité par un éditeur au minimum et au maximum
- Un éditeur édite au minimum un livre et au maximum un nombre indéterminé

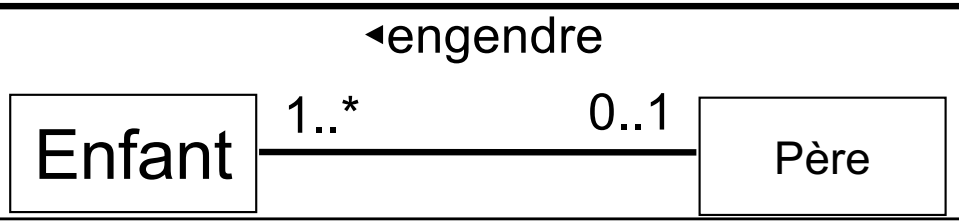
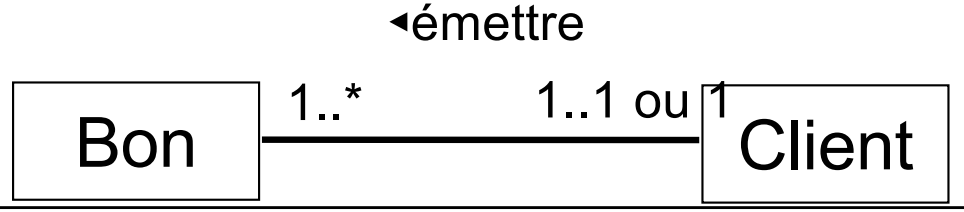
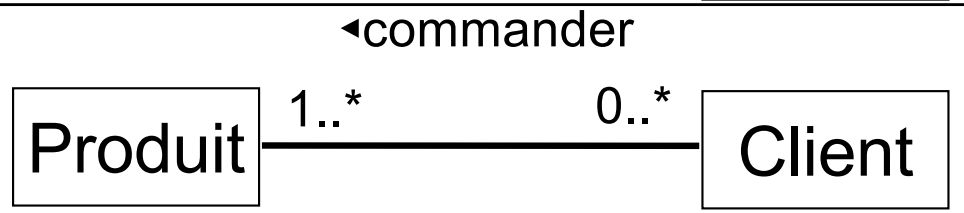
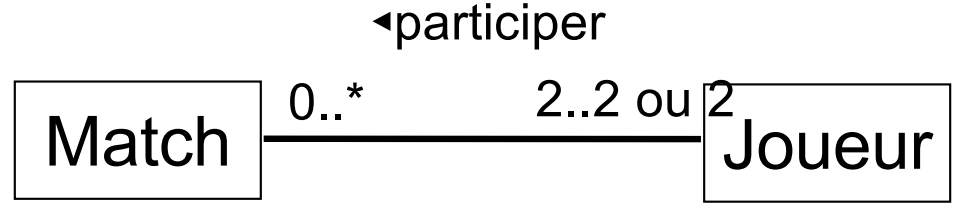
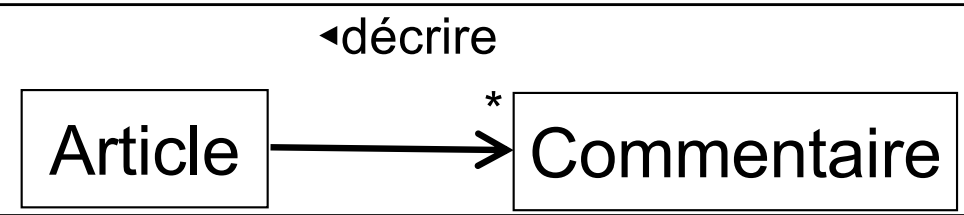
Navigabilité:

- Permet de spécifier dans quel sens l'association peut-être traversée: on place une flèche sur l'association
- Par exemple: on peut retrouver l'éditeur d'un livre, mais pas les livres d'un éditeur



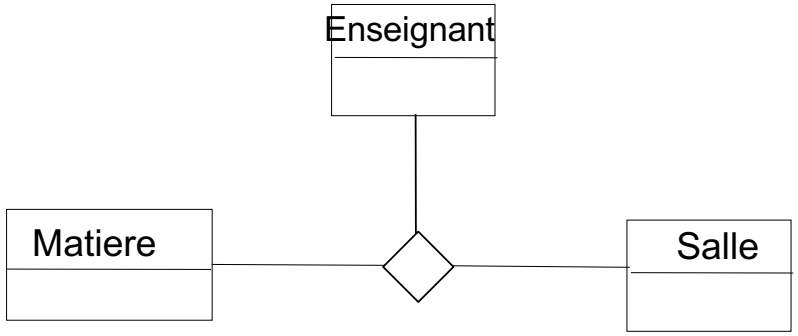
université PARIS-SACLAY

Exemples d'associations

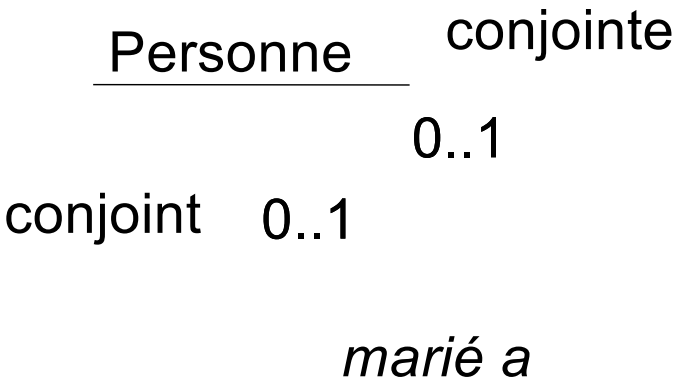
Du point de vue de l'état civil un enfant peut avoir un père ou pas. Un père a au moins un enfant.	
Un bon de commande est émis par un client et un seul.	
Un produit peut n'avoir jamais été commandé par un client, avoir été commandé par un client ou avoir été commandé par plusieurs clients	
Dans un match de tennis simple, participent au moins 2 joueurs et seulement 2.	
Connaissant un article on connaît les commentaires, mais pas l'inverse.	

