Architectures d'application

DUT M3105 : Conception et programmation objet avancées

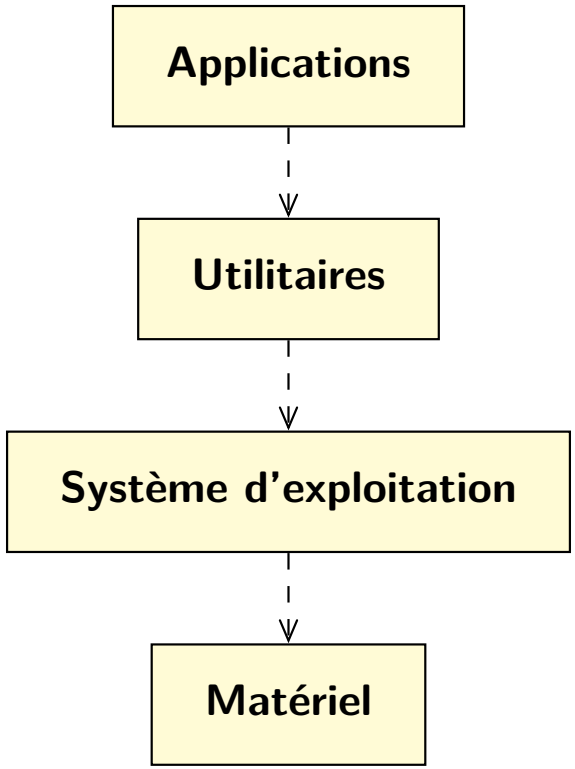
Serge Rosmorduc
serge.rosmorduc@lecnam.net
Conservatoire National des Arts et Métiers

2018–2019

Architecture en couches

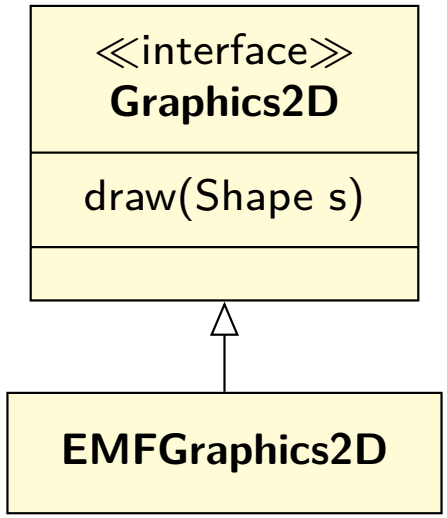
Definition

- sous systèmes empilés ;
- chaque composant est en relation uniquement avec ceux du sous système immédiatement au dessous.
- principe très très souvent utilisé en architecture...



Un petit exemple (un peu technique)

