# Modélisation et mise en oeuvre d'un langage de modélisation des données

Alain Plantec

14 octobre 2020

Un langage de modélisation de données permet la spécification d'entités représentant la structure des objets d'un domaine et la spécification des relations entre ces entités. Dans cet exercice, nous nous intéressons à un exemple d'un tel langage appelé *minispec*.

Dans minispec, une entité comprend un ensemble d'attributs typés. Les propriétés des entités sont modélisées par des attributs de type simple (String, Integer, Real, Boolean) ou par des collections (List ou Array) dont les éléments sont de type simple. Les relations entre entités sont modélisées par des attributs dont le type est une autre entité ou par des collections dont les éléments sont de type entité.

## 1 Plan de travail

Cet exercice comprend une partie méta-modélisation et une partie mise en oeuvre. Ces deux parties sont liées : la mise en oeuvre et déduite du méta-modèle mais ce dernier peut être remis en cause par la mise en oeuvre.

**Il vous est donc demandé :**

- **de produire un méta-modèle en UML pour le langage de modélisation de données *minispec* ;**
- **de traduire votre méta-modèle UML en classes Java ;**
- **de construire un pretty-printer permettant de produire une représentation textuelle lisible dans sa syntaxe concrète (la syntaxe de la figure 3) à partir des instances du métamodèle en mémoire (optionnel) ; vous testerez le pretty-printer à l'aide de tests JUnit (optionnel) ;**

— **de définir une syntaxe XML pour représenter les modèles de données ;**
— **de produire une DTD pour votre syntaxe XML (optionnel)**
— **de programmer les composants de matérialisation et de sérialisation depuis et vers XML (la sérialisation est optionnelle) ;**
— **de construire un outil de renommage simple d'élément de modèle en utilisant un visiteur.**
— **de construire un outil de renommage plus évolué (optionnel).**

**Vous accompagnerez le code livré pour les différentes parties avec un fichier *lisezMoi.txt* dans lequel vous indiquez ce qui a été fait et toutes autres informations utiles à l'évaluation.**

Il s'agit donc d'un exercice d'application de ce qui a été vu en cours autour de l'exemple sur les expressions arithmétiques. Les développements montrés en cours concernant les expressions arithmétiques sont disponibles sur le moodle. [1]

Les développements seront fait indépendamment de la syntaxe concrète. Il n'est notamment pas demandé de construire un analyseur de modèle spécifié en utilisant sa syntaxe concrète lisible (celle de l'exemple montré dans la figure 3).

## 2 Une version minimale pour démarrer

Vous allez développer un premier méta-modèle pour une version minimale du langage *minispec*. La figure 3 montre un exemple simple de specification avec *minispec* pour cette première étape. La première version de *minispec* intègre donc uniquement la déclaration de modèles caractérisés par un nom et comprenant des entités avec des attributs de type primitif (String, Integer...). Le modèle UML d'un exemple de méta-modèle pour *minispec* est montré dans la figure 1. La figure 2 montre le squelette de classes Java permettant de metre en oeuvre le méta-modèle UML 1. Attention, ce méta-modèle ne permet de gérer que des attributs de type simple.

Pour représenter un modèle nous utiliserons une syntaxe XML. Un outils développé pour *minispec* lit les modèles de données via des fichiers XML. Les éléments XML sont ainsi matérialisés en instances du méta-modèle. Par exemple, le code XML de la figure 4 peut être utilisé comme l'équivalent

1. voir https://moodlesciences.univ-brest.fr/moodle/course/view.php?id=875&notifyeditingon=1


FIGURE 1 – Un métamodèle de minispec présenté en UML

```
class Model {
   String nom ;
   List<Entity> entities;
}
class Entity {
   String nom ;
   List<Attribute> attributes;
}
class Attribute {
   String nom ;
   String type;
}
```

FIGURE 2 – Un métamodèle possible de minispec présenté en Java

de notre exemple de modèle avec l'entité *Satellite*. La matérialisation de ce document XML produit une instance de la classe Java *Entity* associée à deux instances de la classe *Attribute*..

Pour la mise en oeuvre du composant de matérialisation, on s'appuie sur un document DOM : l'analyseur XML produit un document DOM. Pour l'étape de matérialisation, une fabrique exploite le document DOM pour produire le graphe d'objets de la représentation en mémoire sous la forme

```
model Flotte;
   entity Satellite ;
      nom : String ;
      id :Integer ;
   end_entity ;
end_model;
```

FIGURE 3 – Un exemple de specification avec *minispec*

```xml
<entity name="Satellite">
  <attribute name="nom" type="String"/>
  <attribute name="id" type="Integer"/>
</entity>
```

FIGURE 4 – Une specification en XML possible de notre exemple de modélisation de l'entité *Satellite*

```
model Flotte;
  entity Satellite ;
    nom : String ;
    id :Integer ;
    parent : Flotte;
  end_entity ;
  entity Flotte;
  end_entity;
end_model;
```

FIGURE 5 – Un exemple de specification avec une association simple entre deux entités

d'instances du méta-modèle écrit en Java. Cette représentation en mémoire pourra ensuite être parcourue et exploitée à l'aide d'un visiteur pour mettre en oeuvre l'outil de renommage.