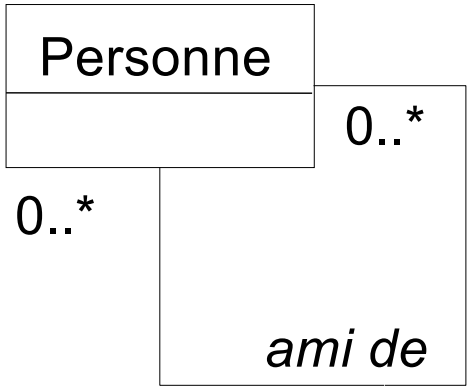
Exemples d'associations rares

- Association ternaire



- Association réflexive

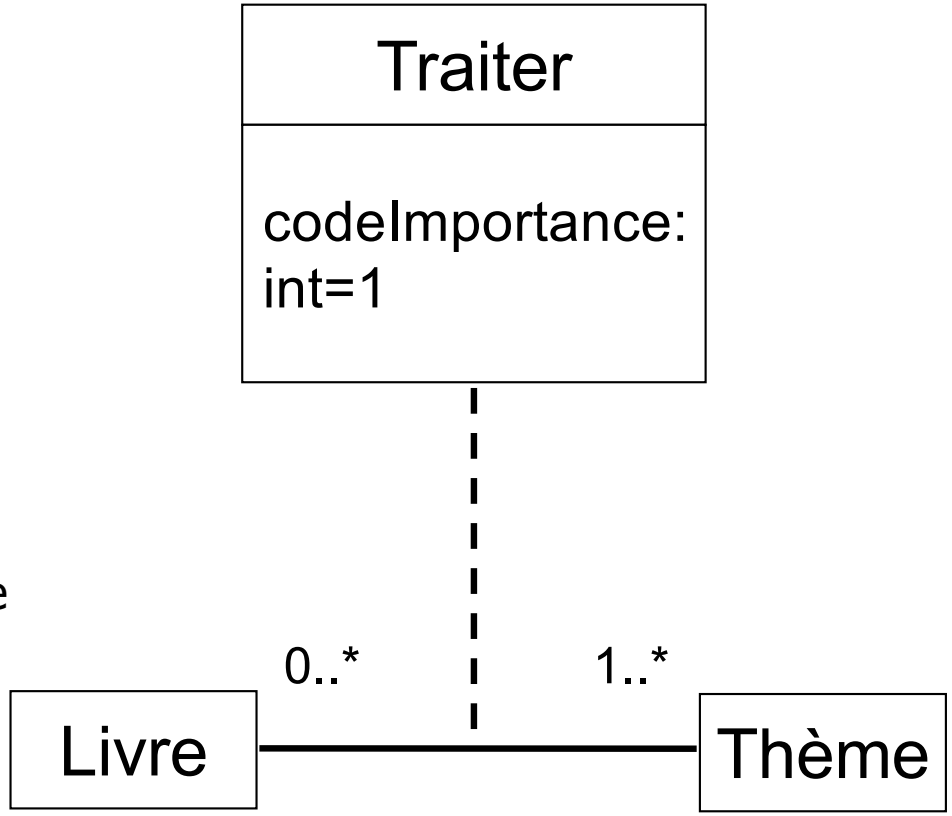




Classe-association

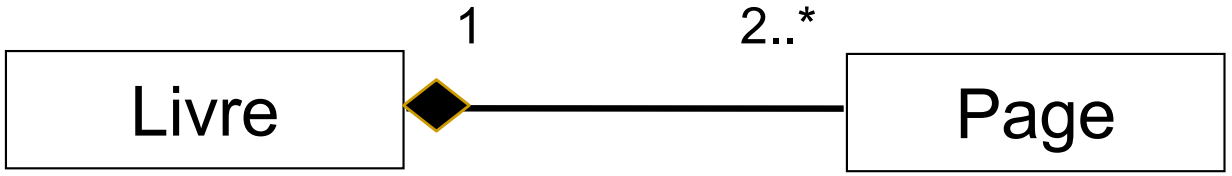
Remarque préliminaire: dans une association entre deux classes, la relation peut posséder certaines propriétés.

- Une **classe-association** est une association dont les propriétés sont regroupées dans une classe
- Exemple: On peut chercher à classer les livres par thèmes.
  - Les livres forment une structure
  - Les thèmes forment une structure
  - **Comment** représenter l'importance relative d'un thème par rapport à un autre?
    - Par un code importance
  - **Où** placer ce code?
    - **Dans l'association qui est « étendue » en classe**



Composition - Agrégation

La relation de **composition** décrit une contenance structurelle entre instances. On utilise un **losange plein**.
La destruction de l'objet composite implique la destruction des parties.
Exemple en Java: seule la classe Livre peut créer et pointer vers la classe Page (Classes internes)



La relation d'**agrégation** décrit l'inclusion de l'instance d'une classe dans l'instance d'une autre classe. On utilise un **losange vide**.
La destruction de l'agrégat n'implique pas la destruction des agrégés.

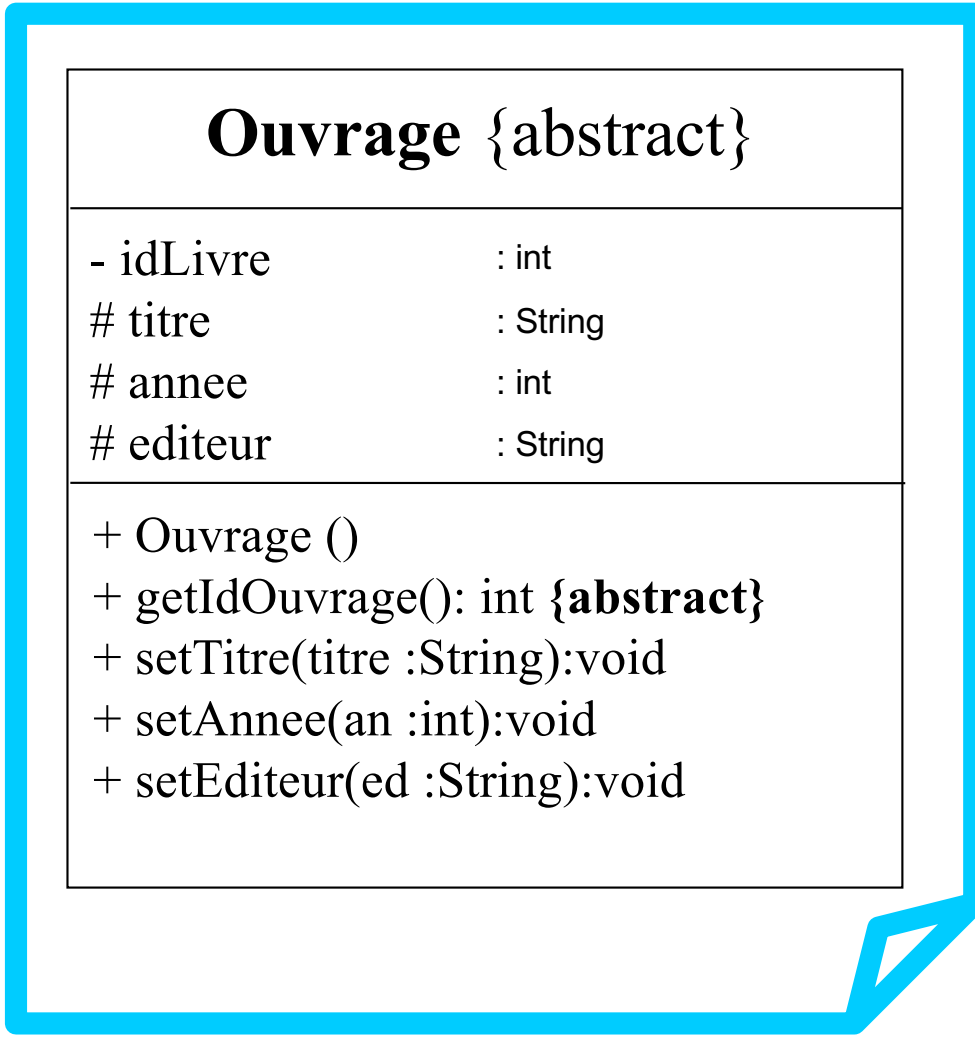
université PARIS-SACLAY

Classe abstraite

Une **méthode** est dite **abstraite** lorsqu'on connait sa signature, mais pas son implémentation. Elle est suivie de mot-clef {abstract}.

Une **classe** possédant une méthode abstraire est **abstraite**, et de ce fait ne peut être instanciée. Son nom est suivi du mot clef {abstract}

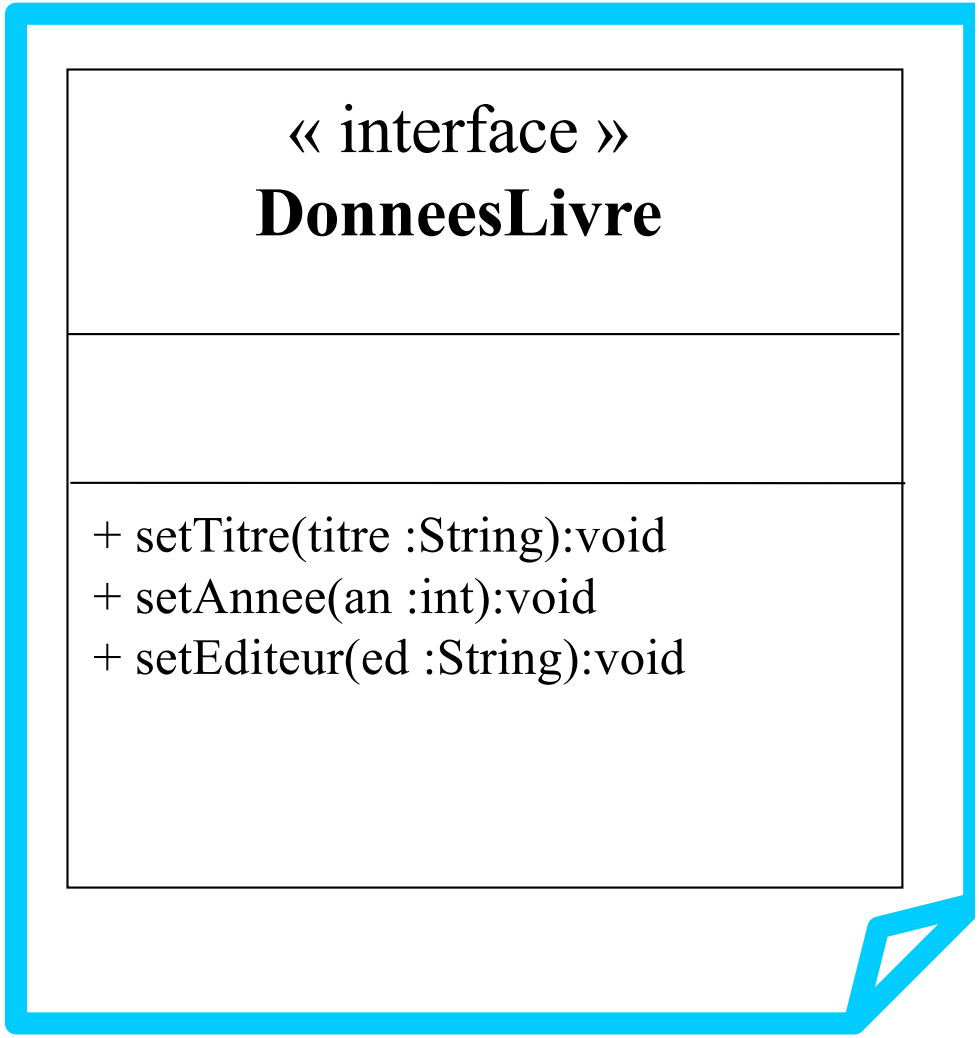
Remarque: en UML2 on peut aussi écrire en italique le nom de la classe et de la méthode abstraite plutôt que le mot clé



