Chapitre 6 Le standard QVT *(*Query,views,Transformations):

1. Introduction

L'ingénierie dirigée par les modèles, une approche de conception qui vise à garder les modèles comme point de référence tout au long du processus de développement logiciel, fait l'objet depuis plusieurs années d'une recherche active. Des organismes tels que l'OMG (Object Management Groupe) cherchent à apporter de nouvelles fonctionnalités aux différents modèles qui sont utilisés pour concevoir des applications, à faciliter la création de nouveaux espaces de modélisation plus adaptés aux besoins de leurs utilisateurs, et à faciliter les différentes étapes de modélisation nécessaires à l'élaboration d'un produit fini.

Un des points clés dans cette approche est la possibilité de transformer les modèles, d'un espace de modélisation ou d'un niveau d'abstraction vers un autre. En effet, la gestion manuelle des modèles s'avère aujourd'hui coûteuse et les modèles sont en conséquence bien souvent laissés de côté. Il devient donc nécessaire de proposer des outils flexibles de gestion automatique des modèles, permettant, tout au long de leur cycle de vie, de les porter aisément sous des formes dans lesquelles ils répondront mieux aux attentes de leurs utilisateurs à un moment donné.

2. Transformation de modèles dans le contexte de MDA :

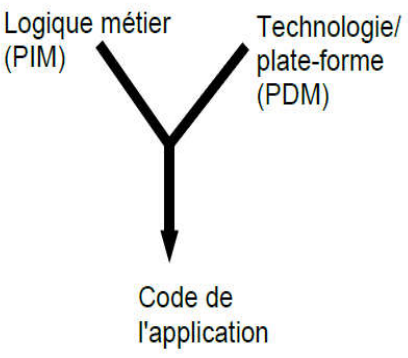
La transformation de modèles est une opération essentielle pour atteindre la pérennité de la spécification de systèmes. Selon MDA,
la première étape du développement de système consiste à définir la spécification de système de manière indépendante des plates-formes ou des intergiciels.

Cycle de développement d'un logiciel selon le MDA

Cycle en Y

Plus complexe en pratique

Plutôt un cycle en épi



Ensuite, ***Techniques de transformations***

3 grandes catégories de techniques de transformation

a. **Approche déclarativ**e

Recherche de certains patrons (d'éléments et de leurs relations) dans le modèle source

Chaque patron trouvé est remplacé dans le modèle cible par une nouvelle structure d'élément

Ecriture de la transformation « assez » simple mais ne permet pas toujours d'exprimer toutes les transformations facilement

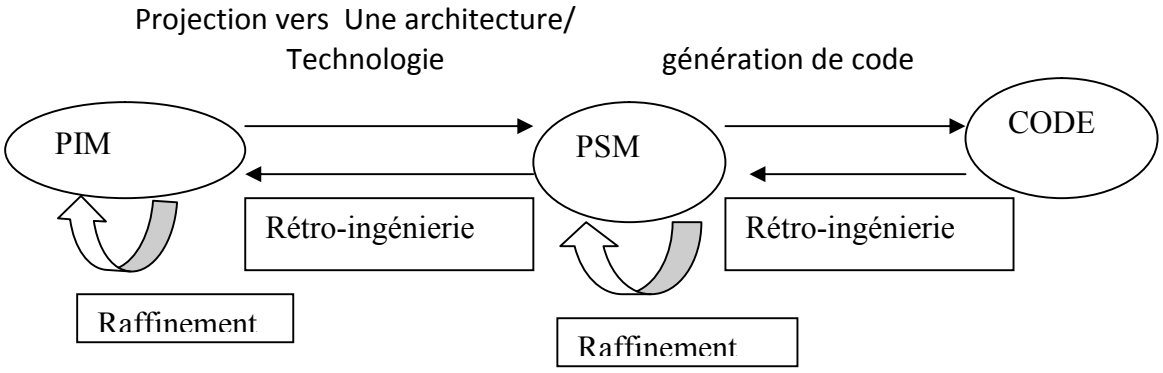
b. Approche impérative

Proche des langages de programmation

On parcourt le modèle source dans un certain ordre et on génère le modèle cible lors de ce parcours Ecriture transformation plus complexe mais permet de toutes les définir

c. **Approche hybride** : à la fois déclarative et impérative

Celle qui est utilisée en pratique dans la plupart des outils



Il existe 2 types de transformation:

Transformation endogène

- Dans le même espace technologique
- Les modèles source et cible sont conformes au même méta-modèle
- Transformation d'un modèle UML en un autre modèle UML

Transformation exogène

- Entre 2 espaces technologiques différents

- Les modèles source et cible sont conformes à des méta-modèles différents
- Transformation d'un modèle UML en programme Java
- Transformation d'un fichier XML en schéma de BDD

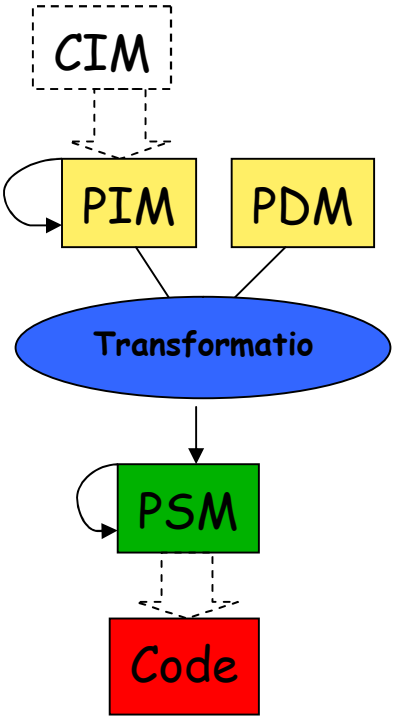
3. Objectif :

Selon MDA la réalisation d'une application Processus basé sur une série de transformations de modèles Partir de CIM (Computation Indépendant Model) : aucune considération informatique n'apparaît Faire des modèles indépendants des plateformes (PIM)

- rattaché à un paradigme informatique ; indépendant d'une plateforme de réalisation précise

Spécifier des règles de passage (transformation) vers les modèles dépendants des plateformes (PSM)

- version modélisée du code ;



Le modèle au centre du développement

- La traduction entre le PIM et les PSM est effectuée à l'aide d'outils automatisés, par exemple des transformations de modèles réalisées avec des outils plus ou moins compatibles avec le standard de l'OMG nommé QVT.

