Manipulation de modèles et métamodèles

Naviguer dans un modèle

Points abordés

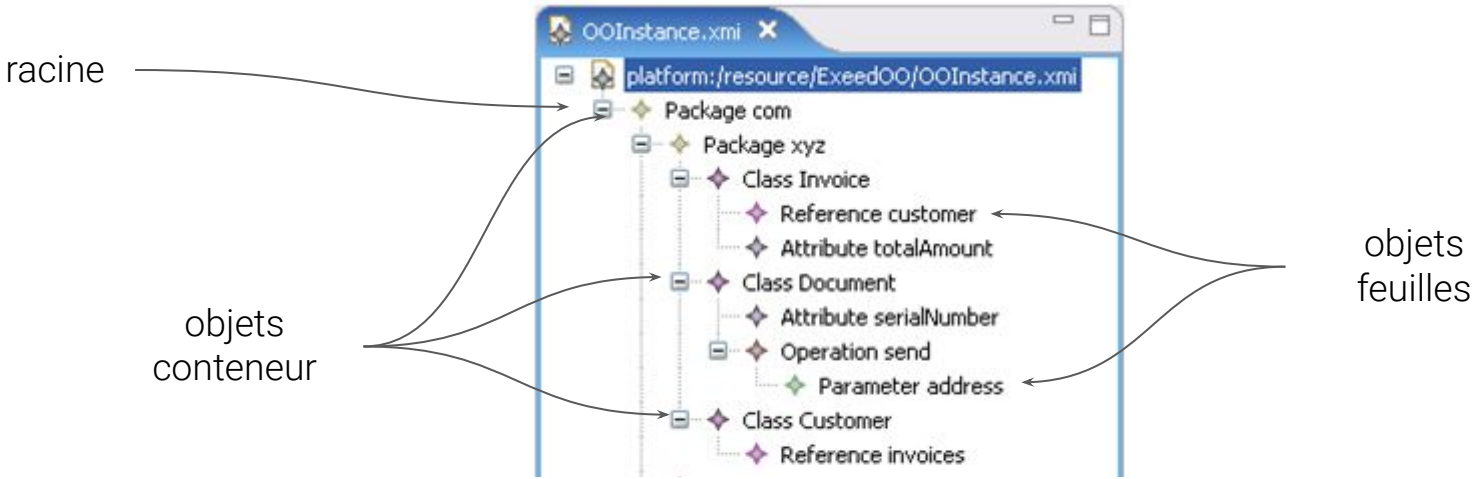
- Query/navigation dans un modèle
- Query/navigation dans un métamodèle
- OCL (intro + réutilisation de navigation)
- Création/modification/suppression (problématiques de suppressions)

Manipulation de modèles ?

- Pour récupérer de l’information dans un modèle
- Modifier un modèle
- Via des outils dédiés
- Programmatiquement

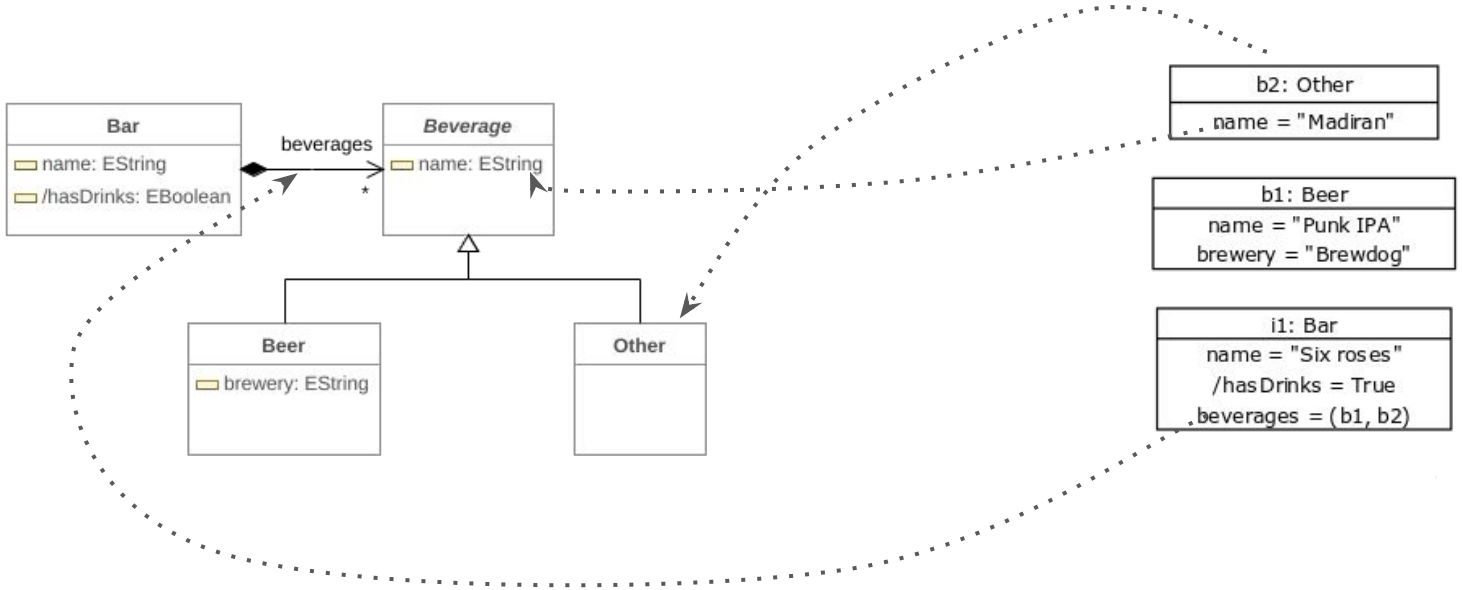
Techniquement, c’est quoi un modèle ?

