TD – Ingénierie des Modèles

Transformation de modèles

Master des Technologies de l'Internet $2^{ème}$ Année

Le but de ce TD est de définir une transformation de modèle simple à la fois via ATL et Kermeta pour étudier la différence d'expression entre un langage déclaratif et un langage impératif.

1 Description des modèles

On traite dans ce TD un modèle très simplifié d'architecture logicielle. Une appplication est formée de clients et de serveurs qui sont liés via des interfaces de services. La tranformation consiste à placer un élément proxy entre chaque client et chaque serveur.

1.1 Description informelle

Dans le modèle initial, on trouve trois types d'éléments :
- Interfaces de service (on n'en détaillera pas le contenu)
- Serveur : implémente une ou plusieurs interfaces et est associé à plusieurs types de clients
- Client : chaque type de client est associé à un seul serveur et utilise une de ses interfaces

Après transformation, il s'agit de trouver un proxy associé à chaque type de client entre ce type de client et le serveur. Le proxy implémente alors l'interface utilisé par ce type de client à la place du serveur. Les clients n'ont plus de liaison directe avec le serveur mais uniquement avec les proxys. Ces derniers sont associés au serveur.

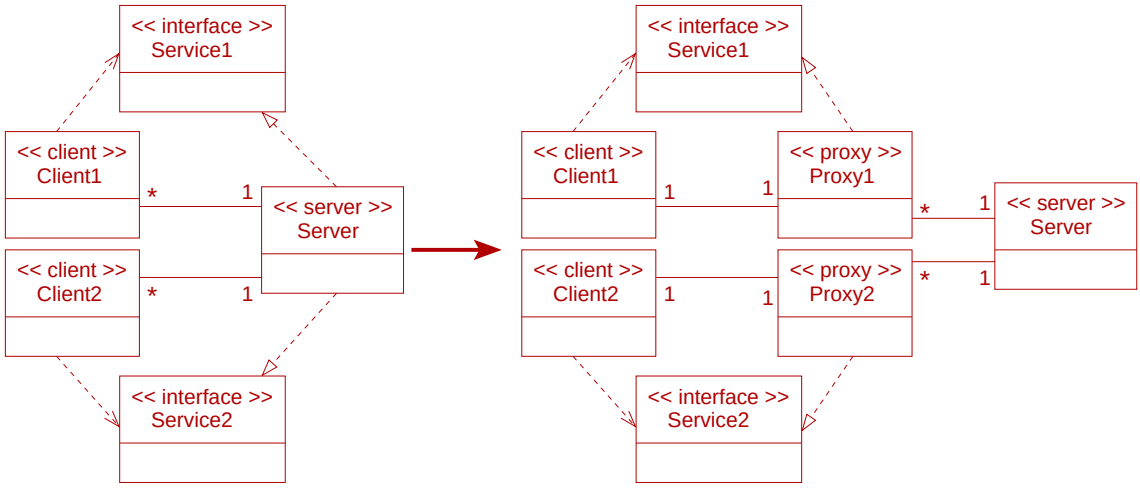

Figure 1: Exemple de transformation d'application

La figure 1 montre un exemple d'une telle application, avant et après transformation, pour 2 types de clients[1].

[1]La notation utilisée ici est celle d'un diagramme de classe UML utilisant des stéréotypes pour chaque type d'élément.

1.2 Définition des méta-modèles

Les 2 méta-modèles source et cible sont définis ci-dessous à la fois en Kermeta et sous la forme d'un diagramme de classe[2].

1.2.1 Méta-modèle source

```
class ModelElement
{
  attribute nom : String
}

class Interface inherits ModelElement
{
  reference estImplementeePar : Serveur
  reference estUtiliseePar : Client
}

class Client inherits ModelElement
{
  reference utilise : Interface
  reference connecteA : Serveur
}

class Serveur inherits ModelElement
{
  reference implemente : Interface[1..*]
  reference connecteA : Client[1..*]
}
```

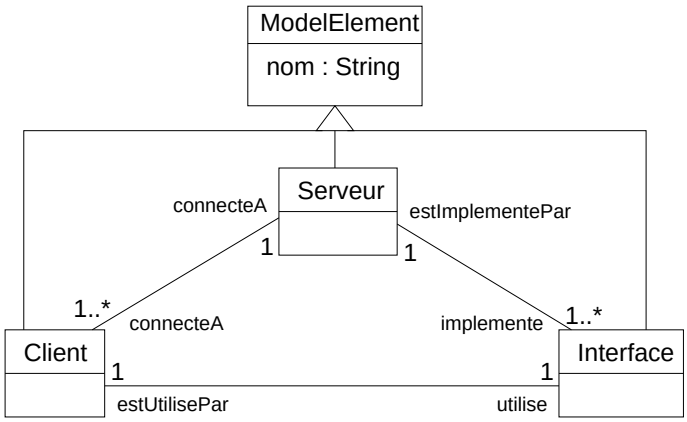

Figure 2: Méta-modèle source

[2]Pour simplifier les méta-modèles, les associations (ainsi que les cardinalités) ne sont pas explicitiment définies, on se contentera de considérer les références entre méta-classes comme des associations simplifiées. On en fera de même pour la relation d'implémentation ou de de dépendance entre une classe et une interface.

1.2.2 Modèle cible

```
class ModelElement
{
  attribute nom : String
}

class Interface inherits ModelElement
{
  reference estImplementeePar : Proxy
  reference estUtiliseePar : Client
}

class Proxy inherits ModelElement
{
  reference clientConnecte : Client
  reference serveurConnecte : Serveur
  reference implemente : Interface
}

class Client inherits ModelElement
{
  reference utilise : Interface
  reference connecteA : Proxy
}

class Serveur inherits ModelElement
{
  reference connecteA : Proxy[1..*]
}
```

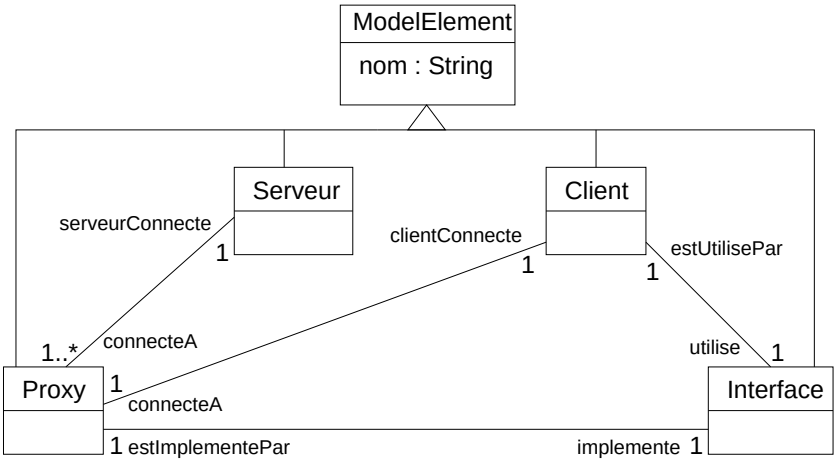

Figure 3: Méta-modèle cible

1.3 Exercice

1. Définir les contraintes OCL manquantes pour exprimer complétement les 2 méta-modèles afin de respecter la description informelle
2. Définir la transformation en mode purement impératif en Kermeta
3. Définir la transformation en mode purement déclaratif en ATL
4. Commenter les différences entre les 2 modes de transformation