Interface

Le rôle d'une **interface** est de regrouper **un ensemble de méthodes** assurant un service cohérent.

Une interface est définie à la façon d'une classe, avec les mêmes compartiments. On ajoute le *stéréotype* "interface" avant le nom de l'interface.



Exemple complet

- Enoncé:

Un livre a un titre. Il est écrit par un auteur ou par plusieurs auteurs dont seuls les noms et prénoms nous intéressent. Un auteur peut avoir écrit plusieurs livres. Le livre est édité par une société d'édition identifiée par son nom. Il possède un nombre de pages défini. Un livre a une date de publication. Le livre peut être emprunté et on désire conserver une trace de l'historique des emprunts c'est-à-dire la date et le client.

1. ***Exercice de révision, représenter le diagramme de cas d'utilisation***
2. ***Représenter le diagramme de classes respectant ces spécifications***

université PARIS-SACLAY

Diagramme de cas d'utilisation

université PARIS-SACLAY

Diagramme de classes (limité aux attributs)

Implémentation en JAVA d'une classe

Livre
- idLivre : int # titre : String # annee : int # dateEd: Date
+ Livre() + Livre(String newTitre, int annee, String ed) + getIdLivre(): int + setTitre(newTitre :String):void + setAnnee(an :String):void + setEditeur(ed :String):void

```
public class Livre {
        private int idLivre;
        protected String titre;
        protected int annee;
        protected Date date;

        public Livre(){…
         }

        public Livre (String titre, int
        annee, String ed)
        {…}

         public int getIdLivre(){
         …
         }
     etc…
}
```

Implémentation de l'association 1-1 : la référence à un autre objet

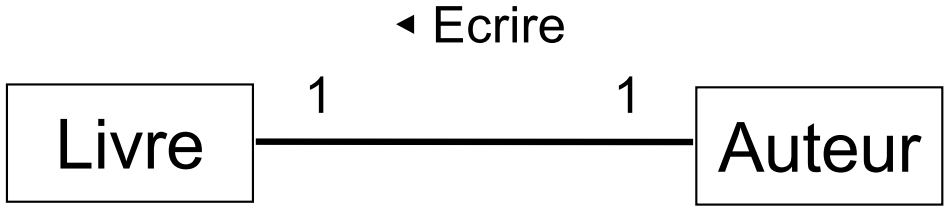
- Comment établir un lien entre deux objets?

  Grâce à la référence de l'autre instance.

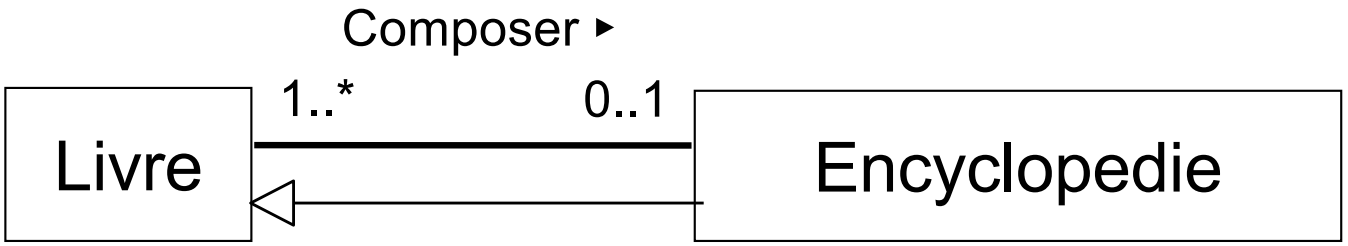
  ⇒ donc il faut créer **une référence** sur l'instance associée en tant qu'**attribut** de la classe.
- Ensuite les objets communiqueront par l'utilisation de leurs méthodes
- Exemple:

```
public class Auteur extends Personne {
        protected Livre livre;
        …
}

public class Livre {
        protected Auteur auteur;
        …
}
```



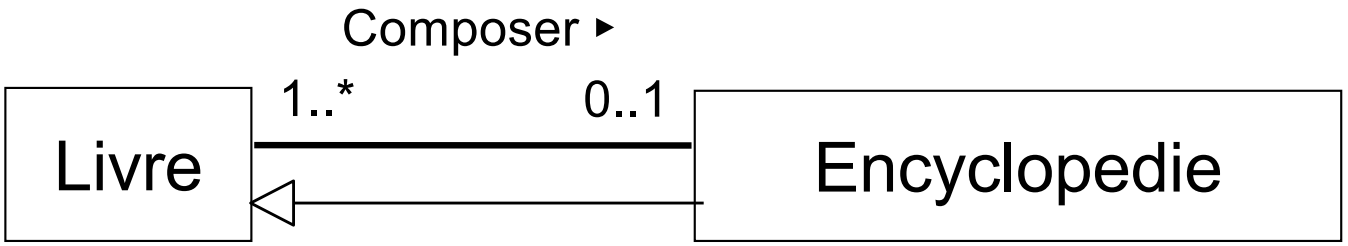
Implémentation de l'association *-* : la référence à un autre objet




```
public class Encyclopedie extends Livre {
        protected Livre[ ] volume;
                                …
    }

public class Livre {
        protected Encyclopedie ency = NULL;
                                …
    }
```

Implémentation de l'association *-* : la référence à un autre objet




```
public class Encyclopedie extends Livre {
        protected Set <Livre> livre = new
        hashset<Livre>();
                                …
    }

public class Livre {
        protected Encyclopedie ency = NULL;
                                …
    }
```

Implémentation de l'association *-* : la référence à un autre objet

Un HashSet<E> est une classe permettant de conserver collection d'éléments de la classe E où chaque élément est *unique* (package java.util).

- L'ordre de classement des éléments peut changer en cours d'utilisation.
- Les méthodes les plus utiles sont : add, remove, contains et size.
- Pour parcourir un HashSet on utilise un objet de la classe Iterator.

