Conception Orientée Objet

Domain Drive Design (DDD)

Tianxiao LIU
Master 1 IISC 1ère Année
CY Cergy Paris Université
http://depinfo.u-cergy.fr/~tliu/coo.php

Sommaire

- Problématique et besoin
- Les principes fondamentaux de DDD
- Langage ubiquitaire
- Entité
- Objet de valeur
- Agrégat et racine d'agrégat
- Service
- Factory
- Repository
- Bounded context

Problématique et résolution

- Problématique
  - Modélisation du domaine de l'application → cruciale
  - Très peu de travaux de recherche à ce sujet
  - Peu de personnes savent comment le faire correctement

- DDD : *Domain Driven Design*
  - Eric EVANS : proposition de DDD
  - Ses expériences : ses succès et ses « échecs » → enrichissant

Analyse du besoin et clés de la réussite

- Besoin du DDD
  - Système logiciel complexe
  - Domaine commercial ou technique
  - Développement orienté objet

- Clés de la réussite « communes »
  - Un modèle de domaine riche
  - Modèle évolutif par itérations de conception

Complexité : succès et échec

Autres facteurs pouvant influencer le projet

- Mauvaise organisation
- Objectifs flous
- Manque de ressources

…

Concentration sur les techniques

- Quand on parle de la conception
  - Beaucoup d'efforts fournis sur les détails suivants
    - Librairies utilisées
    - Bases de données à choisir
    - Protocoles réseaux
    - …
  - Beaucoup de publications à ces sujets → développeurs très compétents en ces technologies

- Mais ces technologies ne sont pas la résolution de la complexité du domaine

Principes fondamentaux de DDD

- Domaine d'application
  - Sa logique mérite une attention particulière
  - Sa conception se concrétise par un modèle central → cela reflète à sa complexité

- Esprit de DDD
  - Une façon de penser
  - Un ensemble de priorités visées
  - Amélioration de la procédure de la fabrication du logiciel → résolution de la problématique de complexité (évolutive)

Définition du domaine

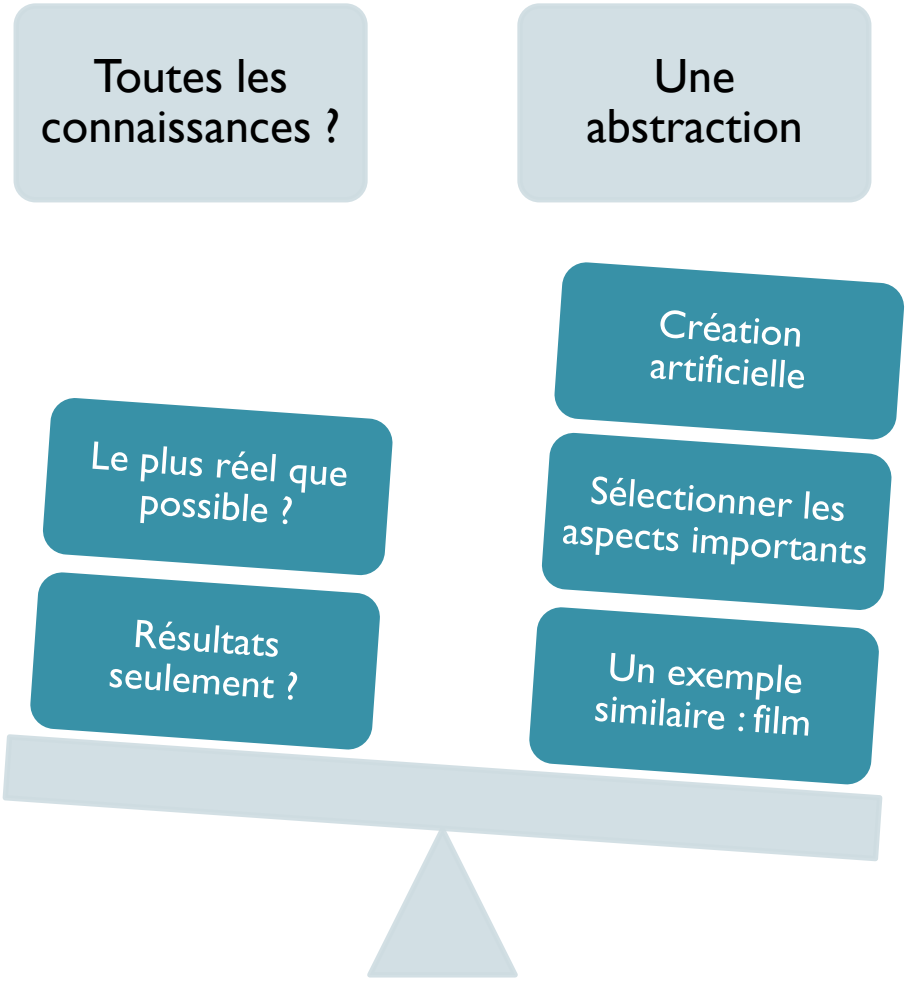
- Un bon logiciel
  - Une grande valeur ajoutée dans les activités quotidiennes de l'utilisation
- Un grand ensemble de connaissances liées ces activités
  - Cela peut déjà « intimider » certains développeurs
  - Grand volume, complexité difficile à maîtriser
- On a besoin d'un modèle
  - Un modèle : une forme simplifiée et structurée de connaissances nécessaires → liés à un problème