# 3 Association simple entre entités

Avec *minispec*, une association unidirectionnelle et unaire d'une entité A avec une entité B est modélisée à l'aide d'un attribut de l'éntité A dont le type est B. Cette association spécifie que toute instance de l'entité A est associé avec une instance de l'entité B. La figure 5 montre une association simple entre l'entité *Satellite* et l'éntité *Flotte*. Cette association est spécifiée par l'attribut *parent*. Le nom *Flotte* est utilisé comme type de l'attribut parent. La figure 6 montre une solution pour représenter cette association en XML. La première version du méta-modèle donné dans la figure 2 peut convenir ici : on remarque que le nom *Flotte* est directement utilisé comme valeur de

```
<entity name="Satellite">
  <attribute name="nom" type="String"/>
  <attribute name="id" type="Integer"/>
  <attribute name="parent" type="Flotte"/>
</entity>
<entity name="Flotte">
</entity>
```

FIGURE 6 – Specification d'une association simple en XML

```
entity Flotte;
  satellites  : List of Satellite;
end_entity;
```

FIGURE 7 – Une association multiple entre deux entités

l'attribut *type* pour indiquer le type de l'attribut *parent*.

# 4 Association multiple

Une association multiple est utilisée pour indiquer une relation d'une instance d'entités avec potentiellement plusieurs autres instances d'entités. Dans la figure 7, l'entité *Flotte* est complétée avec l'attribut *satellites* qui spécifie une collection d'instances de *Satellite*. Il peut être interessant d'indiquer des cardinalités min et max. Par exemple, dans la figure 8, les cardinalités indiquent explicitement qu'une flotte comporte au minimum 1 satellite et au maximum 10. Ces informations peuvent être utilisées pour implanter des controles dans les méthodes permettant de modifier les collections ou générer des tests.

Le type de collection utilisé peut etre *List*, *Set*, *Bag* ou *Array*. Attention donc aux points suivants :

— Le métamodèle doit être adapté pour supporter les collections à plusieurs dimensions et les collections de natures différentes (*List*, *Array*, *Set*, *Bag*) ;

— Un tableau (Array) est de taille fixe, les autres collections (List, Set,

```
entity Flotte;
  satellites  : List [1 :10] of Satellite;
end_entity;
entity Satellite;
  panneaux  : Array [2] of PanneauSolaire;
end_entity;
```

FIGURE 8 – Exemples d'associations multiples avec des cardinalités (List) ou une taille fixe (Array)

Bag) peuvent avoir une cardinalité minimale et maximale ;

La première version du méta-modèle montrée dans la figure 2 n'est plus suffisante. Il vous faut l'adapter pour pouvoir stocker les informations concernant les associations multiples avec les cardinalités ou la taille.

De même, il vous faut déterminer une syntaxe XML adéquate pour spécifier une association multiple.

# 5 Outil de renommage

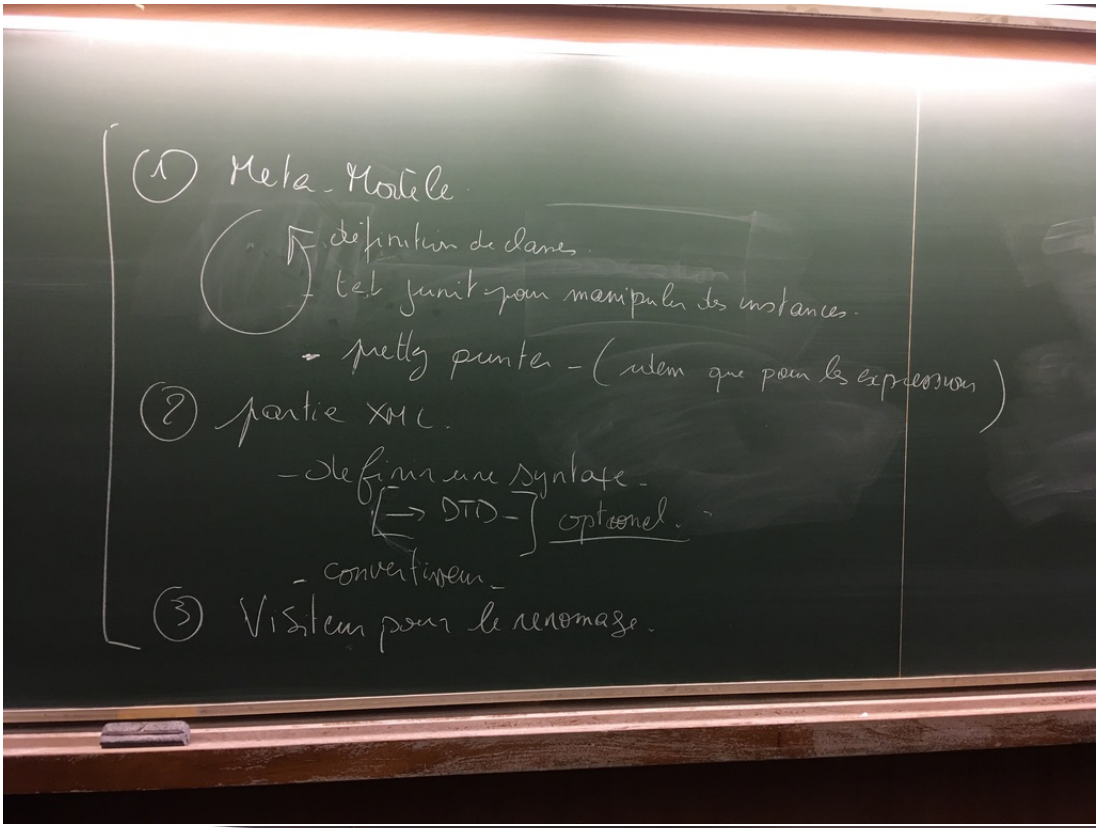
L'outil à mettre en oeuvre permet de renommer n'importe quel élément ayant un nom : modèle, entité ou attribut. Dans une version basique, l'outil prend en entrée deux identifiants. Le premier identifiant correspond au nom de l'élément ou des éléments à renommer. Le second identifiant correspond au nouveau nom du ou des éléments à renommer. Il est demandé de produire des tests JUnit pour montrer le bon fonctionnement de votre outil. Au delà de cette version basique, vous pouvez produire un outil de renommage plus complet ou plus évolué

