Transformation entre un profil UML et un méta-modèle conforme au MOF: Application du langage MTrans

Mikaël Peltier

France Télécom R&D, France

LRSG, Université de Nantes, France

Plan

- Contexte
  - La notion de méta-modèle
  - La notion de profil
- La transformation de modèles
- Le langage MTrans
- Exemple
- Le prototype
- Conclusion

La notion de méta-modèle

- La spécification MOF (Meta Object Facility)
  - Définir la sémantique des modèles d'un domaine particulier
    - Langage de premier niveau
- Les outils associés
  -

La notion de profil

- Le mécanisme d'extension de UML
  - Définit un ensemble de stéréotypes, de valeurs étiquetées et de contraintes pour personnaliser UML à un domaine spécifique.
  - Un profil n'étend pas UML en ajoutant de nouveaux concepts, il fournit un mécanisme pour personnaliser les concepts standards de UML à un autre domaine.
    - Langage de deuxième niveau
- Les outils associés
  - Support du mécanisme de profil par les AGL du marché

Comparaison méta-modèle/profil

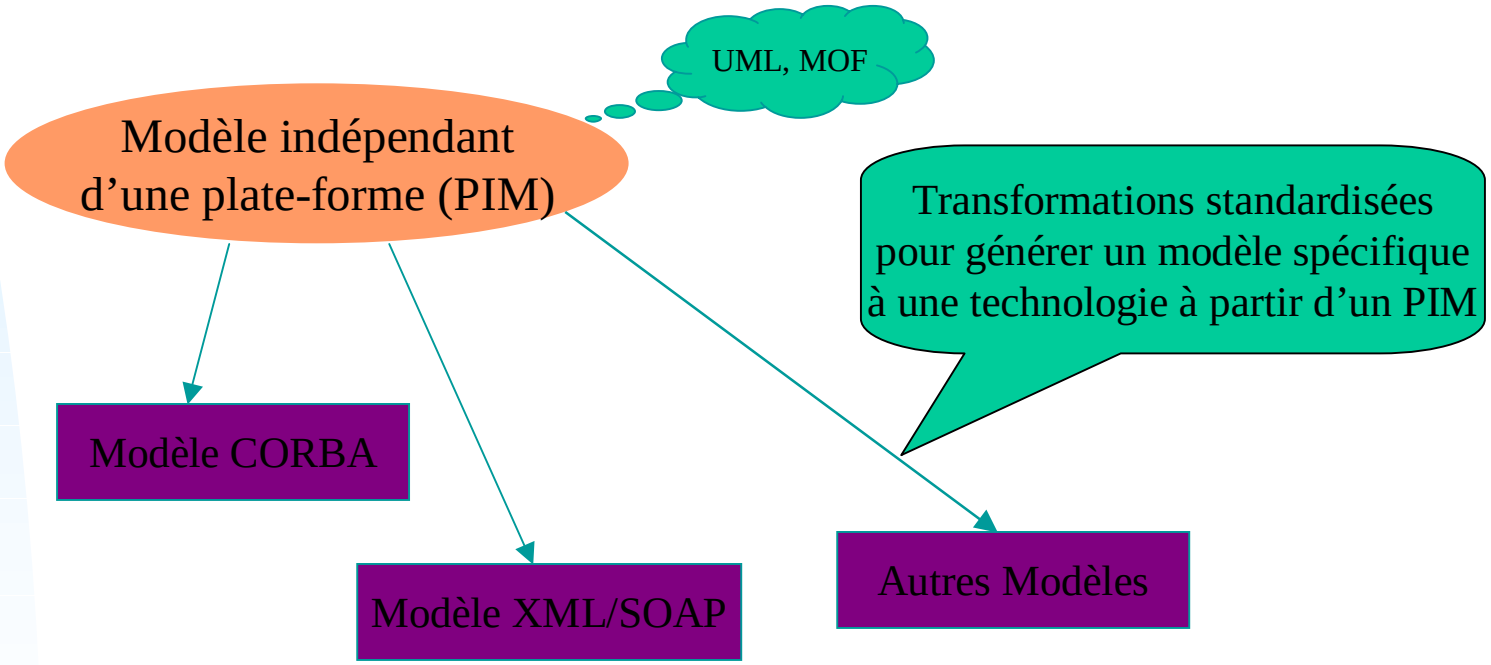
- Similarité des objectifs
  - Définir la sémantique des modèles d'un domaine particulier
- Utilisation combinée des deux technologies
  - Profil UML pour bénéficier d'un environnement interactif et graphique de modélisation
  - Référentiel MOF pour bénéficier des facilités de stockage et de manipulation.
- Résolue par transformation de modèles