Modélisation du domaine

On peut faire un diagramme, pourtant…

- Pas un diagramme en particulier
- L'idée exprimée par le diagramme est plus importante

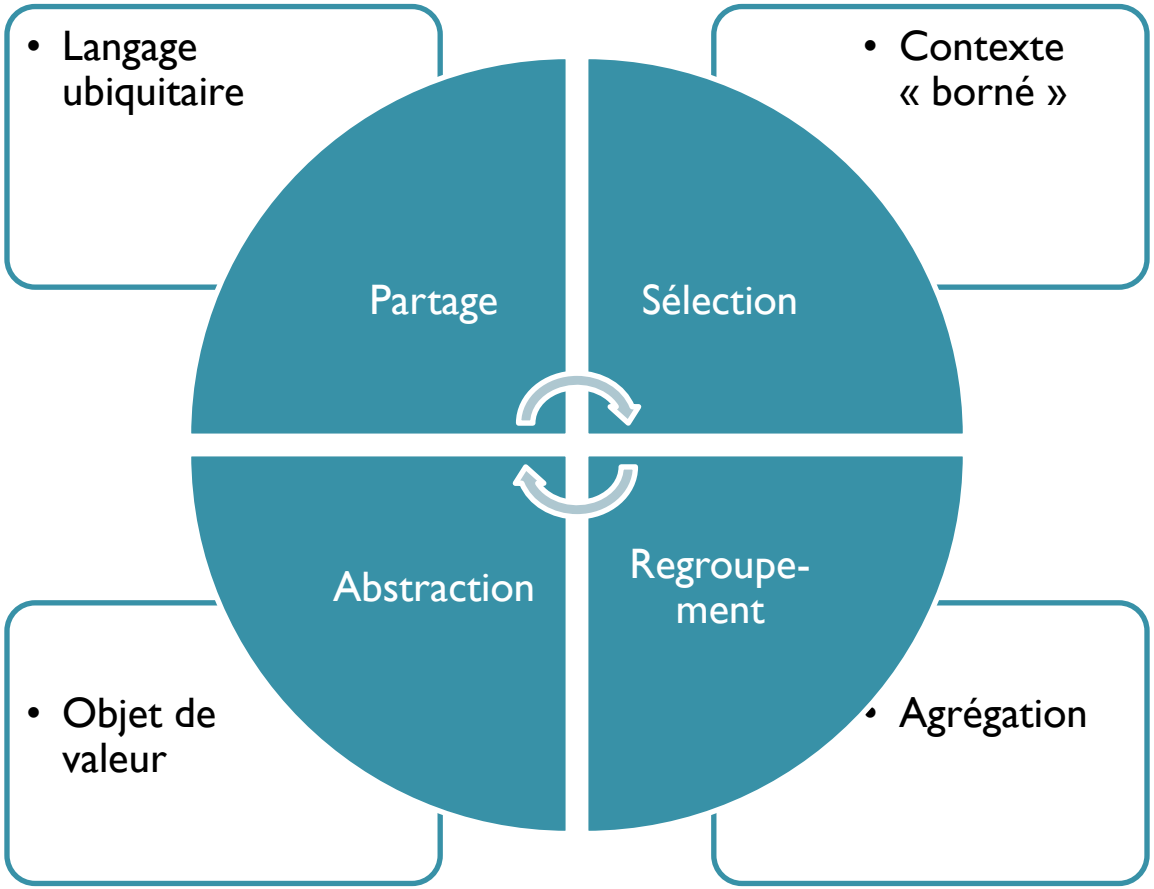
…



« Colonne vertébrale » du langage utilisé

- Le modèle de domaine
  - Utilisé et partagé par tous les membres de l'équipe
  - On peut parler de l'implémentation avec les mêmes notions
  - Le plus important : les experts du domaine utilisent les mêmes notions de ce modèle → Pas besoin de traduction
- Une collaboration étroite entre développeurs et clients