Exemple:

```
protected Set <Livre> livre = new HashSet<Livre>();
livre.add(new Livre());

Iterator it = livre.iterator();
while(it.hasNext()) System.out.println(it.next());
```

Diagramme d'objets (ou instances)

- Le diagramme d'objets représente les objets d'un système ***à un instant donné***. Il permet d'illustrer le modèle de classes, en **montrant un exemple** qui explique le modèle

  → *c'est une vue logique permettant la conception logicielle comme le diagramme de classes*

- Syntaxe JAVA création d'objet
  - **Définition:** *nom_classe nom_instance;*

  Exemple: Livre monLivre;
  - **Création:** *nom_instance = **new** nom_classe (); // Constructeur par défaut*
  - **Initialisation des attributs** de l'instance (ou champs de la variable composée):
    - L'opérateur . est le sélecteur de champ *nom_instance.nom_attribut*
    - Exemple: monLivre.idLivre=1120;

Modélisation par schéma UML

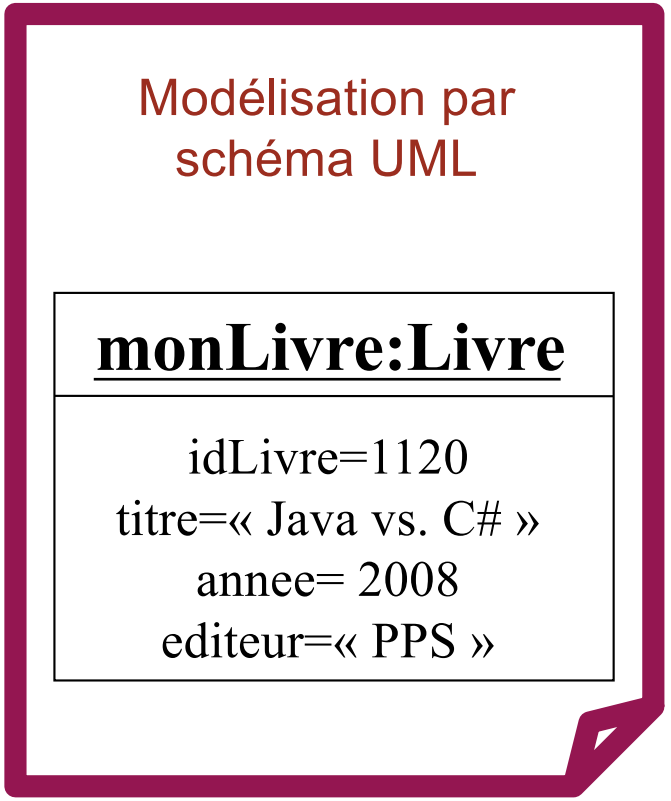
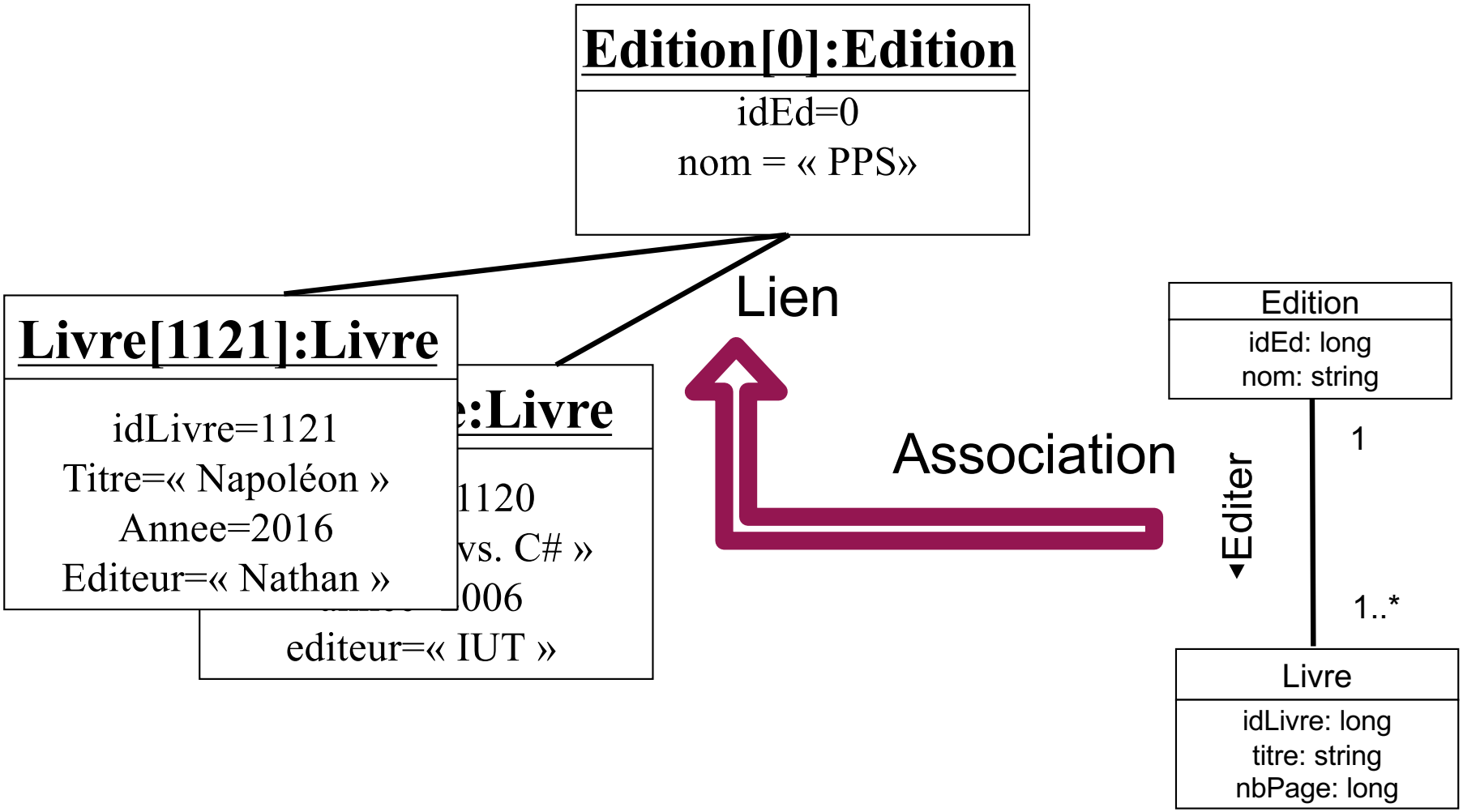


Diagramme d'objets (ou instances)



Le formalisme du Unified Modeling Language

Vue processus : diagramme de communication diagramme de séquence

université PARIS-SACLAY

Vues Processus - Scénario (Conception logicielle)

- En architecture logicielle les vues processus permettent **la conception des méthodes**
- Un **scénario** est un chemin particulier au travers de la description abstraite et générale fournie par le cas d'utilisation. En pratique, on ne décrit que les scénarios les plus représentatifs.
- Quelques diagrammes permettant de représenter des scénarios:
  - diagramme de communication (simplifié)
  - diagramme de séquence (détaillé)
  - diagramme d'état / transition (interne à un objet)

université PARIS-SACLAY

Diagramme de communication

- Les diagrammes de communication montrent les **interactions** entre un ensemble d'**objets** participant à un **scénario** de réalisation d'un cas d'utilisation.
- Ils montrent, et aident à la conception, des **différentes étapes successives** de réalisation sans que les aspects temporels précis des interactions soient rendus explicites:
  - soit parce que ce n'est pas indispensable
  - soit parce qu'il s'agit d'une ébauche