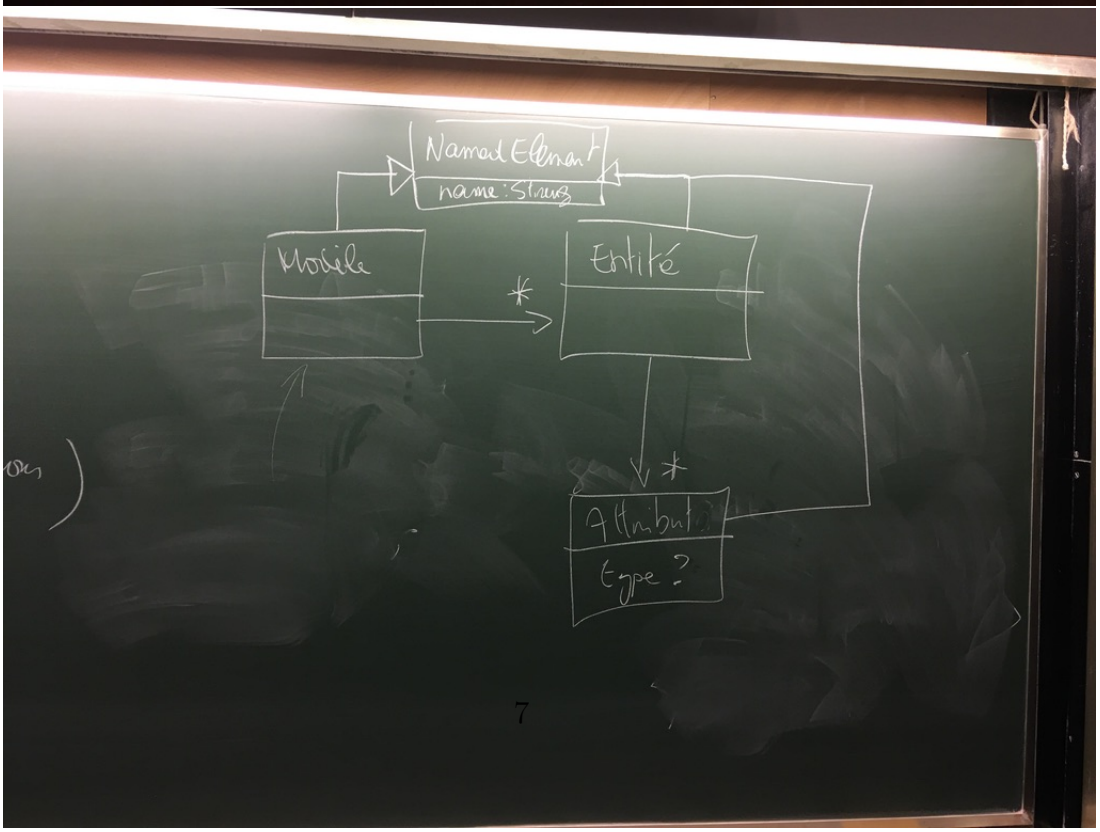
# 6 Annexes

Pour mémoire, ce que je vous avais présenté au tableau avec :

— le plan de travail

— une ébauche d'un méta-modèle en java.


① Meta-Modèle
définition de classes
tel junit pour manipuler des instances.
- pretty printer - (utilisé que pour les expressions)
② partie XML.
- définir une syntaxe.
[→ DTD -] optionnel.
- conversion
③ Visiteurs pour le renommage.


NamedElement
name : String
Modèle
Entité
*
*
Attribut
Type ?