  - Gestion de projet moderne → Agilité

DDD : un aperçu des notions importantes



Langue ubiquitaire

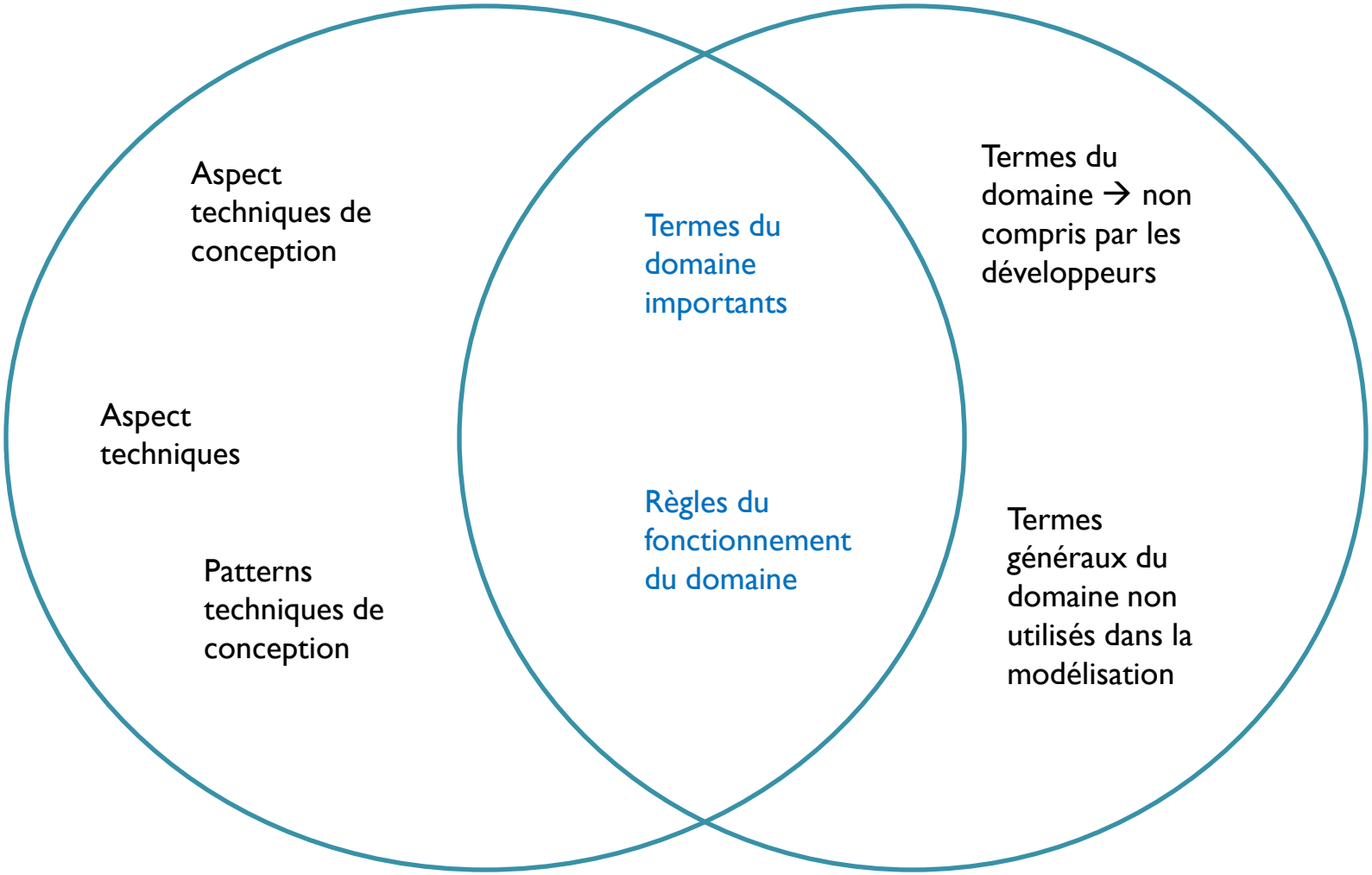
- Problématiques
  - Différents langages et vocabulaires utilisés par différents personnes : experts domaine, développeurs, maître d'ouvrage…
  - Parfois c'est pire : entre différents (groupes de) développeurs
  - Coût supplémentaire en traduction et mauvaise compréhension
- Un langue commun
  - Un effort de tout le monde
  - Fondamental pour une conception du domaine