- Le passage du PSM à la génération du code est la suite logique de ce traitement. Elle peut être réalisée par des générateurs afin de produire tout type de cibles technologiques.

4. Définition de référentiel :

Référentiel (Repository) pour stocker modèles et méta-modèles

- Référentiel pour stocker modèles et méta-modèles
- Les (méta) méta modèles sont stockés selon le formalisme de l'outil
- XML par exemple pour les modèles
- Et DTD/Schema XML pour les méta-modèles
- Forme de stockage d'un modèle
- Modèle est formé d'instances de méta-éléments, via la syntaxe abstraite
- Stocke ces éléments et leurs relations
- L'affichage du modèle via une notation graphique est faite par l'AGL (atelier de génie logiciel)
- Les référentiels peuvent être notamment basés sur
- XML
- XMI : norme de l'OMG pour représentation modèle et méta-modèle
- Base de données relationnelle
- Codage direct dans un langage de programmation

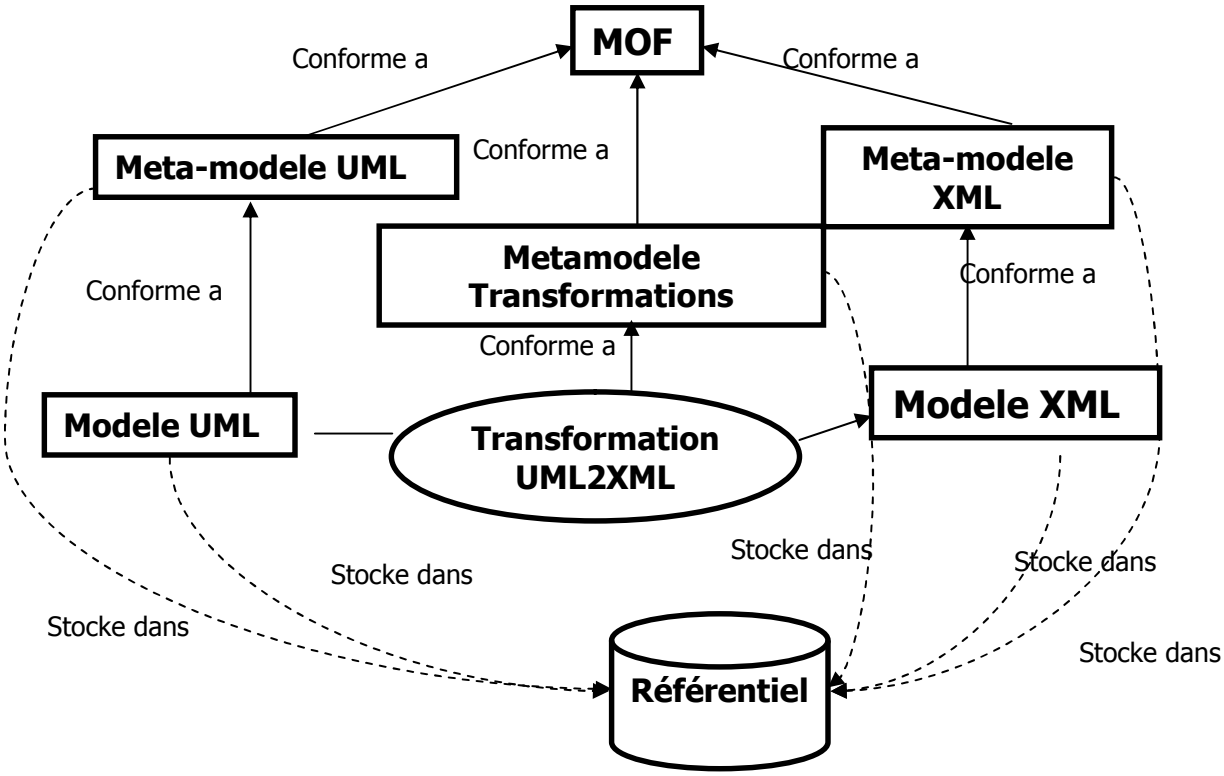
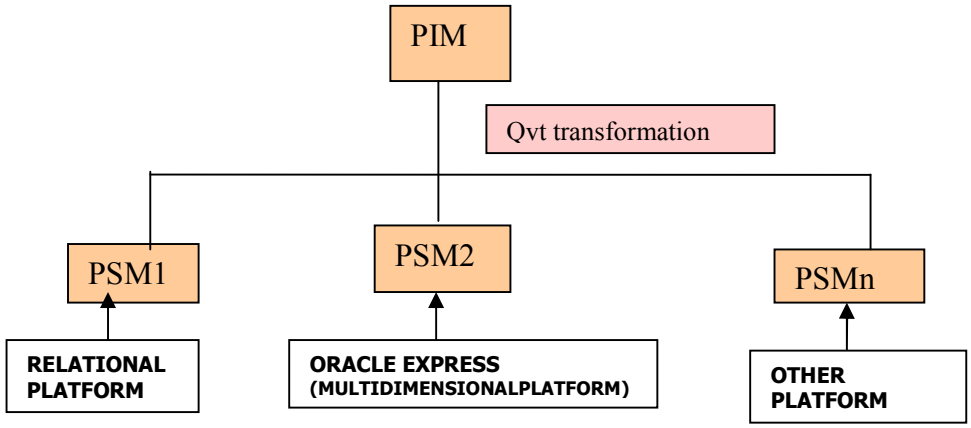


Figure 1 :Schéma de référentiel

5. Le standard QVT (Query,views,Transformations):

QVT(Query/Views/Transformations) un standard défini par l'OMG. Il s'agit d'un langage standardisé pour exprimer ces transformations de modèles. La notion de transformation de modèles est essentielle en MDA (Model Driven Architecture)