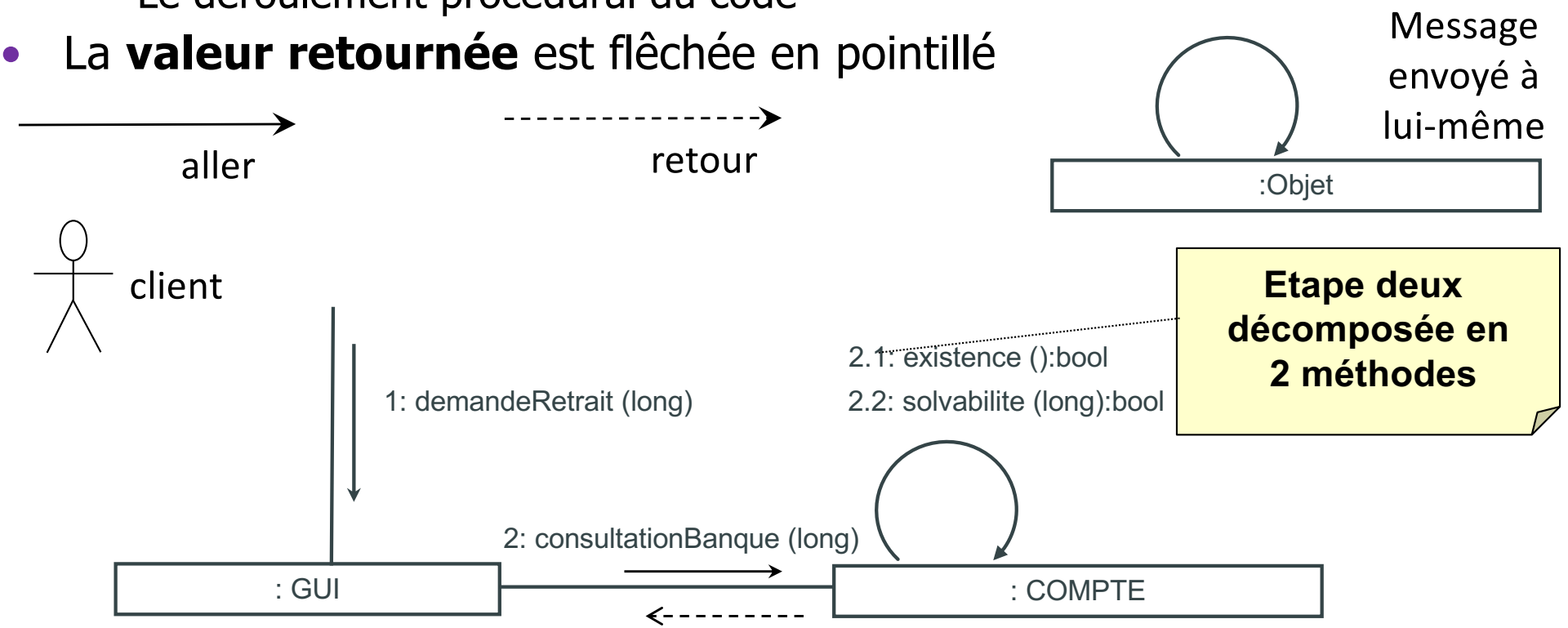
université PARIS-SACLAY

Diagramme de communication

- Les **objets** sont dessinés comme dans le diagramme d'objet, mais sans faire apparaître les attribus.
- Les **liens** correspondent aux **associations** entre classes (implémentées par la *référence* d'un objet dans un autre objet). Ils sont dessinés comme dans le diagramme d'objet, un simple trait.
- Les **actions** entre objets (implémentées par des **appels de méthodes** d'un objet par un autre) portent de nom de **messages** (respecte la syntaxe d'un appel de méthode UML).
- Les messages sont **ordonnés** par la présence d'une **étiquette numérique** dans l'ordre prévu par le scénario

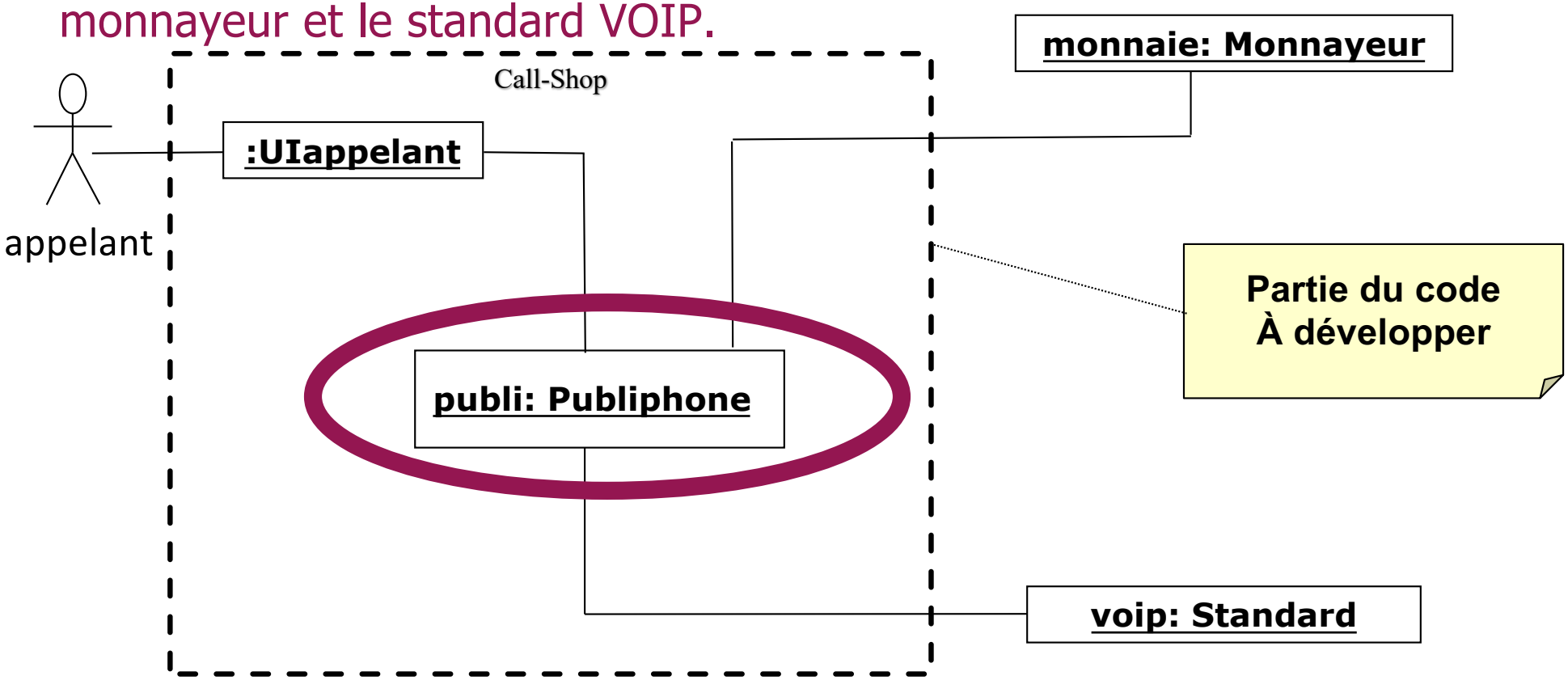
Diagramme de communication

- Les **messages** sont **flêchés** pour indiquer qui l'envoie : c'est un **appel de la méthode** provoqué par:
  - Un **événement** (clic de souris par exemple) ou une interruption matérielle
  - Le déroulement procédural du code
- La **valeur retournée** est flêchée en pointillé