Langue ubiquitaire

- Vocabulaire du langage
  - Termes correspondant aux noms des classes et des attributs
  - Termes définissant les règles du fonctionnement (verbes) : les relations entre les classes et les méthodes

- Conception DDD : fournir un modèle
  - Combinaisons de règles : mots, phrases…
  - Ce n'est pas seulement un élément de conception
    - Expression des besoins → échanges
    - Les tâches de développement de fonctionnalités

Langue ubiquitaire



Entités (*Reference objects*)

- Pour certains objets
  - Ils ne sont pas définis par leurs attributs.
  - Ils représente un « fil d'identité »
  - Evolutif entre différentes représentations
- Entité : un objet défini essentiellement par son identité
  - Cycle de vie évolutif → changement de forme ou de contenu
  - La continuité doit être garantie.
  - Ne pas confondre avec le concept « equals » en Java

Entités : un exemple concret

- Application de réservation de billets dans un stade
  - Premier cas
    - Chaque billet correspond à un numéro du siège (unique dans le stade)
    - Siège → une entité
    - Numéro (ou numéro de ligne et celui de colonne) → identité
  - Deuxième cas
    - Billet → admission générale
    - Pas de distinction individuelle des sièges
    - Siège → n'est pas une entité même s'il y a un attribut numéro

Objet de valeur (*Value objects*)