Techniquement, un modèle est un ensemble d’objets qui sont tous contenus dans un autre, sauf la racine du modèle (qui est un objet qui peut contenir d’autres objets, mais qui n’a pas de parents).

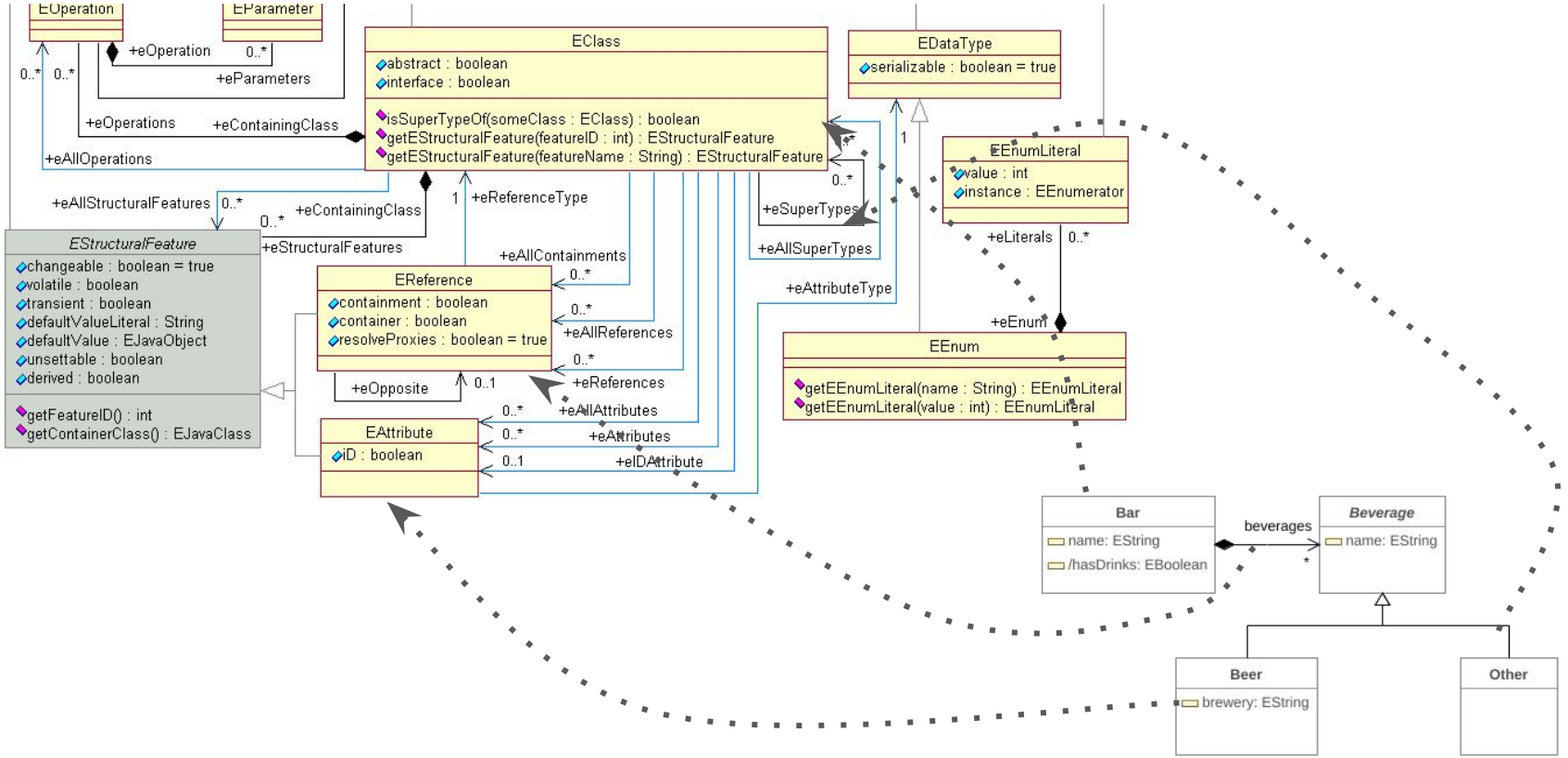


Relation d’un modèle avec son métamodèle

- Une instance possède des valeurs pour les attributs et relations définies par sa métaclasse

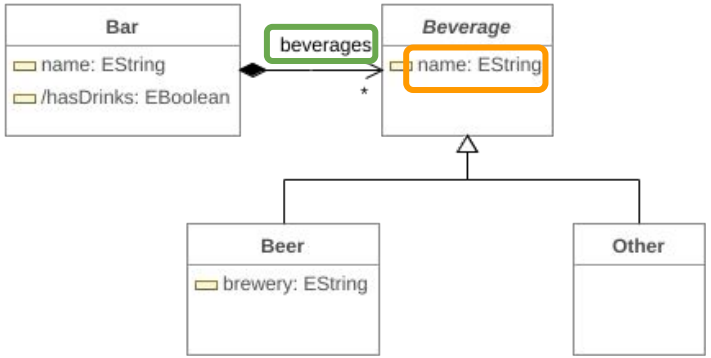


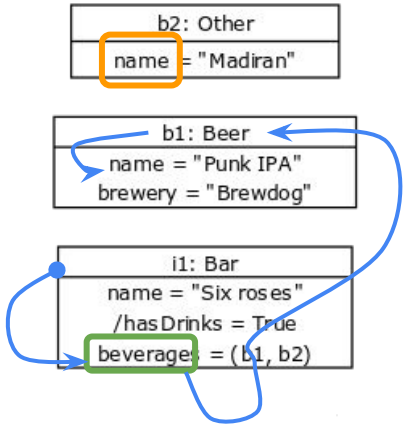
Relation d'un métamodèle avec son métamétamodèle



Navigation

- La navigation dans un modèle repose sur les attributs et références définis par son méta-modèle





OCL

```
b2.name  # returns “Madiran”