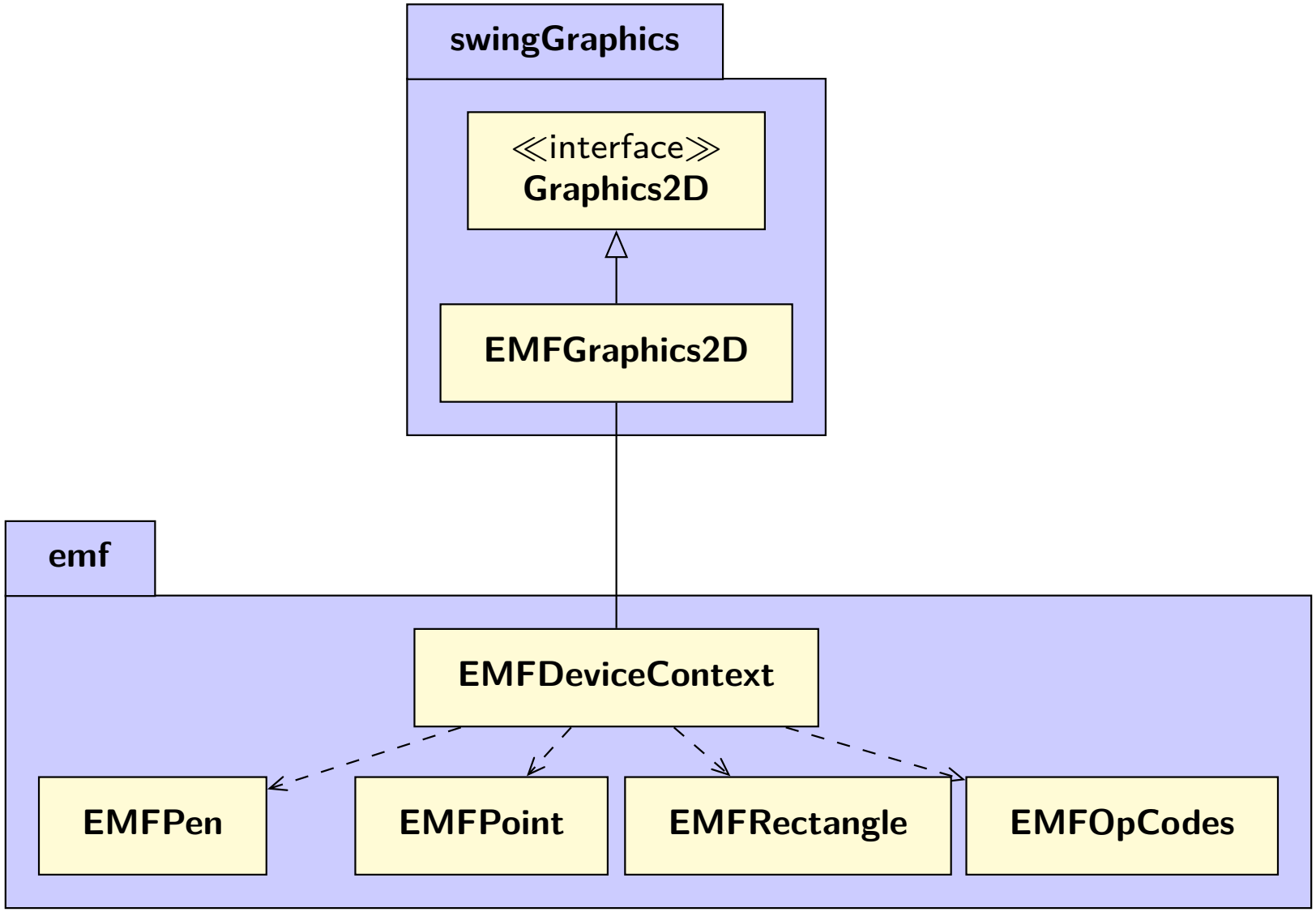
Pour produire des fichiers EMF (un format graphique de Windows en Java, je voulais écrire une implémentation de `Graphics2D`) :



Mais :

- le code de génération du format EMF est complexe ;
- chaque méthode de Graphics 2D peut produire des données EMF complexes...

D'où l'architecture :


swingGraphics
≪interface≫
Graphics2D
EMFGraphics2D
emf
EMFDeviceContext
EMFPen
EMFPoint
EMFRectangle
EMFOpCodes

Conséquence...

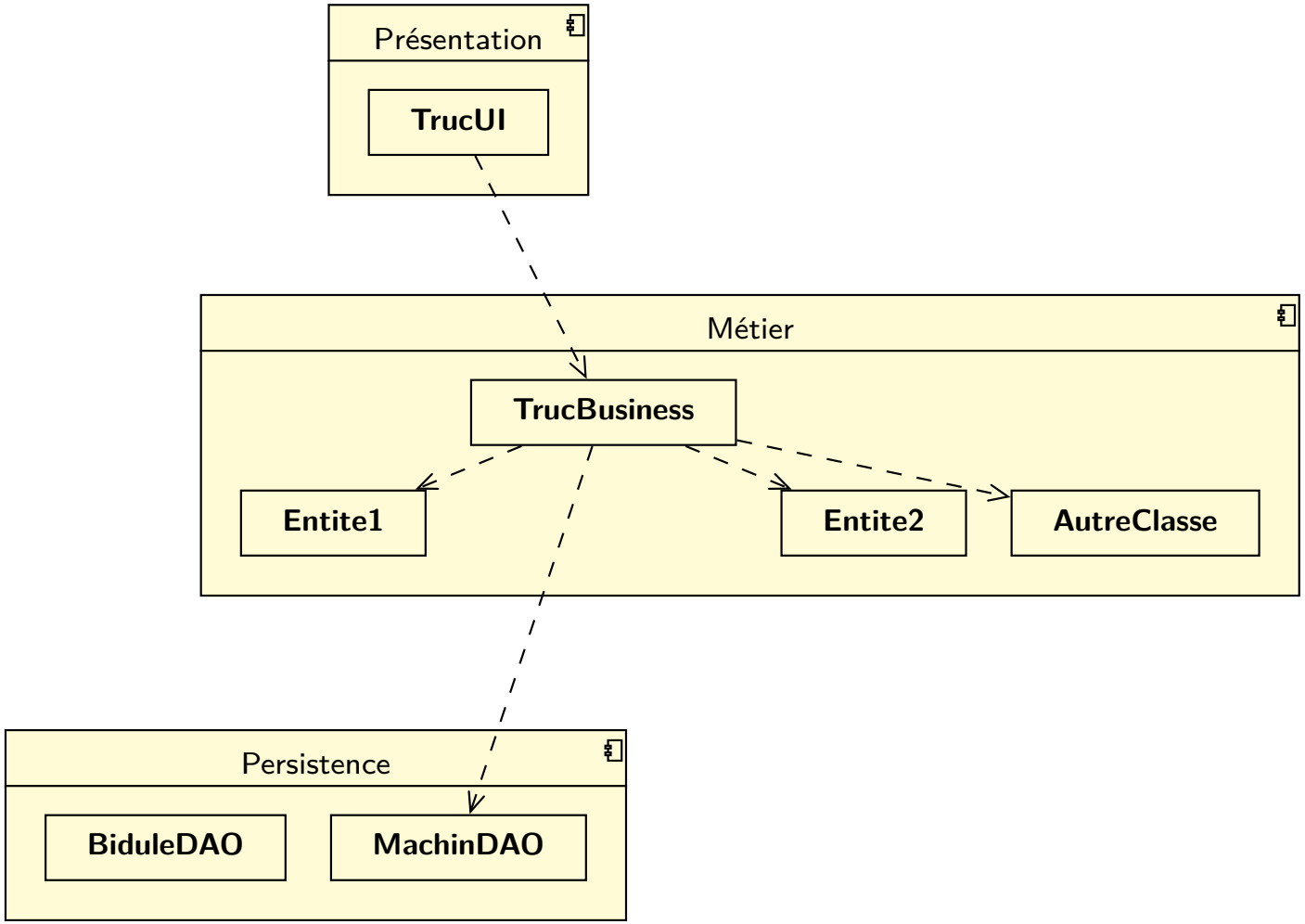
On évite de gérer dans la même classe la transformation des appels Graphics2D en appels EMF *et* l'écriture des données au format EMF.