La transformation de modèles (1/3)

- Transformation d'un profil UML vers un méta-modèle défini avec le MOF
- Adaptation de l'univers de discours
- Évolution des méta-modèles
- Intégration/Séparation de modèles
- Manipulation de modèles
  - Réorganisation
  - Modification
  - Application de design pattern

La transformation de modèles (2/3)

- Faire face à la prolifération des middle-wares



La transformation de modèles (3/3)

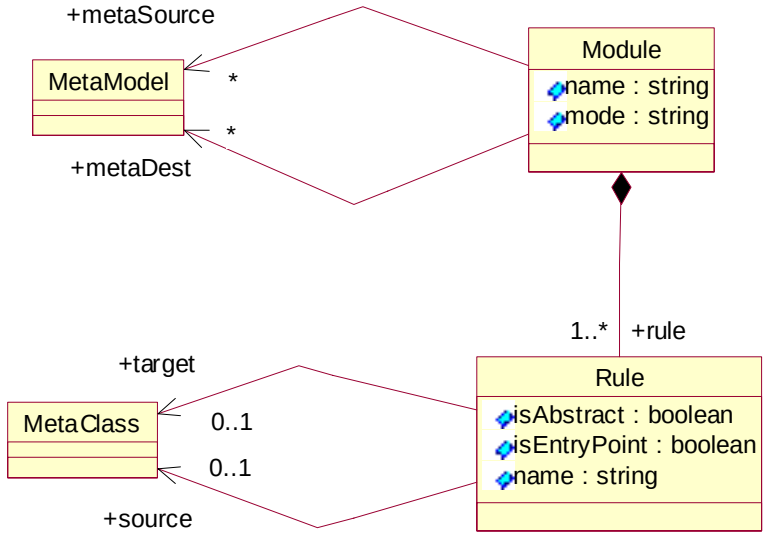
- La spécification des transformations dans MDA est informelle :
  - Tables de correspondance
  - Contraintes littérales
- ⇒ Non exécutable et sujette à ambiguïté
- Un langage formel de transformation qui soit exécutable améliorerait la définition des conversions

Le langage MTrans (1/6)

- Permet de décrire un processus de transformation de manière formelle et exécutable
- MTrans est un langage de règles
- Compromis entre syntaxe déclarative et procédurale
- Inclut les notions définies par le langage OCL
  - Raccourcis de notation
  - Nouvelles fonctionnalités

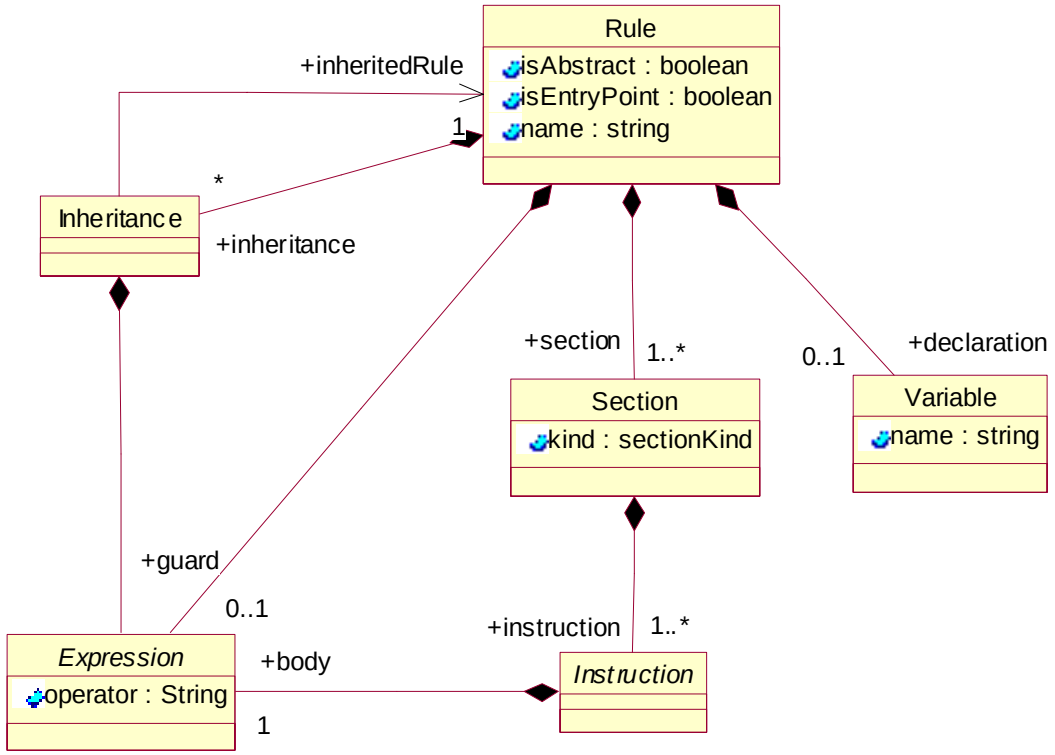
Le langage MTrans (2/6)