i1.beverages->at(1).name  # returns "Punk IPA"
```

Cardinalités dans le métamodèle - questions

- Quels sont les “types” de retours des attributs/relations en fonction des différentes cardinalités et pourquoi ?
- Quel va être l’impact de la composition ?

Frameworks pour la métamodélisation

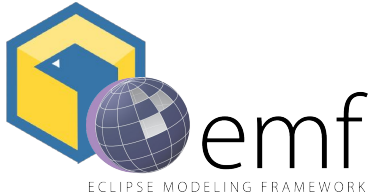
EMF - Java

- Implémentation de référence EMOF
- Usage établie dans l'industrie
- Usage dans le monde académique
- Peut être lourd à manipuler

PyEcore - Python

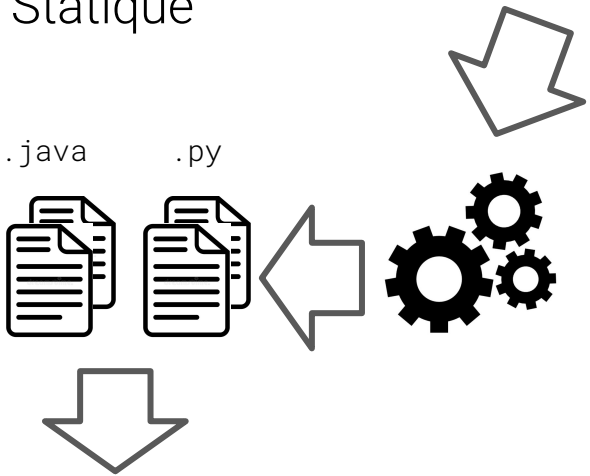
- Implémentation d'EMF en Python
- Usage grandissant dans l'industrie
- Usage dans le monde académique
- Manipulation réflexive plus légère