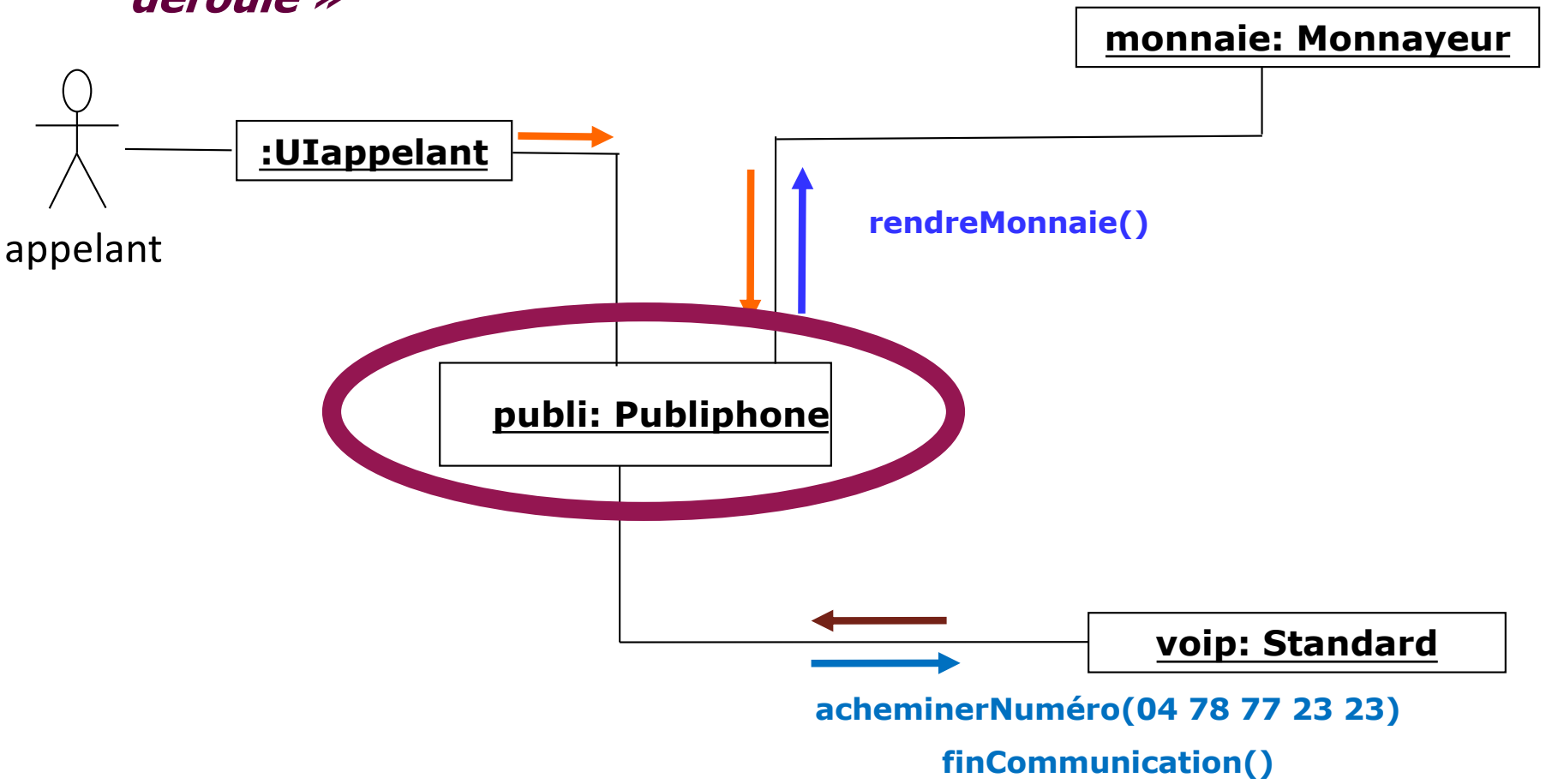
Exercice: CallPhone VOIP

On fait l'hypothèse qu'une application possède deux classes: une appelée UI-appelant, l'autre Publiphone.
En plus de ces classes, on fait apparaître les classes qui proviennent de bibliothèques (ou API) qui servent d'interface logicielle avec le monnayeur et le standard VOIP.



Exercice: CallPhone VOIP

- **En bleu** les méthodes fournies avec les bibliothèques du monnayeur et du système VOIP
- **En orange** les méthodes de la classe Publiphone: ***ce sont celles qu'il faut imaginer pour que le scénario « payer, téléphoner se déroule »***



Exercice: CallPhone VOIP

- Diagramme de communication réalisant le scénario: payer, téléphoner

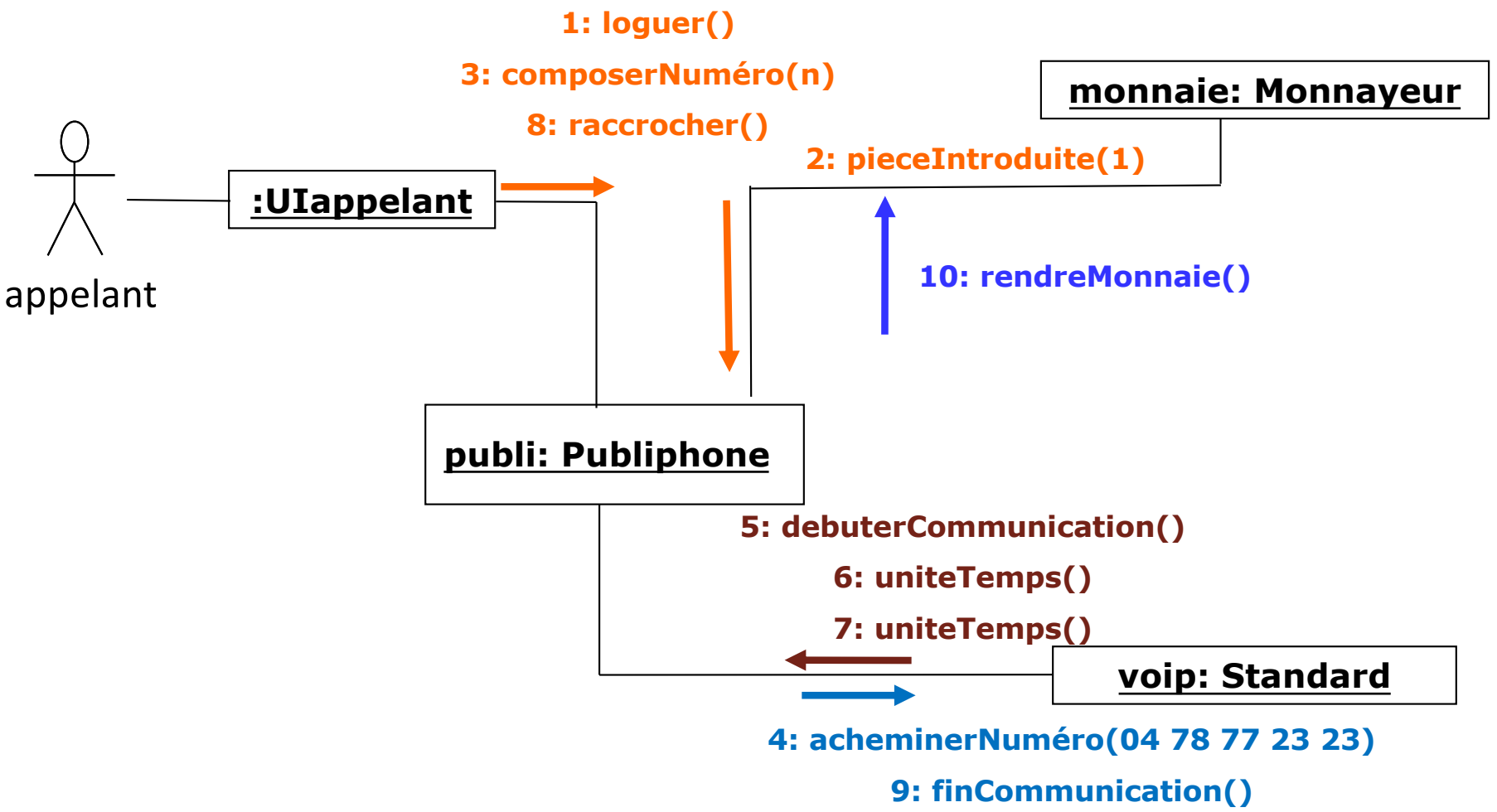


Diagramme de séquence

Les principes sont très proches du diagramme de communication avec en plus deux axes:

- **l'axe vertical** montre le **temps**
- l'axe horizontal montre l 'ensemble objets en interaction dans un scénario.

Un message peut entraîner la création ou la destruction d'objets

Un objet peut s'envoyer un message

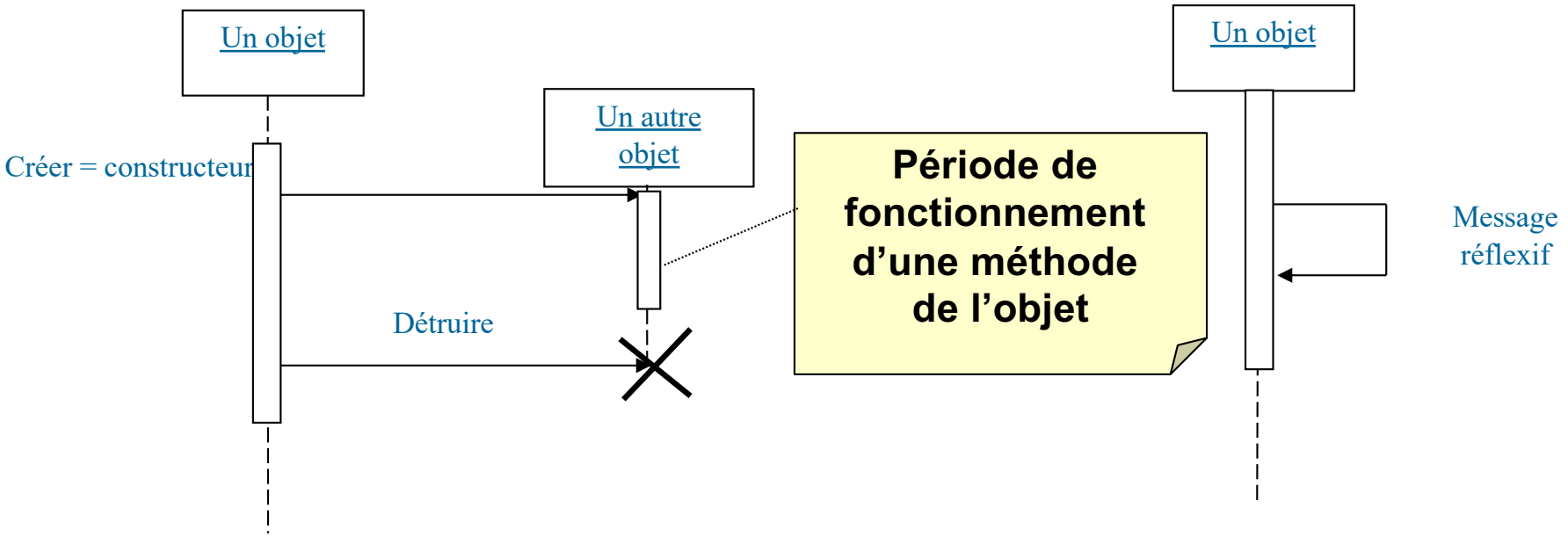
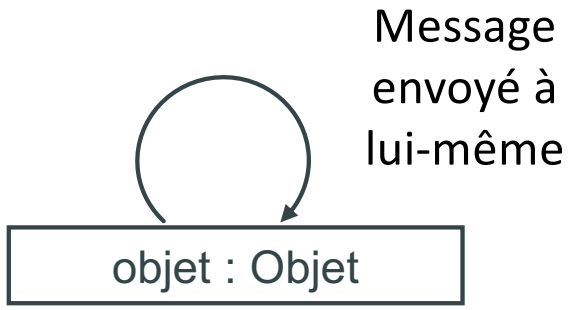