- La structure générale d'un programme MTrans :



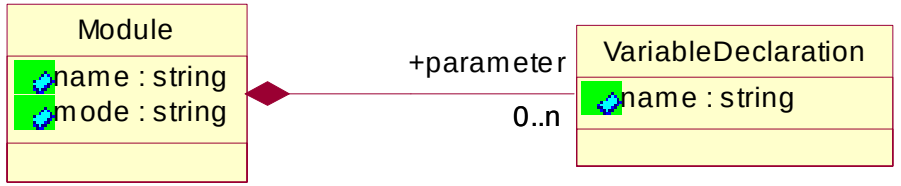
Le langage MTrans (3/6)

- La structure générale d'une règle de transformation :

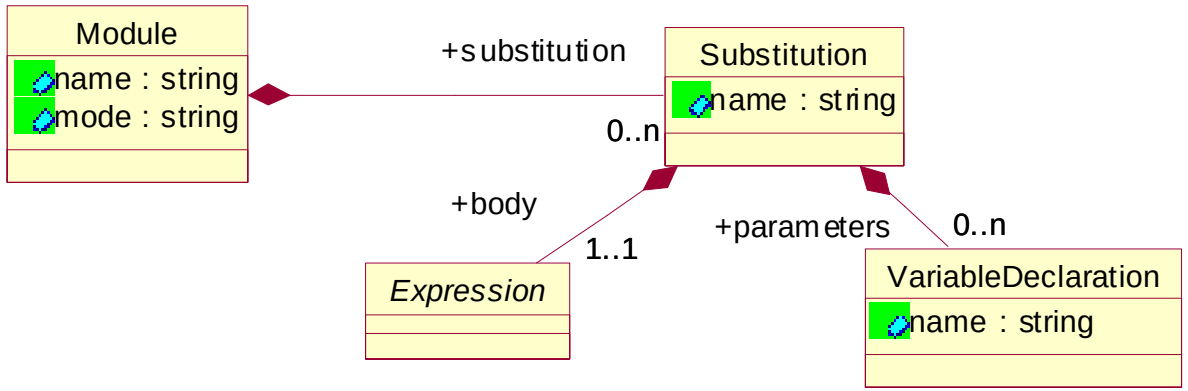


Le langage MTrans (4/6)