Deux usages particuliers



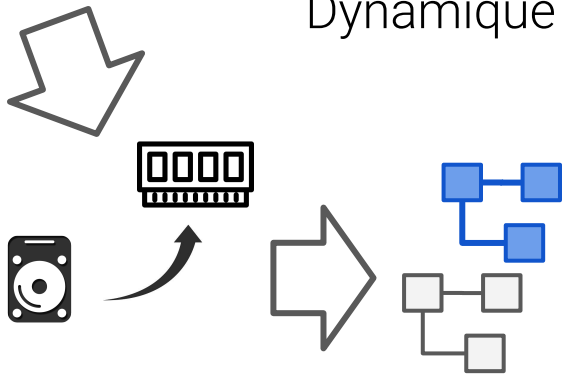
Métamodèle

Statique



- Utilisation mixte

Dynamique



- Utilisation réflexive (Java)
- Utilisation mixte (Python)

Navigation programmatique - avec EMF (Java)

```java
Beer b2 = ...; // Chargé de quelque part
b2.getName();

Bar i1 = ...; // Chargé de quelque part
i1.getBeverages().get(0).getName();
```

Navigation programmatique - avec PyEcore (Python)

```
b2 = ...  # Chargé ce quelque part
b2.name

i1 = ...  # Chargé de quelque part
i1.beverages[0].name
```

Navigation programmatique complexes - en Java

```
Collection<Beverage> bevs = i1.getBeverages()
                                .stream()
                                .reject(b -> b.getName() == null)
                                .filter(b -> b isinstanceof Beer)
                                .filter(b -> b.getName().contains("IPA"))
                                .collect(Collectors.toList());

for (Beverage bev: bevs) {
  Beer b = (Beer) bev;
  System.out.println(
    String.format("Name: %s, Brewery: %s", b.getName(), b.getBrewery())
  );
}
```

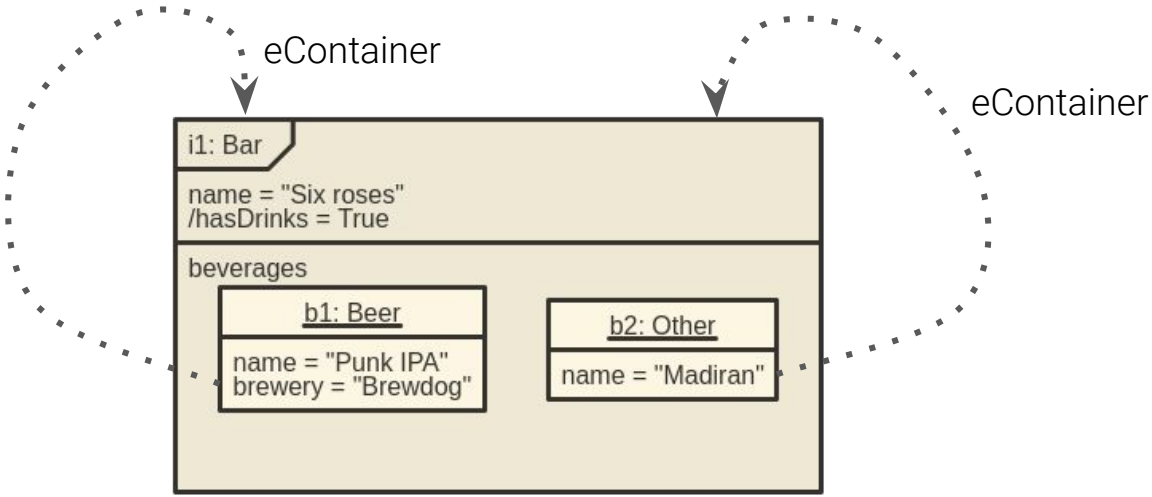
Navigation programmatique complexes - en Python

```
bevs = [b for b in i1.beverages if b.name is not None
                                   and isinstance(b, Beer)
                                   and "IPA" in b.name]

for b in bevs:
  print(f"Name: {b.name}, Brewery: {b.brewery}")
```