QVT transformation

5.1. Query

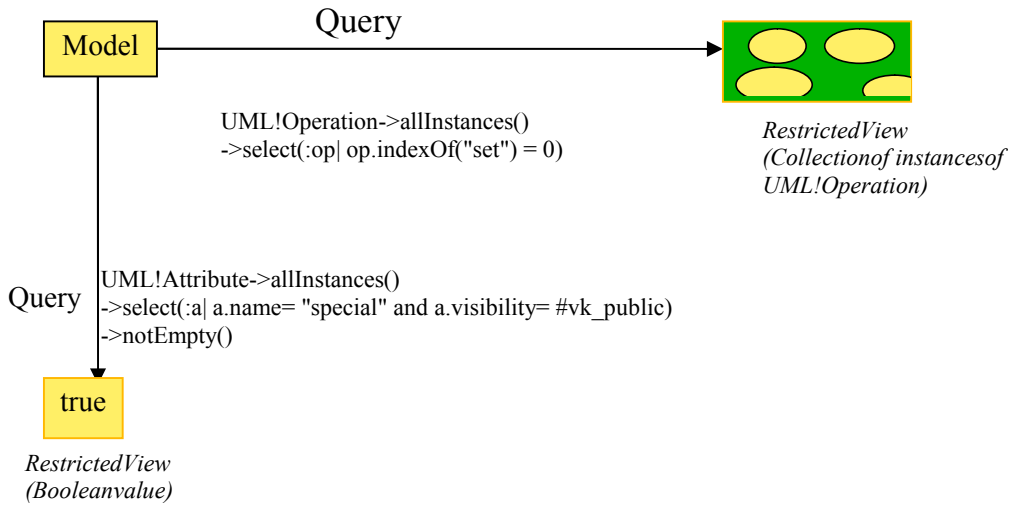
5.1.1. Définition de query

Sélectionner des éléments sur un modèle. Le langage utilisé pour cela est OCL.

OCL : Object Constraint Language, langage développé à l'origine pour exprimer des contraintes sur des éléments de diagrammes UML, il offre un ensemble de fonctions efficaces pour naviguer entre les éléments d'un modèle UML ou MOF et récupérer les informations liées à ces éléments .Un exemple de requête sur UML : retourne tous les packages qui ne contiennent pas des packages fils.

Le résultat est une collection des instances de package metaclasses. Un autre exemple est le suivant : Est-ce que une attribut particulière dans la source a une vision publique. Le résultat devient une variable booléenne.

5.1.2. Représentation de Query (requête)



5.2. View

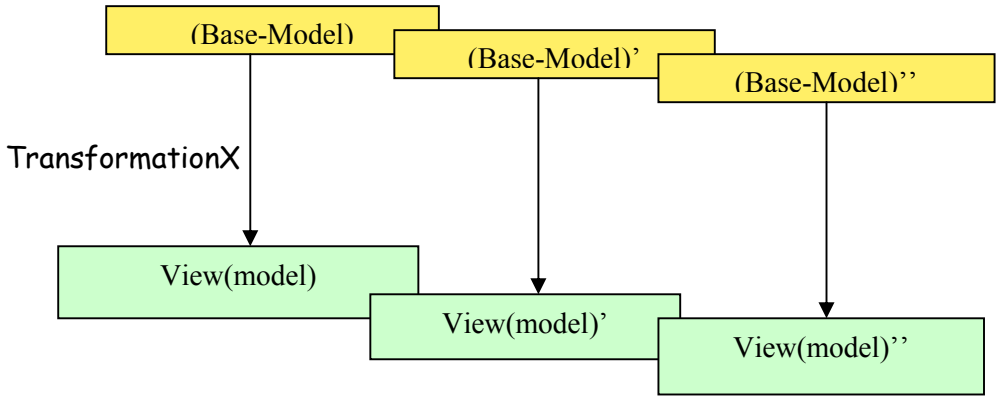
5.2.1. Définition de vue

Une vue est une sous partie d'un modèle Peut être définie via une Query(requête)

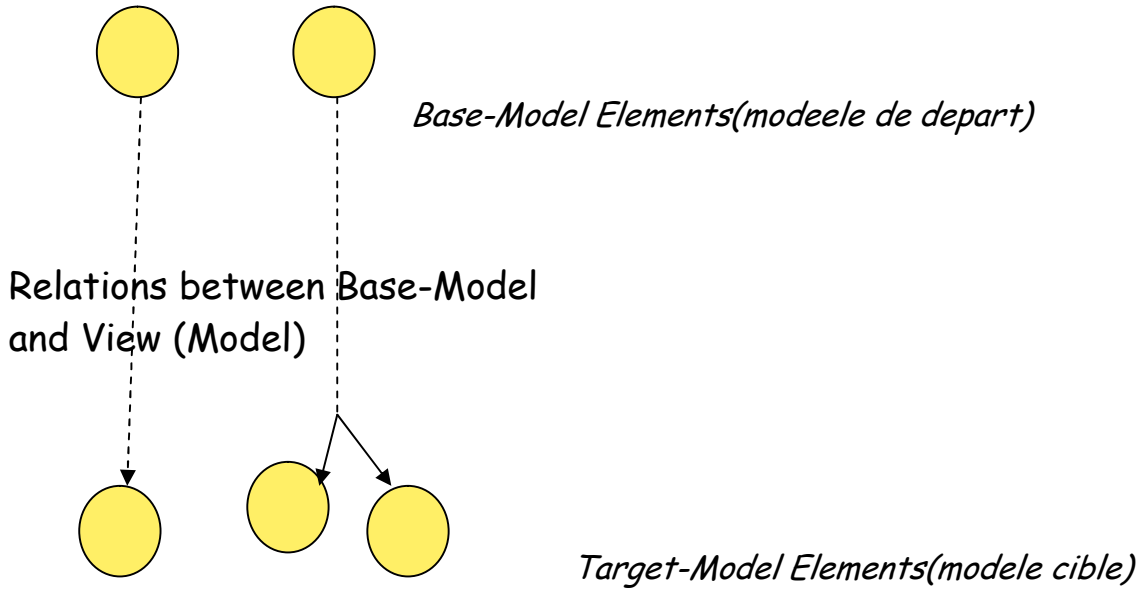
Une vue est un modèle à part, avec éventuellement un métamodèle restreint spécifique à cette vue. Elle ne peut être changée séparément du modèle de base, un changement sur le modèle de base provoque un changement sur la vue.