- Processus de transformation paramétrable

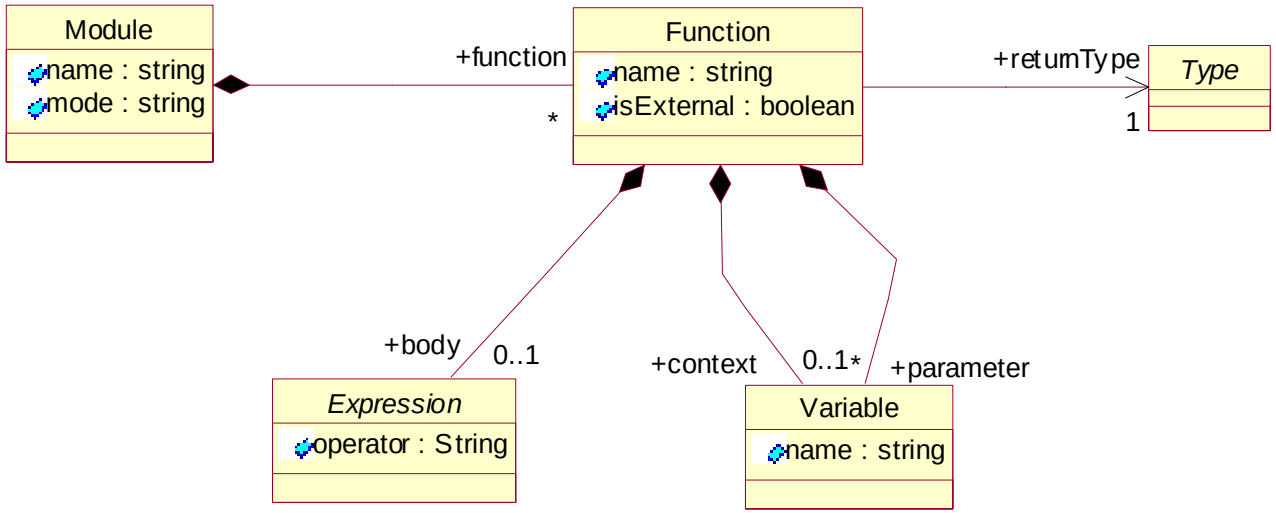


- Définition de substitutions



Le langage MTrans (5/6)

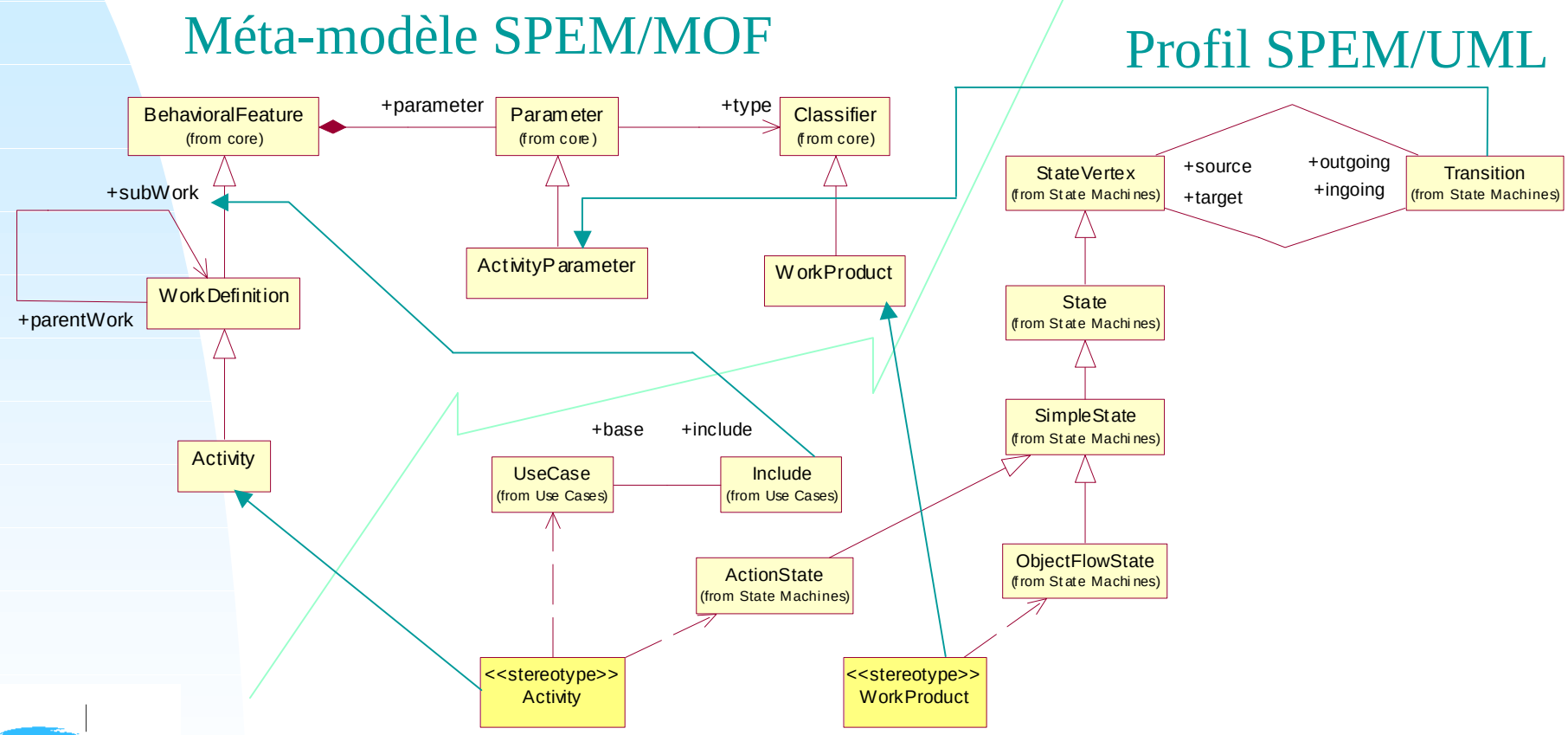
- Mécanisme d'extension (importation de fonction)
  - Permet de prendre en compte les futurs besoins
  - Permet de réaliser des traitements spécifiques