- EMFGraphics2D raisonne en interne en termes d'appels aux commandes graphiques de EMFDeviceContext
- EMFDeviceContext gère les problèmes liés au format de fichier interne des fichiers EMF (taille des données, codes compris par EMF, etc.)

Si vous êtes curieux : `https://github.com/rosmord/jsesh/tree/master/jvectClipboard/src/main/java/org/qenherkhopeshef/graphics/emf`

Architecture 3-tiers (trois couches)

pas à proprement parler un pattern, mais il faut en parler quand même...



Data Access Object

Motivation

Permet d'isoler les technologies de persistence (souvent le code SQL) du reste du programme.

On crée des classes DAO qui implémentent CRUD :

CRUD

- **C**reate : crée une nouvelle entrée ;
- **R**retrieve : récupère un ou plusieurs objets ;
- **U**pdate : met à jour un objet modifié ;
- **D**elete : supprime une entrée de la base.

- typiquement, une DAO par classe entité ;
- les DAO sont très bêtes : **pas de logique métier dans la DAO.**
- on lit, on écrit, c'est tout ;
- connexion/transaction : a priori externes à la DAO.

DAO

```
// DAO à créer à la demande et à ne pas conserver.
class PersonneDAO {
  private Connection db;
  public PersonneDAO(Connection db) {...}
  public void create(Personne p) {...}
  public void update(Personne p) {...}
  public void delete(Personne p) {...}
  public List<Personne> findAll() {...}
  public List<Personne> findByName(String name) {...}
  public Personne findById(long id) {...}
}
```

DAO – remarques

- create : peut *éventuellement* fixer l'identifiant de l'objet ;
  - soit en modifiant l'objet passé en paramètre ;
  - en renvoyant l'identifiant comme résultat ;
  - en renvoyant un nouvel objet ;
  - on peut (voir TP) passer aussi les données nécessaire à la création, et pas l'entité elle même.
- update : pour les objets avec des dépendants (exemple : l'objet Contrôle dans le TP), si ceux-ci sont en petit nombre, on peut les détruire et les recréer plutôt que de chercher ceux qui ont été modifiés.
- delete : peut prendre aussi comme argument l'ID de l'objet.

DAO – Remarques

Pour *retrieve*, on a en fait une méthode par famille de requêtes ;

- findById : renvoie un objet unique dont on connaît l'ID (ou un `Optional` de java 8) ;
- findAll : renvoie tous les objets ; on peut aussi envisager de paginer (avec un numéro de page et une taille de page passés en paramètres) ;
- findXXX : on peut imaginer de très nombreuses autres requêtes ;
- il est possible de renvoyer des objets complexes (voir TP), mais c'est compliqué ;
- alternative : JPA.