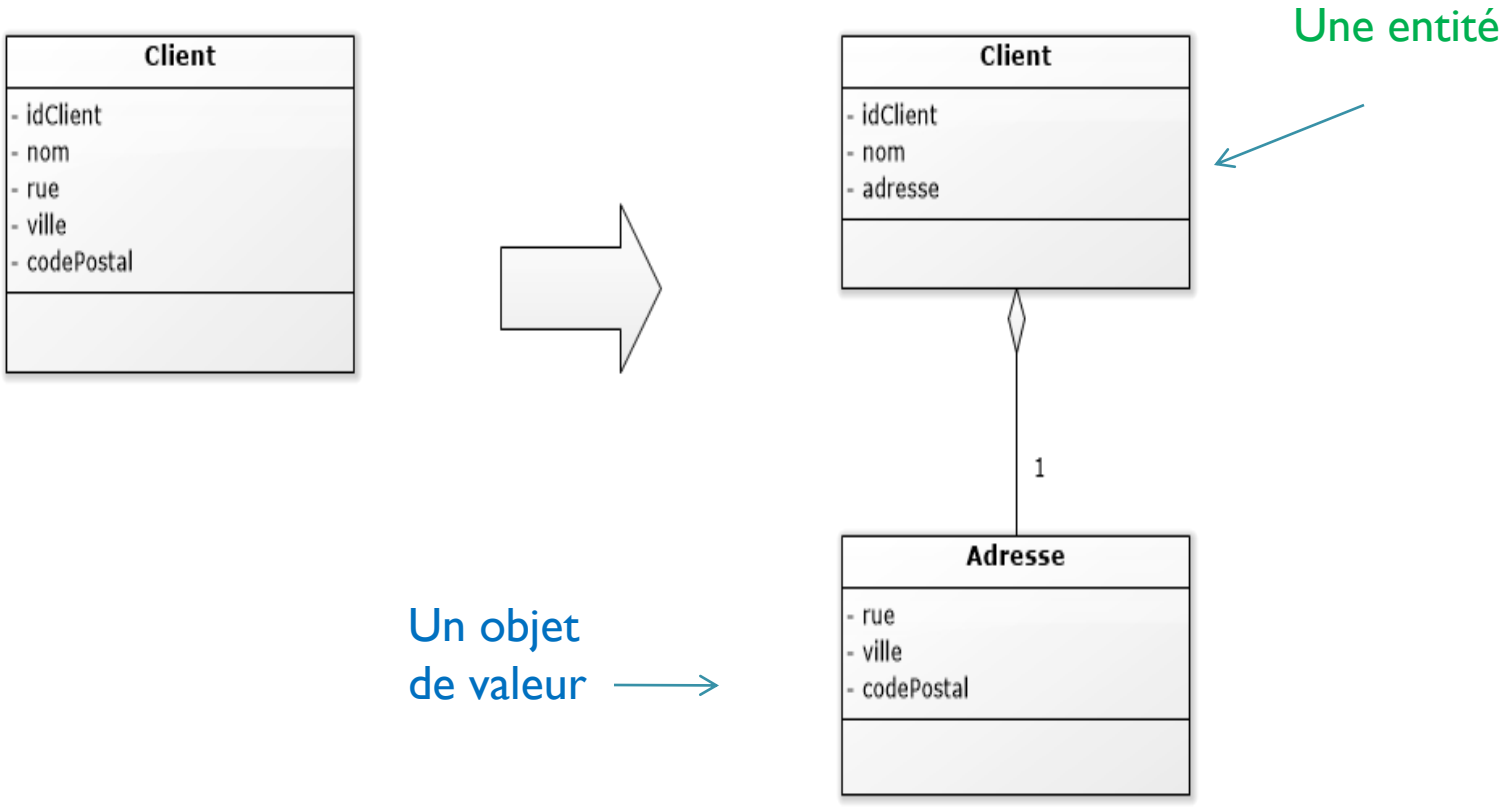
- Pour certains objets
  - Pas d'identité conceptuelle
  - Description des caractéristiques d'une chose

- Un exemple
  - Dessin des enfants en utilisant différents crayon en couleur
  - Indifférent : quel trait est fait par quel crayon