Le langage MTrans (6/6)

- La stratégie de transformation est explicite
  - Opérateur *entrypoint*
  - Opérateur *new*
- La destination d'une règle peut être indéfinie
  - Pas de section *properties et relations*
  - Utilisation de la section *init* d'une règle pour réaliser des calculs ou pour continuer de définir la stratégie

Exemple (1/2)


Méta-modèle SPEM/MOF
Profil SPEM/UML
BehavioralFeature
(from core)
+parameter
Parameter
(from core)
+type
Classifier
(from core)
+subWork
ActivityParameter
WorkProduct
WorkDefinition
+parentWork
Activity
+base
+include
UseCase
(from Use Cases)
Include
(from Use Cases)
ActionState
(from State Machines)
<<stereotype>>
Activity
<<stereotype>>
WorkProduct
StateVertex
(from State Machines)
+source
+target
+outgoing
+ingoing
Transition
(from State Machines)
State
(from State Machines)
SimpleState
(from State Machines)
ObjectFlowState
(from State Machines)

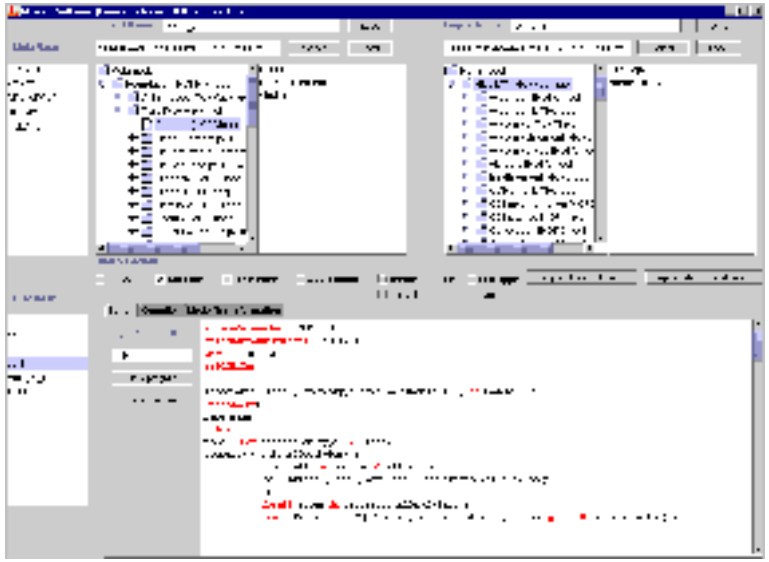
Exemple (2/2)

```
Rule [TrActivity] Activity from UseCase[stereotype.name == “Activity”]  {
attributes:
        name = name;
roles:
        subWork = replace dependencyAccess (false) ;
        parentWork = replace dependencyAccess (true);
        parameter = foreach act in ActionState [stereotype.name == "Activity" && name == self.name] {
                                act.outgoing->new [outParam] ActivityParameter;
                                act.incoming->new [inParam] ActivityParameter;
                                }
}
```

```
Rule [outParam] ActivityParameter  from Transition  {
attributes:
         name = target [ObjectFlowState].name;
         kind = "out";
roles:
        type = target [ObjectFlowState].typeState.type-> new WorkProduct();
}
```

Le prototype

- Compile un programme MTrans en XSLT
  - XMI est le standard pour la représentation XML des modèles et méta-modèles conformes au MOF



Conclusion

- Formalisme formel et exécutable pour décrire des transformations entre modèles
  - Clair et concis
  - Extensible
  - Paramétrable
  - Indépendant d'un outil spécifique (XSLT,XMI)
- Évite les erreurs d'interprétation
- Gain de temps (formalisme exécutable)
- Réduction des coûts (évolution, maintenance)