Une requête est type restreint de vue. Les vues sont générées à travers les transformations.

5.2.2. Représentation de view :



A : modification dans un modèle source provoque une modification dans le modèle cible



B : la relation entre les modèles source et les modèles cible sous forme de vues

Schémas A, B montrent une représentation de vue

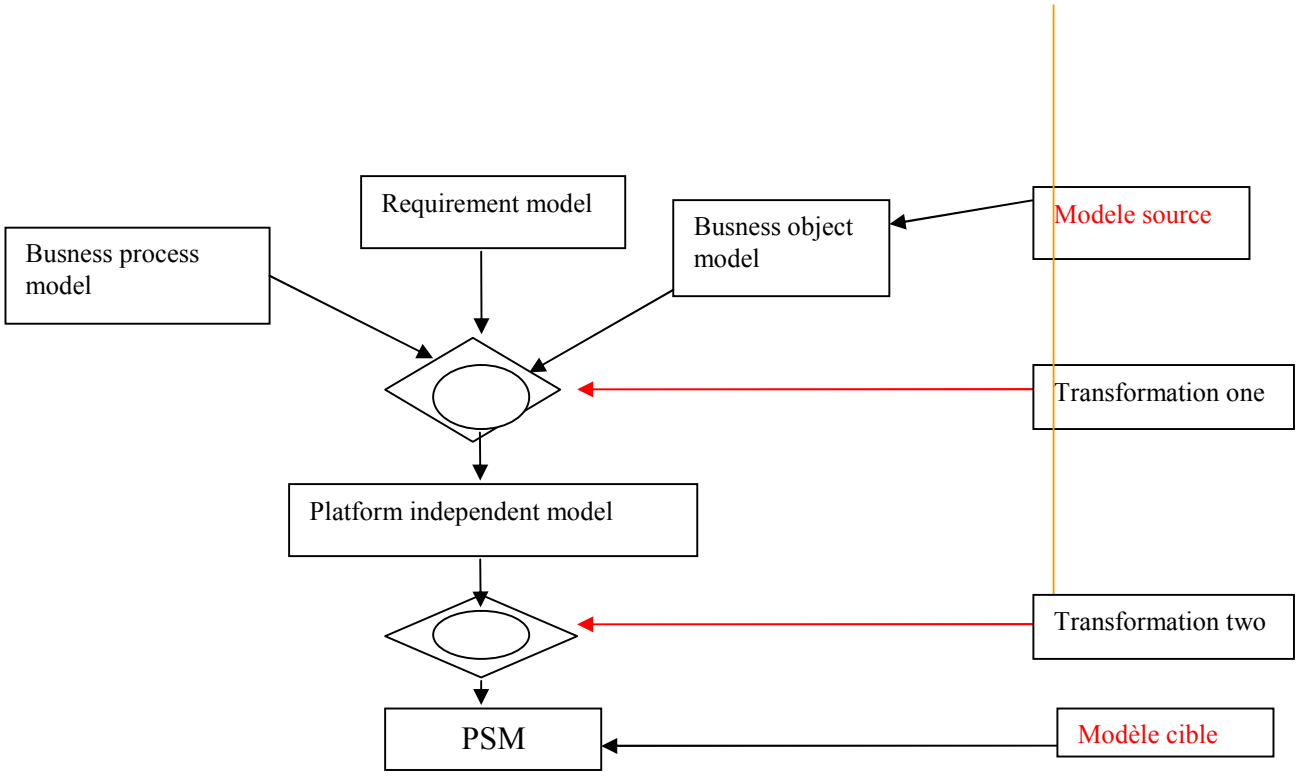
5.3 Transformation

5.3.1 Définition

C'est la transformation d'un modèle en un autre. La transformation génère un modèle cible a partir d'un modèle source.

5.3.2 Exemple de transformation

Le schéma suivant montre un exemple sur la transformation dans les différents niveaux de modèle en QVT



6. L'appel à propositions QVT :

Les standards MOF et XMI étant largement utilisés dans le cadre de la métamodélisation, ils ne suffisent toutefois pas à définir le procédé de transformation qui permet de passer d'un modèle à un autre sur la base des méta-modèles de départ et d'arrivée. En effet, des APIs ont été définies pour passer d'un formalisme à l'autre, mais l'enjeu des outils de transformation est de pouvoir transformer plusieurs classes de modèles, définies suivant plusieurs méta-modèles différents.

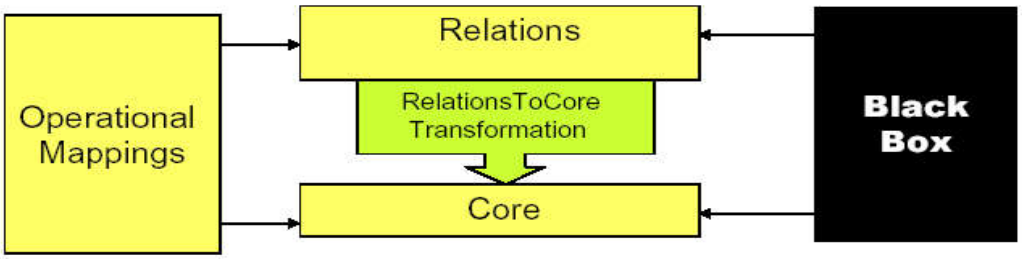
Dans cette perspective, l'OMG a lancé l'appel à propositions QVT (Query, Views, Transformations), pour le développement d'outils permettant :

- de soumettre des requêtes sur des modèles.
- de produire des vues des méta-modèles.
- d'opérer des transformations sur les modèles.

Cet appel à propositions définit une architecture de langages déclaratifs et procéduraux permettant une manipulation ergonomique d'éléments issus de modèles et de méta-modèles, et la définition de règles de transformation sur ces éléments.