Objet de valeur

- Définition
  - Un objet qui représente un aspect descriptif du domaine sans identité conceptuelle
  - On s'intéresse à ce que c'est, non à la question « quel objet »
- Un exemple
  - Gestion des couleurs dans les langages de programmation
- Quelques cas pratiques
  - Un objet de valeur peut être composé d'autres objets
  - « Transient » : souvent passé comme paramètre (message) entre différents objets → créé pour une opération et ne sera plus utilisé

Objet de valeur : un exemple concret

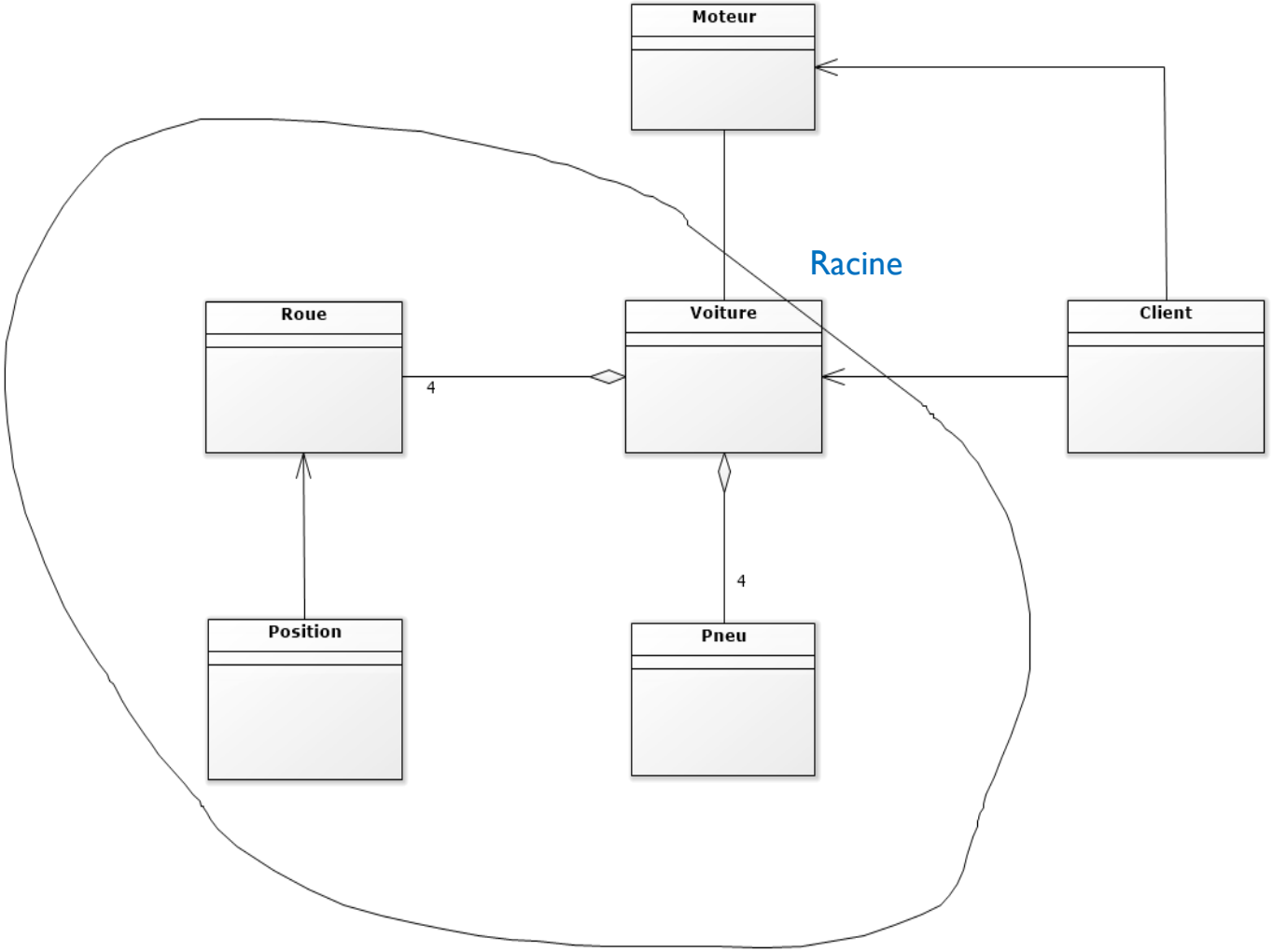


Agrégat (*Aggregate*)