Navigation programmatique - remonté sur parent

- Chaque objet inclus dans une relation composite possède un parent
- On peut remonter sur son parent sans avoir de relation inverse



Navigation programmatique - remonté sur parent

```java
// En JAVA, besoin de caster

EObject obj = b2.eContainer();
((Bar) obj).getName();

// ou
((Bar) b2.eContainer()).getName();

// ou
Bar bar = (Bar) b2.eContainer();
bar.getName();
```

```python
# En Python, pas besoin de cast

b2.eContainer().name

# or

bar = b2.eContainer()
bar.name
```

Navigation réflexive

- La navigation réflexive d'un modèle repose sur le fait de ne pas passer directement par une relation dans le code

`i1.getBeverages().get(0)` ❌ `i1.beverages[0]` ❌

- On va essayer de récupérer des informations depuis le méta-modèle

✅ `i1.get(<relation from MM>)[0]`

- Permet d'écrire des algorithmes plus génériques

Navigation réflexive avec EMF

La récupération de la valeur d'un attribut/relation est plus "verbeuse" :

1. récupérer la méta-classe d'un objet
2. chercher la relation voulue
3. demander à l'objet la valeur de la relation

```
Bar b1 = ...;

EClass barEClass = b1.eClass();
EStructuralFeature nameFeature = barEClass.getEStructuralFeature("name");
String result = (String) b1.eGet(nameFeature);

EList<Beverage> bevs = (EList<>) b1.eGet(barEClass.getEStructuralFeature("beverages"));
```

Navigation réflexive avec PyEcore

La récupération de la valeur d'un attribut/relation est plus "verbeuse" :

1. récupérer la méta-classe d'un objet
2. chercher la relation voulue
3. demander à l'objet la valeur de la relation

```
b1 = ...

barEClass = b1.eClass
nameFeature = barEClass.findEStructuralFeature("name")
result = b1.eGet(nameFeature)

bevs = b1.eGet(barEClass.findEStructuralFeature("beverages"))

# or
bevs = barEClass.eGet("beverages")
```

Lister toutes les attributs/relations d’une méta-classe

Diverses méthodes existent pour récupérer la liste de toutes les “features” d’une métaclasse :

- eAllStructuralFeatures() : toutes les features en considérant l’héritage
- eAllReferences() : toutes les références en considérant l’héritage
- eAllAttributes() : tous les attributs en considérant l’héritage
- eStructuralFeatures : toutes les features sans considérer l’héritage
- eReferences : toutes les références sans considérer l’héritage
- eAttributes : tous les attributs sans considérer l’héritage

Langage dédié - OCL

- Historiquement uniquement utilisé pour poser des contraintes sur les modèles
- Contrainte => nécessité de naviguer dans le modèle
- Langage sans effet de bords
- Facilité de filtrage des collections
- Impossible de manipuler le modèle de façon réflexive

De votre expérience de l’année passée, quels sont les avantages et désavantages que vous voyez entre l’utilisation d’OCL et d’un langage généraliste ?

Création/Modification/Suppression

- Navigation = récupérer de l'information dans un modèle
- Genre de CRUD opérations = altérer la "tête" d'un modèle

Création

Modification

Suppression

Conservation de cohérence ?