7. Architecture de QVT

L'architecture proposée est la suivante :



Spécification QVT des langages de manipulation des modèles

7.1. Les langages de QVT

Langages de transformation dans QVT

3 langages et 2 modes pour définir des transformations

- Mode déclaratif
  - *Relation*

Correspondances entre des ensembles/patrons d'éléments de 2 modèles

Langage de haut niveau

- *Core*

Plus bas niveau, langage plus simple

Mais avec même pouvoir d'expression de transformations que *relation*

- Mode impératif
  - *Mapping*

Impératif, mise en oeuvre/raffinement d'une relation

Ajout de primitives déclaratives inspirées en partie d'OCL

Manipulation d'ensembles d'objets avec primitives à effet de bords

Plusieurs syntaxes pour écriture de transformation selon les langages

Syntaxe textuelle

Syntaxe graphique

QVT : langage « relation »

Une transformation est définie par un ensemble de relations

Une relation fait intervenir 2 domaines

Domaine = ensemble d'éléments d'un modèle

Relation = contraintes sur dépendances entre éléments de 2 domaines

Domaine du modèle source

Domaine du modèle cible

Une relation peut s'appliquer

Par principe quand on applique la transformation

En dépendance d'application d'une autre relation

a. **Le langage de relations** : sert à définir de manière déclarative les relations qui existent entre des éléments du modèle source et des éléments du modèle cible, suivant certaines conditions sur ces éléments. La vérification d'une relation conduira à la transformation de l'élément source en élément cible. Il est nécessaire que ce langage soit déclaratif afin que les diverses transformations nécessaires puissent se faire en parallèle. Par exemple, la transformation d'une classe UML fait appel non seulement à la relation qui concerne cette classe, mais également à la relation qui concerne les éléments englobés par la classe selon le méta-modèle (attributs et opérations entre autres). Le langage de relations doit être réflexif : on doit notamment pouvoir définir un ensemble de relations permettant de passer du langage de relations vers le langage noyau. Cet ensemble de relations est représenté par la transformation « RelationsToCore »

b. **Le langage noyau** : est un langage de bas niveau, avec une syntaxe assez simple, qui permet d'opérer un filtrage sur des ensembles de variables en évaluant des conditions issues des modèles traités. Le langage noyau est au langage de relations ce que le bytecode est au langage Java ; sa sémantique est implémentée dans une sorte de machine virtuelle.

c. **Le langage d'opérations de filtrage** : produit les mêmes effets que le langage de relations. Il s'agit d'une enveloppe procédurale du langage déclaratif des relations, permettant en particulier d'assurer une certaine facilité d'utilisation aux développeurs peu habitués aux les langages déclaratifs. Des opérations peuvent être utilisées pour aider à la description d'une relation complexe, ou encore pour décrire

une transformation entière sous forme procédurale. Ce niveau de langage fait appel au langage OCL.

d. **Le langage d'implémentation opaque** d'opérations permet de transformer des opérations MOF décrites dans un modèle. Ce langage étant très difficile à mettre en place.

8. Exigences liées à la transformation de modèles

Fournir un langage de requêtes pour les modèles permettant la sélection et le filtrage des éléments d'un modèle, en particulier pour choisir les éléments source d'une transformation.

- Fournir un langage de requêtes pour les modèles permettant la sélection et le filtrage des éléments d'un modèle, en particulier pour choisir les éléments source d'une transformation.
- Fournir un langage de définition des transformations, permettant d'établir des relations entre éléments d'un méta-modèle MOF et ceux d'un autre, décrivant comment générer un modèle cible conforme au second à partir d'un modèle source conforme au premier, sachant que l'on peut avoir le même méta-modèle au départ et à l'arrivée.
- Le langage de définition des transformations doit être capable d'exprimer toutes les informations nécessaires à l'automatisation d'une transformation.
- Le langage de définition des transformations doit permettre d'obtenir une vue d'un méta modèle.
- Le langage de définition des transformations doit être déclaratif, afin que les transformations opérées sur le modèle source soient immédiatement disponibles dans le modèle cible.
- Toutes les fonctionnalités proposées doivent être applicables à des instances de méta-modèles décrits en MOF 2.0.

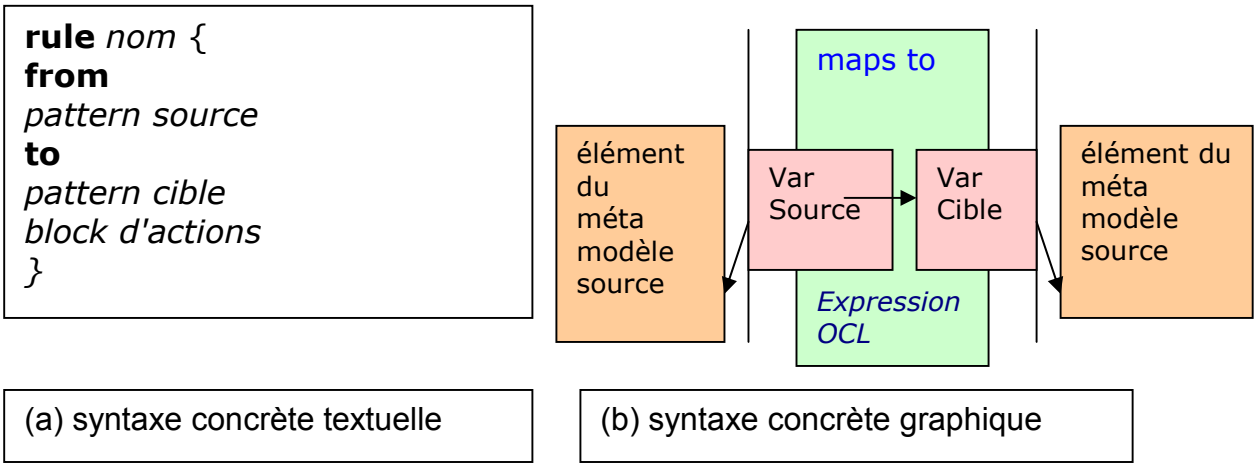
9. Transformations dans QVT

Le langage de transformation ATL appartient à l'espace technologique MDA. Certaines propositions faites dans ce langage pourraient donc avoir une influence sur le standard QVT

de l'OMG. ATL est développé par le groupe ATLAS de l'Université de Nantes en partenariat avec TNI-Valyosis.