- Objectif
  - Minimiser les relations (associations) non nécessaires
  - Garantir la consistance de (changement de) objets

- Un agrégat
  - Un cluster d'objets associés à traiter ensemble en termes de changement de données
  - Racine d'agrégat : une entité dans l'agrégat exposée à l'extérieur
  - Frontière : définit ce qui est dans l'agrégat

Agrégat : un exemple concret



Services : description

- Ici, on ne parle pas de micro-services.
- Description d'un service
  - Une opération offerte
  - Distingué dans le modèle du domaine
  - Sans état encapsulé →différent que Entité et Objet de Valeur
- Relation entre les objets du domaine
  - Nom d'un service : une activité (un verbe)
  - Cohérence avec le langage ubiquitaire

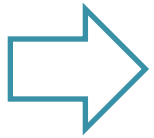
Services : principes de conception

- Sans état
  - Pour utilisateur (programmeur) du service, pas besoin de considérer l'historique de l'instance du service
  - Exécution du service
    - Besoin d'informations qui sont accessibles globalement
    - Effet de bord, dans certains cas, informations globales modifiées par l'exécution du service
- Conception d'un service
  - Dans une classe (ou interface) isolée
  - Granularité du service → ne pas exposer les détails du domaine

Factories

- Création d'objet (ou agrégat) → encapsulation
  - Responsabilité
    - Les objets du domaine sont sensibles
    - Assembler les objets n'est pas la tâche du client (utilisateur des objets du domaine) → Même appel d'un constructeur : déconseillé
    - Besoin d'une classe séparée pour la création d'objet

~~Le client spécifie ce qu'il veut : les paramètres~~



La factory crée l'objet qui satisfait le besoin du client, en respectant les règles intérieures

Factories : conception et mise en œuvre

- Solutions possibles
  - Simple factory
    - Pour créer les objets de la même famille (variantes)
  - Builder
    - Pour créer les objets complexes qui ont des associations vers d'autres objets du domaine
- Méthode et classe
  - Méthode : atomique → consistance
  - Placer la factory auprès de la classe racine de chaque agrégat

Repositories :principes

- Pour les objets créés
  - Où les stocker ? Comment les récupérer ?
  - Façon d'accès : accès direct vs accès via associations
- Référence d'objet
  - Ne pas confondre avec l'identifiant d'objet (Entité ou Objet de Valeur) → Ici, on parle de la mise en œuvre (programmation)
- Effets de bord sur la conception
  - Objets créés → Objets répertoriés → Accès facile
  - Pas d'hésitation pour créer les associations entre les objets

Repositories : conception et recherche

- Accès aux objets d'un agrégat
  - Via l'objet racine
  - Il est toujours préférable, si possible, de commencer par les objets du type Entité
- Recherche des objets
  - Besoin de formuler les méthodes sous forme de « requêtes »
    - Définir les paramètres de recherche
    - Résultat de la recherche : quel objet → accès direct ou indirect ?

Bounded context : principes

- Projet complexe
  - Différents contextes : souvent différentes équipes de travail sur différentes features du système
  - Multiple modèles
    - Chaque modèle pour un contexte d'application
    - Chaque modèle : une perspective du domaine pour le contexte
- Défi
  - Réutilisabilité du code → confusion
  - Liaison logique entre les programmes de différents contextes → inévitable → besoin de traduction (mapping)

Bounded context et context map