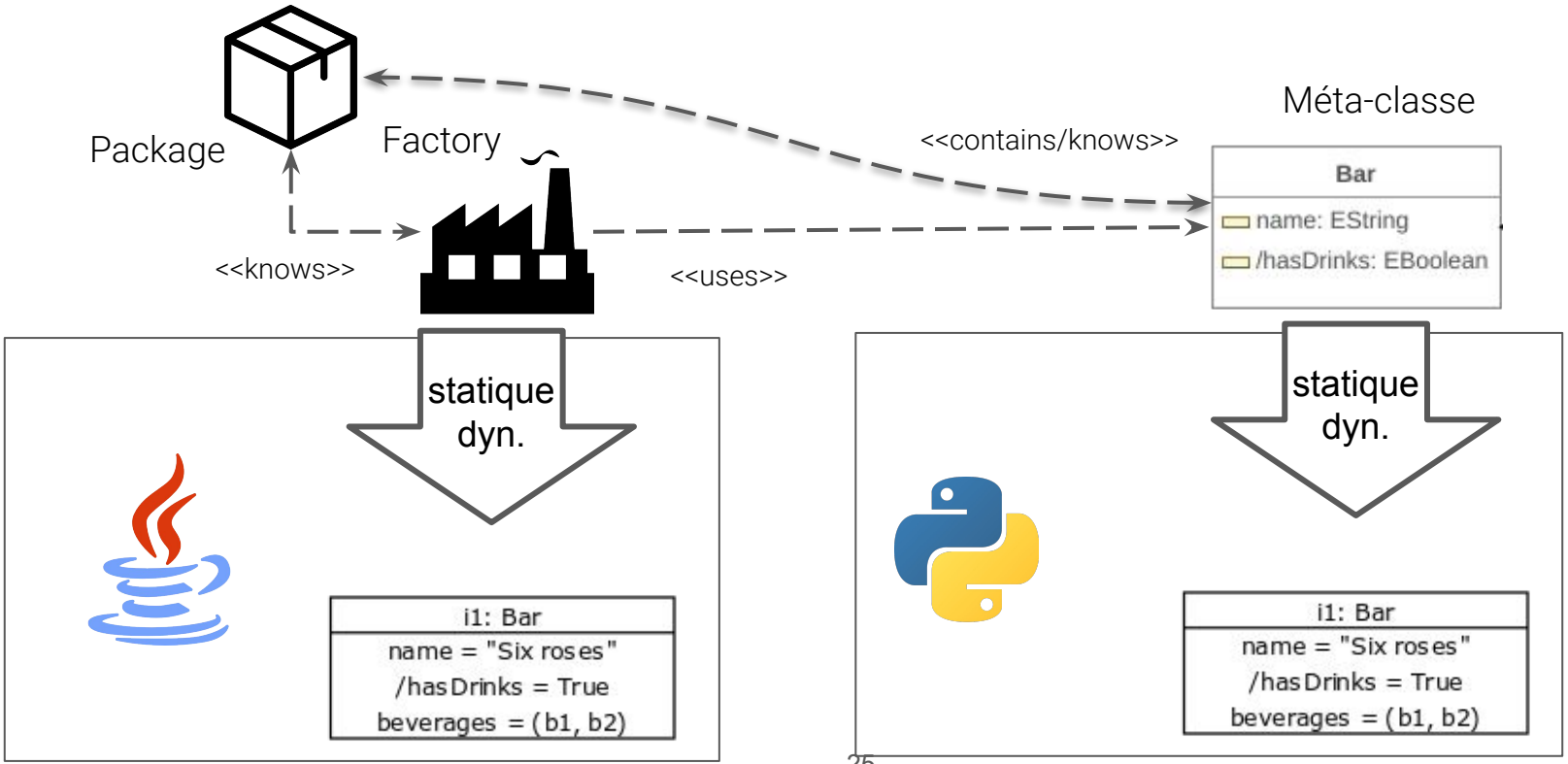
- Quelles sont les opérations sur le modèle qui nécessitent de mettre à jour les instances d’un modèle ?
- De quoi dépendent-elles ?

Conservation de cohérence ?

La modification d’un instance d’un modèle entraîne un ensemble de mécanisme interne pour conserver la cohérence des instances

- en fonction du type des éléments
- en fonction de la composition
- en fonction des relations opposites
- en fonction de la cardinalité

Créer des instances



Créer des instances

```java
// Through the metaclass
EClass metaclass = BeerMetaPackage.eInstance
                                  .getBeer();
Bar b1 = BeerMetaFactory.eInstance
                        .create(metaclass);

// Using directly the generated classes
Bar b2 = BeerMetaFactory.eInstance
                        .createBeer();
```

```python
# Gain access to a metaclass
# by navigation through the metamodel
# let's suppose we got the Bar metaclass
Metaclass = metaPackage.eClassifiers[0]

i1 = Metaclass()

# Using directly the generated metaclass
from beermeta import Bar
i2 = Bar()
```

Modifier des instances

- on passe par les setters
- on utilise l'api des collections pour les relations many
- utilisation différentes entre le mode statique et dynamique
- manipulation réflexive avec `eGet()` et `eSet()`

- on utilise la syntaxe d'affectation
- on utilise l'api des collections pour les relations many
- aucune différences entre le mode statique/dynamique
- manipulation réflexive avec `eGet()` et `eSet()`