Avantage d'ATL :

Le langage ATL a une syntaxe de base très simple. Un développeur ayant une bonne compréhension des concepts typiques d'UML pourra utiliser ATL relativement naturellement. La syntaxe graphique ne permet de fournir qu'une vision globale de la transformation, mais celle-ci est néanmoins intéressante en complément avec le texte. OCL est l'un des points forts d'ATL. Ce langage standardisé est de plus en plus connu et répandu. De plus OCL est un langage tout à fait adapté à la navigation dans les meta-modele.



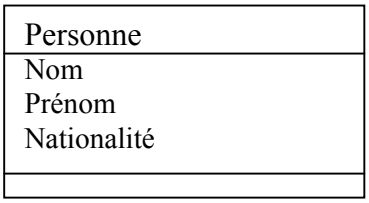
Syntaxe concrète textuelle et graphique d'une règle de transformation en ATL

9.1. Exemple 1 (transformation en ATL)

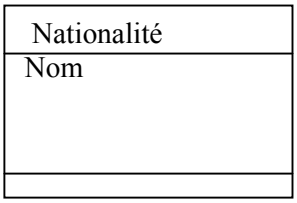
Prenons l'exemple "Liste2Liste". Le modèle d'entrée est exprimé ici en UML. Il pourrait être stocké dans un fichier Rose, Objecteering ou XMI (il sera exprimé en XMI puis en UML). Cette fois, notre fichier source « personnes » est présenté sous format d'un modèle UML. Après le processus de transformation dans ATL, nous aurons un autre modèle UML « nationalités »

Les modèles source et cible de « Liste2Liste » exprimés en UML

Ces deux modèles sont conformés aux deux méta-modèles exprimés en UML suivants :



(a) méta-modèle source



(b) méta-modèle cible

Les méta-modèles source et cible de « Liste2Liste » exprimés en UML

La règle de transformation est basée sur ces méta-modèles. Les principes de transformation sont :

Chaque instance de la classe « Personne » va être transformée en une instance de la classe «Nationalité ».

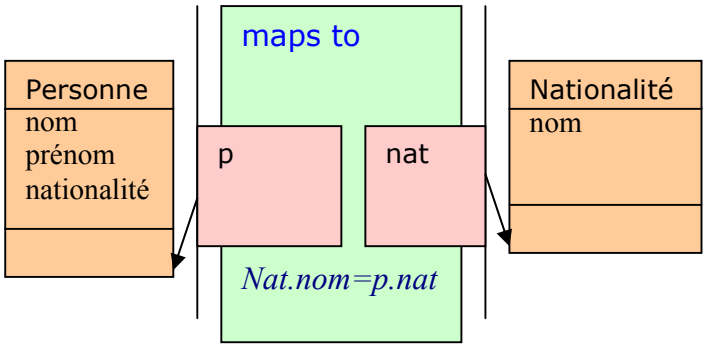
La valeur de l'attribut « nationalité » de la classe « Personne » va donner la valeur à l'attribut « nom » de la classe « Nationalité ».

Cette règle est exprimée en syntaxe concrète textuelle ci-dessous puis dans la syntaxe concrète graphique.

```
rule personne2nationalité {
from
p: personnes!Personne,
to
nat :nationalités!Nationalité (
nom <- p.nationalité
)
}
```

Règle « personne2nationalité» en syntaxe concrète textuelle

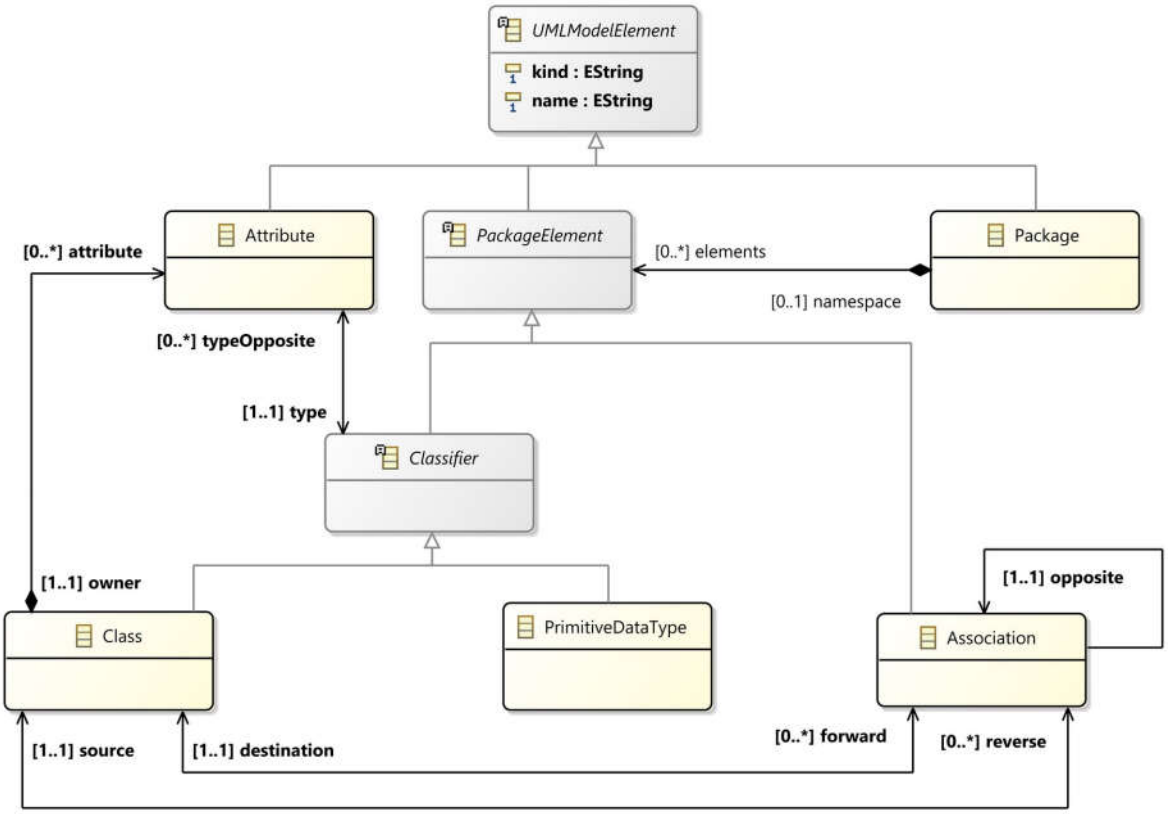
Le schéma suivant montre larègle « personne2nationalité » en syntaxe graphique