Diagramme de séquence

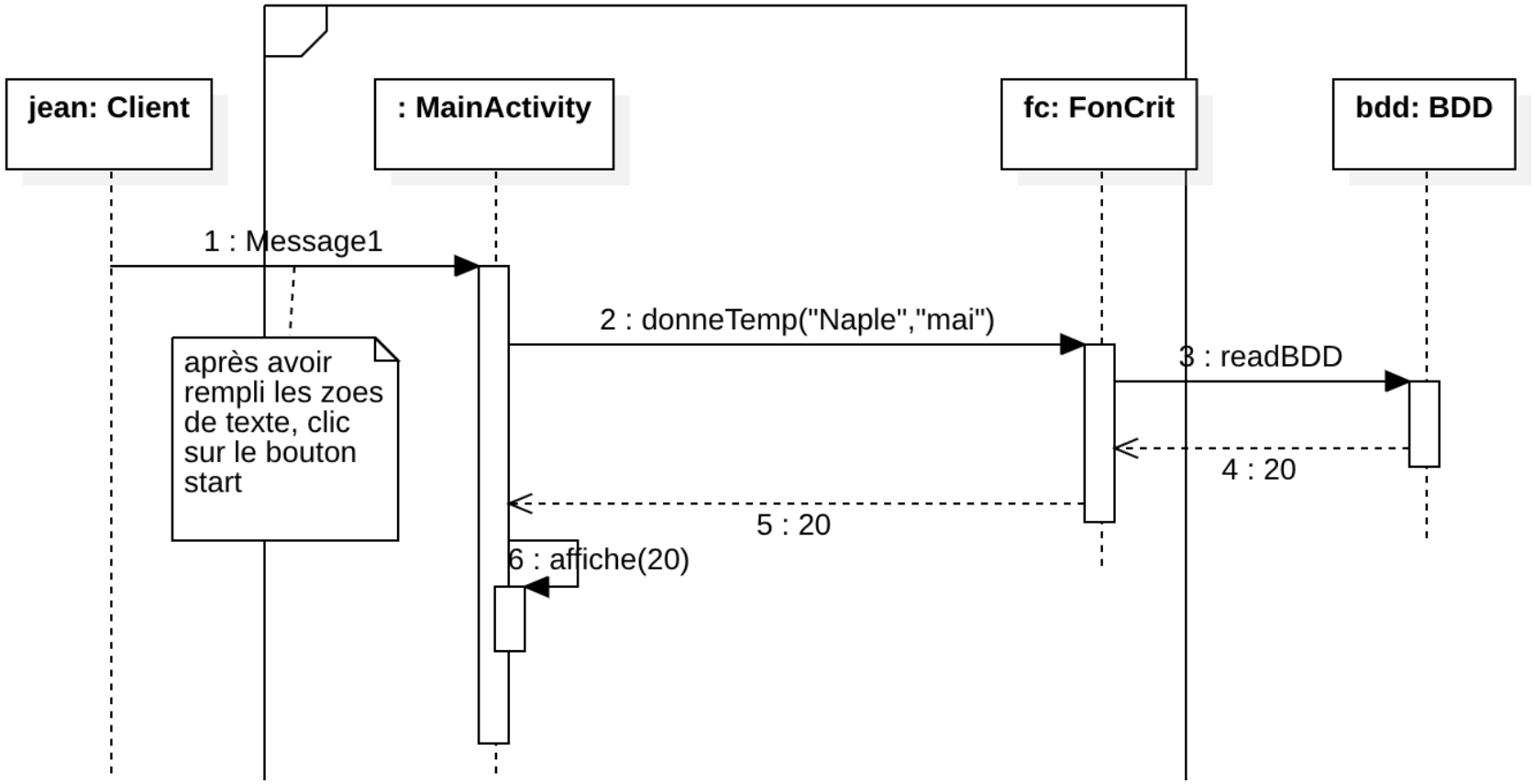
- Les messages sont **flêchés** pour indiquer qui l'envoie : c'est un **appel de la méthode** provoqué par:
  - Un événement (clic de souris par exemple) ou une interruption matérielle
  - Le déroulement normal du code (procédure ou timer par exemple)

- La valeur **retournée** est flêchée en **pointillé**



- La **pointe** de la flèche indique l'objet sur lequel s'applique la méthode: le schéma ci-dessous est équivalent au code **`objet.message()`**

Exemple de diagramme sous StarUML



université PARIS-SACLAY

Exercice: CallPhone VOIP

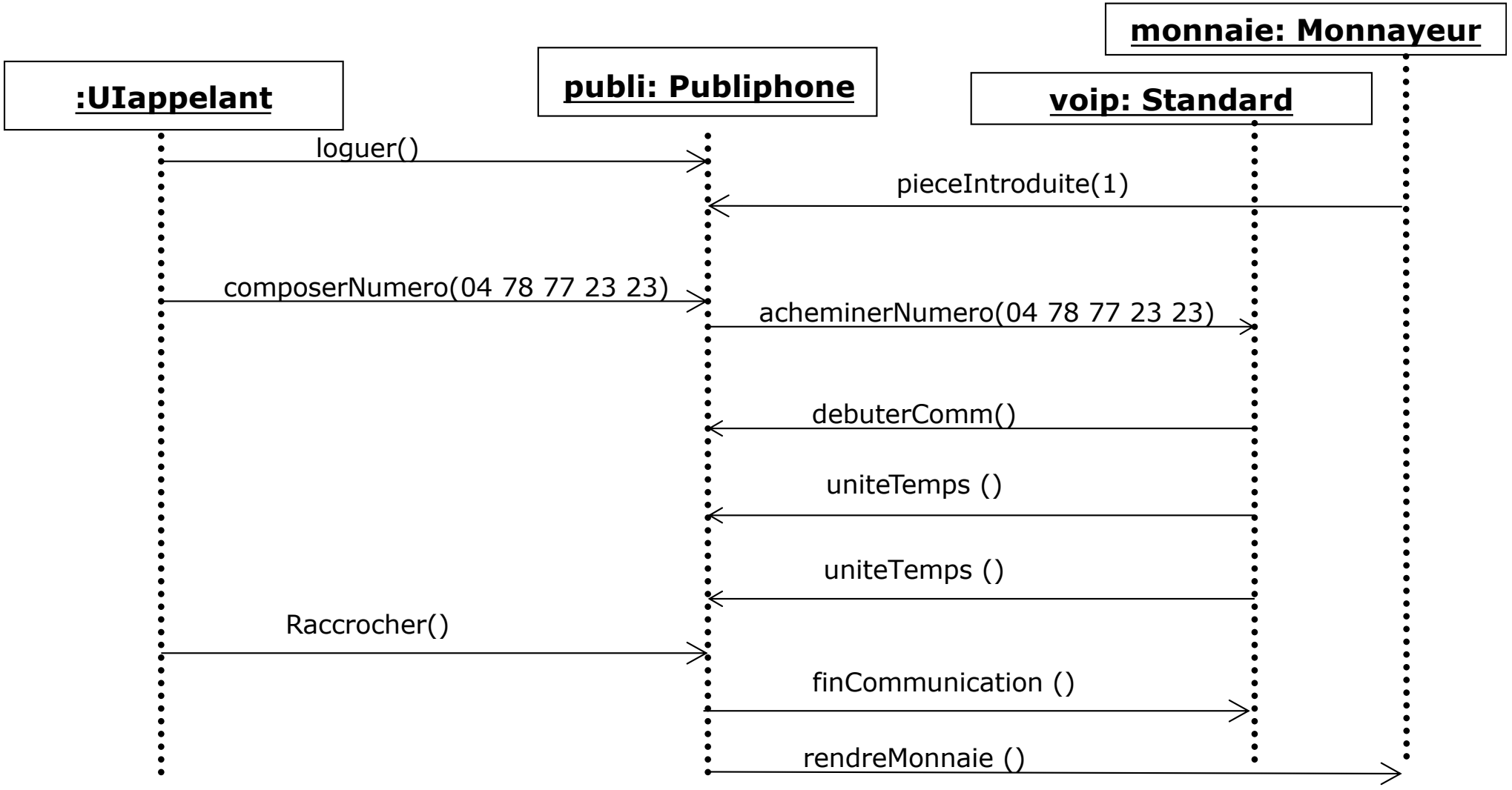
Scénario:

L'utilisateur se logue, puis il introduit une pièce et compose le numéro 0478772323. Une personne répond à l'appel et deux unités de temps s'écoulent. Ensuite l'utilisateur raccroche et récupère sa monnaie.

Représenter le diagramme de séquence

Exercice: CallPhone VOIP

L'activation des objets est omis



université PARIS-SACLAY

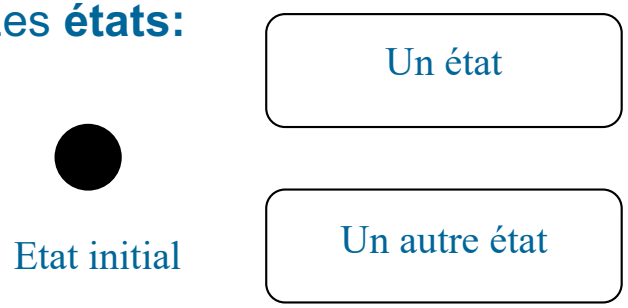
Exercice: CallPhone VOIP

Représenter le diagramme de communication correspondant

Diagramme d'état/transition (conception d'une classe)

Le **comportement d'un objet d'une classe** peut être décrit de manière formelle en termes d'états. Un **état** est un mode de fonctionnement de l'objet

Les **états:**



Une transition:

Evenement ou condition / méthode (param)

Une **transition** peut-être causée par:

- **Un événement (**par exemple un clic de souris,…)
- **Une condition** (valeur supérieure à…, une durée s'est écoulée)

  Si c'est une condition **logique** (renvoie VRAI) alors elle est écrite entre **[…]**

On peut faire suivre cet ensemble du nom de la méthode appelée

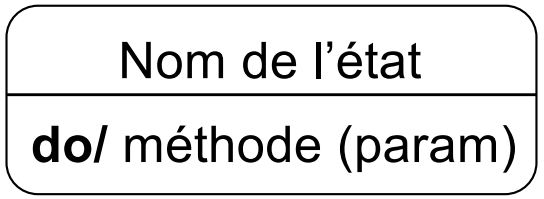
université PARIS-SACLAY

Diagramme d'état/transition (conception d'une classe)

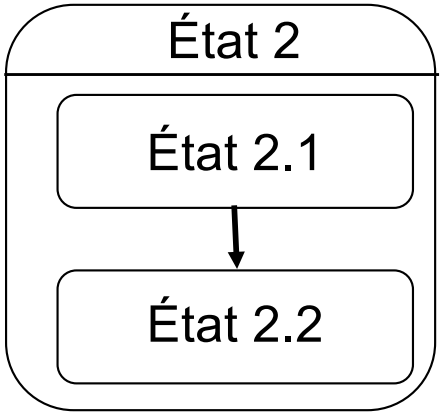
On *peut (pas une obligation)* indiquer dans l'état les **activités** exécutées.

Une activité est une série d'actions, généralement correspondant à une **méthode**

- entry/ = actions à l'entrée dans l'état
- exit/ = à la sortie de l'état
- do/ = pendant l'état

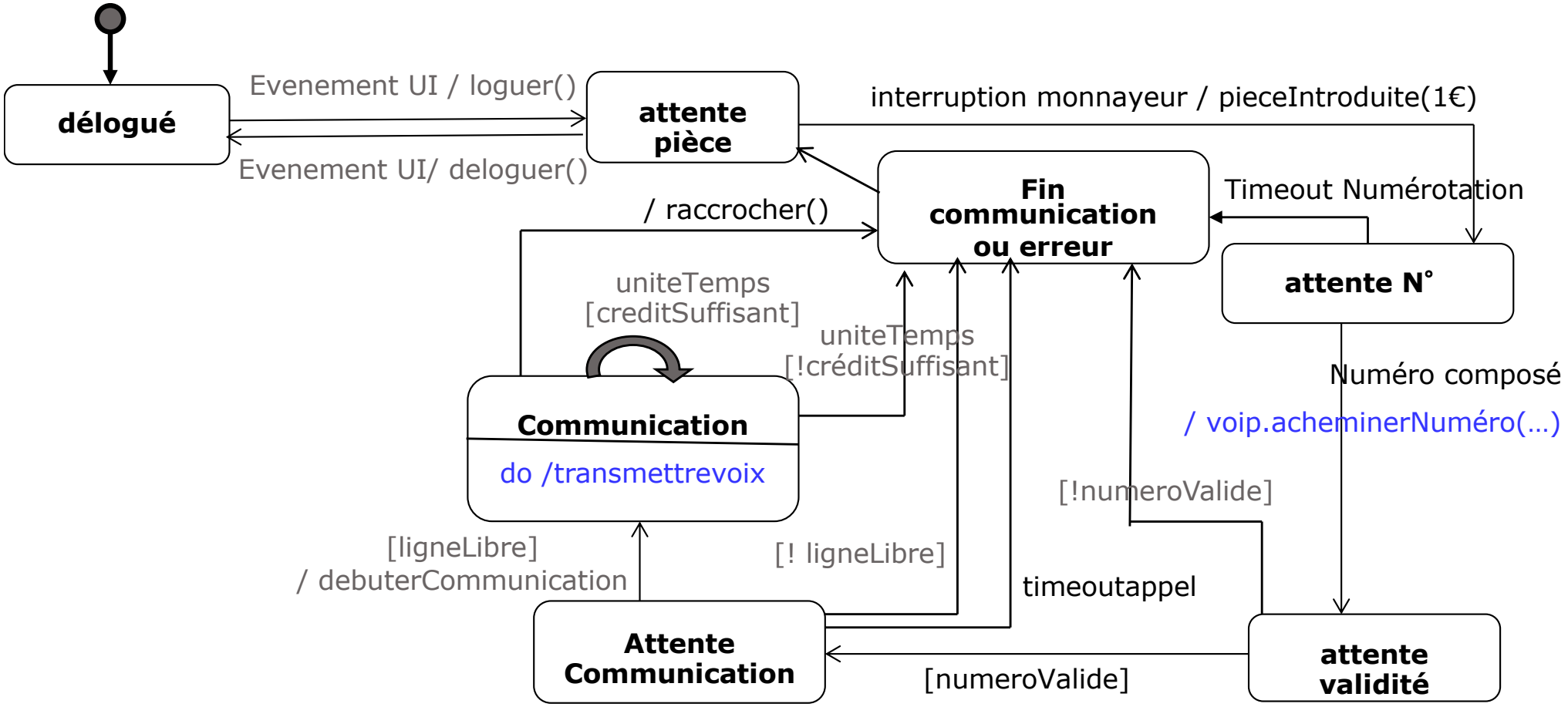


Un état peut contenir des **sous-états**



Exemple: client léger pour Callshop

Diagramme d'états de l'objet publi: Publiphone



université PARIS-SACLAY

Exercice: client léger pour Callshop

Décrire ci-dessous le sous-état « fin de communication ou erreur » par un diagramme d'état