Modification d’instances

```java
Beer b1 = ...; // A beer is created
b1.setName("Omniprairie");

EStructuralFeature nameF = b1.eClass().getEStructuralFeature("name");
b1.eSet(nameF, "Omniprairie");
```

```python
b1 = ... # Created somewhere
b1.name = "Omniprairie"

nameF = b1.eClass.findEStructuralFeature('name')
b1.eSet(nameF, "Omniprairie")  # or
b1.eSet("name", "Omniprairie")
```

Suppression d’instances

- problématiques de suppression (retirer du graphe, conservation de cohérence)
- La suppression propre d’instances passe par des méthodes spéciales
- La façon la plus simple de faire est de retirer l’objet de son conteneur et de “sérialiser” le modèle
- En Java : opération longue
- En Python : opération plus rapide

Lier plusieurs modèles entre eux ?

Il est possible de lier plusieurs modèles, même de différents méta-modèles, entre eux (si permis par les métamodèles)

- Techniquement c’est quoi un modèle ?
- Permet un découpage de gros modèles en plusieurs petits modèles
- Pour manipuler proprement un modèle coupé en plusieurs morceaux, il faut que tous les morceaux soient chargés en mémoire.

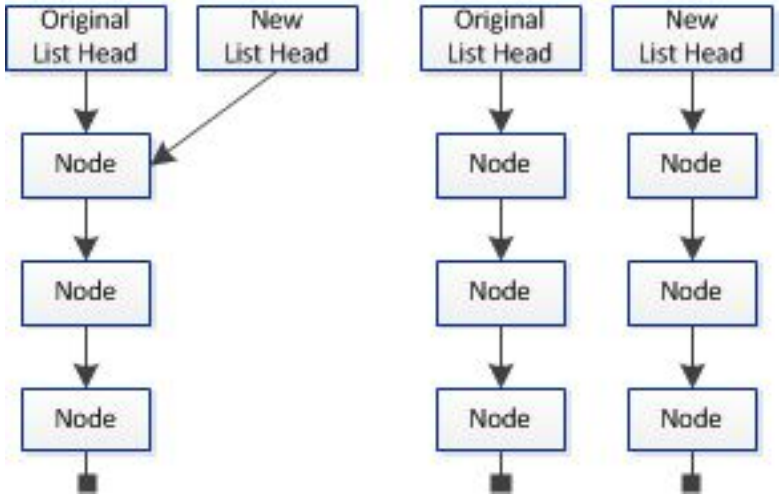
Problématiques connues dans les modèles

Copie/clone de modèles

Les opérations de copies/clones d’objets sont une problématique connues des systèmes à objets :

- Shallow vs Deep
- Problème des cycles pour le deep clone/copy

Shallow copy



Deep copy

Comparaison/égalité de modèles

Comparer des modèles et des objets ou s’assurer de leur égalité sont des tâches complexes :

- que signifie l’égalité de deux objets ? Shallow vs Deep
- Comparer == donner un ordre, en dehors de l’égalité, généralement pas possible

Diff de modèles

Effectuer un diff de modèle suppose qu'on est capable de trouver exactement quelles sont les modifications qui ont été effectuée sur un modèle

- On doit considérer un méta-modèle constant pour deux modèles
- Tous les opérateurs d'égalités et de "diff" doivent être redéfinis, spécialement pour les collections
  - Il est difficile de savoir entre deux versions si un élément est le même (si on a pas d'identifiants unique), ça veut dire qu'il est difficile de savoir s'il y a eu un "suppression + ajout" potentiellement d'un élément ou un "renommage + modification"

Ce qu’il faut retenir

- Un métamodèle donne des règles sur la manière dont ses modèles peuvent être construits
- On peut naviguer un modèle et un métamodèle de la même manière
- On peut naviguer dans un modèle en utilisant les relations décrites dans le métamodèle
- On peut naviguer dans un modèle de façon réflexive en utilisant directement les informations tirée du métamodèle
- Chaque opération de modification entraîne des opérations internes de conservation de la cohérence du modèle

Ce que vous devez savoir faire

- Utiliser Java ou Python pour naviguer dans un modèle
- Pouvoir créer un modèle entièrement programmatiquement
- Pouvoir créer un méta-modèle entièrement programmatiquement
- Utiliser les informations du méta-modèle pour naviguer dans un modèle de façon réflexive
- Comprendre les enjeux impliqués par les objets dans les opérations plus complexes de copie/égalité/diff de modèles