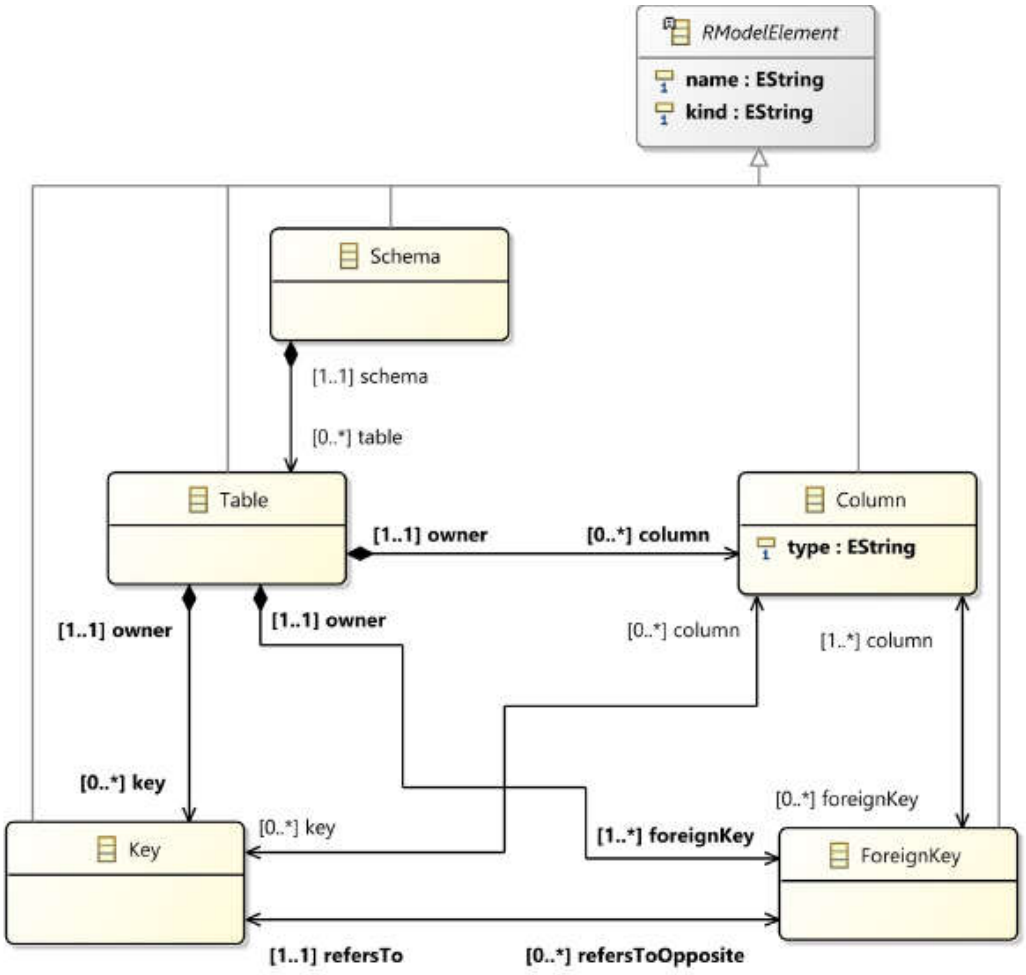
La règle « personne2nationalité » en syntaxe graphique

9.2

Exemple de transformation exogène :
D'un diagramme de classes à un schéma de base de données relationnelles
Meta modèle UML simplifié



Méta-modèle SGBDR simplifié



- ➢ But de la transformation
- Modèle de données en UML vers équivalent schéma de données relationnel et inversement
- transformation umlToRdbms(uml : SimpleUML, rdbms :

SimpleRDBMS) {
top relation PackageToSchema {…}
top relation ClassToTable {…}
relation AttributeToColumn {…}
...
}

- ➢ Etapes de la transformation
- Chaque package UML correspond à un schéma de BDD, chaque classe persistante à une table, chaque attribut de classe à une colonne de table ...
- ➢ Relations marquées avec « top »
- S'appliquent par principe
- Les autres s'appliquent si dépendantes d'autres relations
- top relation PackageToSchema

{

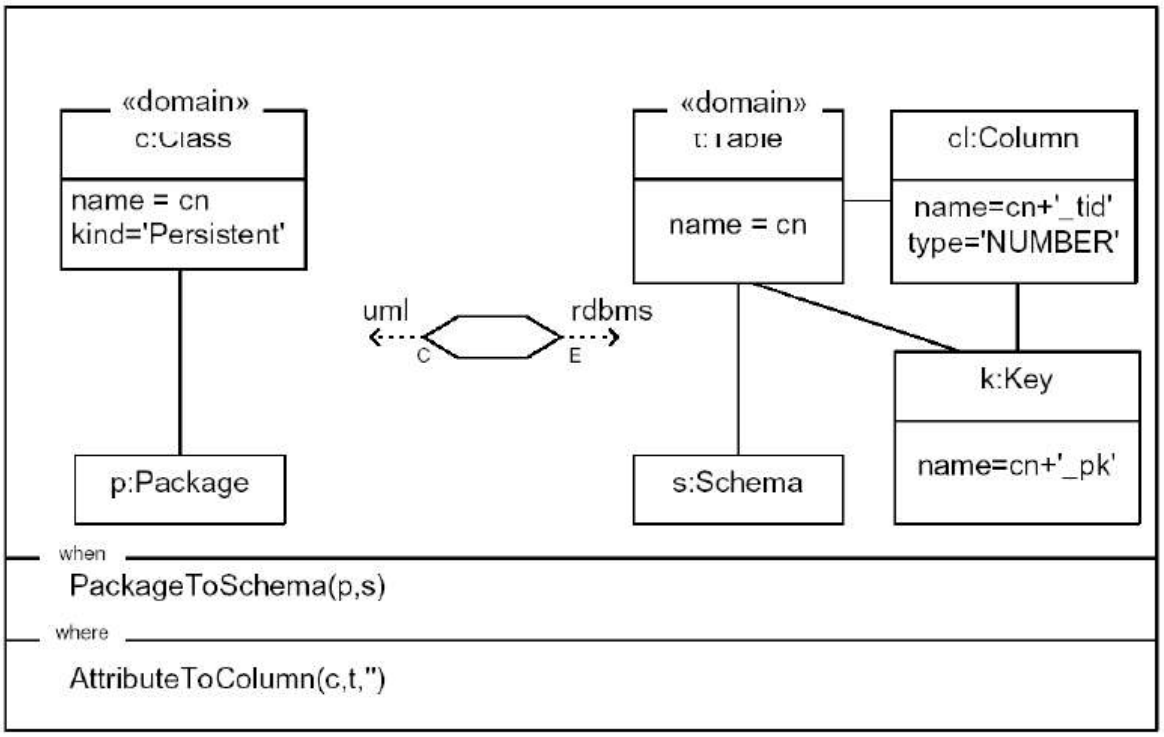
```
domain uml p:Package {name=pn}
domain rdbms s:Schema {name=pn}
}
```

- Pour chaque package UML, on a un schéma de donnée portant le même nom
- Les attributs name correspondent à la même variable pn

```
top relation ClassToTable {
domain uml c:Class {
namespace = p:Package {},
kind='Persistent',
name=cn
}
domain rdbms t:Table {
schema = s:Schema {},
name=cn,
column = cl:Column {
name=cn+'_tid',
type='NUMBER'},
primaryKey = k:PrimaryKey {
name=cn+'_pk',
column=cl}
}
when {
PackageToSchema(p, s);
}
where {
AttributeToColumn(c, t);
}
}
```

- Pour chaque classe persistante
  - On a une table avec
  - Le même nom
  - Une colonne pour l'identifiant avec nom formé à partir du nom de la classe
  - Une clé primaire avec nom formé à partir du nom de la classe
- Dépendances avec autres relations
  - « ClasseToTable » est appliquée quand on applique « PackageToSchema »
  - Et il faut appliquer aussi « AttributeToColumn » pour la classe et la table

- Relation « ClassToTable », syntaxe graphique



Types de relations entre modèles

- Transformations définies par des relations
  - Correspondances/dépendances entre 2 modèles dans un sens comme dans l'autre
  - Spécification est bi-directionnelle par défaut
  - A l'exécution, on choisit une direction de transformation
  - Exécution est mono-directionnelle
  - Modèle cible peut exister ou pas à l'exécution
  - Sera alors complété, modifié ou créé selon les cas
- Possibilité de spécialiser une transformation/relation
  - Juste vérifier si le modèle cible est cohérent rapport au modèle source (checkonly)
  - Imposer que le modèle cible soit cohérent rapport au modèle source (enforced)
- Exemple avec transformation UML/SGDBR
  - relation PackageToSchema {
    checkonly domain uml p:Package {name=pn}
    enforce domain rdbms s:Schema {name=pn}
    }
- Exécution dans le sens UML vers SGBDR
  - Source = uml, cible = rdbms
  - Le modèle cible comportera strictement un schéma pour chaque package du modèle source
  - Création des schémas manquants
  - Suppression des schémas existants mais ne correspondant pas
- Exécution dans le sens SGBDR vers UML
  - Source = rdbms, cible = uml

- Vérifie seulement, en précisant les erreurs le cas échéant, que chaque schéma du modèle source correspond à un package du modèle cible (aucune modification du modèle cible)

Langage de transformation ATL

- Langage déclaratif utilisant OCL
  - On déclare des règles qui associent un élément du modèle source à un ou plusieurs éléments générés dans le modèle cible
- Champ from
  - Sélectionne un élément du source par son type
  - Et un éventuel filtre écrit en OCL
- Champ to
  - Définit un ou plusieurs éléments du cible
  - Pour chacun, précise les valeurs de ses attributs et références
  - Avec l'affectation : <-
- Champ optionnel to
- Partie impérative d'une transformation
- Peut écrire des fonctions utilitaires en OCL (helpers)

10. Points forts de la démarche QVT

Le principal avantage des spécifications QVT est qu'elles s'appuient sur des standards préexistants, et permettent ainsi de développer des outils qui seront d'une part faciles à prendre en main pour un grand nombre d'utilisateurs, d'autre part compatibles avec un grand nombre d'outils de modélisation utilisés à l'heure actuelle.

Ces points sont les suivants :

La possibilité d'exécuter une transformation dans deux directions (le méta-modèle source peut devenir cible et inversement).

La réutilisabilité des transformations : les transformations définies pour des éléments génériques d'un modèle s'appliqueront automatiquement aux éléments spécifiques, ou pourront être redéfinies pour ces derniers.

L'exécution transactionnelle des transformations : une transformation peut être annulée en cours d'exécution.

La possibilité de fournir des données supplémentaires à l'entrée d'une définition de transformation, indiquant comment générer le modèle cible, avec éventuellement des valeurs par défaut.

La possibilité d'exécuter des transformations ayant pour cible le méta-modèle source, permettant ainsi la mise à jour de modèles.

11. **Conclusion :**

On considère les transformations comme : un complément indispensable pour l'IDM qui offre un grand ensemble de gains comme : automatiser (au moins partiellement) le passage entre modèles et/ou espaces technologiques, favoriser l'interopérabilité entre outils avec un avantage très intéressant consiste à capitaliser le savoir faire avec un plus haut niveau d'abstraction. Meilleure maîtrise, vérification, validation.
Mais mal heureusement dans l'état actuel ce complément soufre de plusieurs obstacles comme : manque de maturité des outils disponibles, comment tester une transformation ? Nombreuses contraintes techniques. Et il reste toujours un domaine de recherche