# Extension d'UML par les rôles

**Mohamed Dahchour**

*Département d'Informatique, Réseaux et Mathématiques, Institut National des Postes et Télécommunications, Av. Allal El Fassi, Madinat Al Irfane, Rabat, Maroc*

*dahchour@inpt.ac.ma*

**Hamza Rayd**

*Laboratoire de Recherche en Informatique, Intelligence Artificielle & Reconnaissance de Formes (LRIIARF), Faculté des Sciences, Université Mohammed V- Agdal Rabat*

*raydhamza@yahoo.fr*

**Younes Lakhrissi**

*Laboratoire de Recherche en Informatique, Intelligence Artificielle & Reconnaissance de Formes (LRIIARF), Faculté des Sciences, Université Mohammed V- Agdal Rabat*

*lakhrissi_younes@yahoo.fr*

**Abdelaziz Kriouile**

*Université Mhammed V-Souissi, Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes, Rabat, Maroc*

*kriouile@ensias.ma*

**Résumé**

Les associations génériques jouent un rôle important dans la modélisation conceptuelle. Les associations génériques les plus utilisées sont association, spécialisation/généralisation, classification/instantiation et agrégation/décomposition. D'autres associations génériques ont été identifiées dans la littérature, telle role-of qui permet de représenter les aspects dynamiques des objets. Ces aspects dynamiques ne peuvent pas être correctement modélisés ni par les associations génériques offertes par UML ni par son propre concept de rôles relatifs aux diagrammes de collaborations. Pour combler cette lacune, le présent travail propose une extension d'UML par l'association role-of. De nouvelles métaclasses et règles OCL sont ajoutées au métamodèle de base d'UML pour capturer la sémantique des rôles.

**Abstract**

Generic relationships play an important role in conceptual modeling. The most used generic relationships are association, specialisation/generalisation, classification/instantiation and aggregation/decomposition. Other generic relationships have been identified in the literature, such as role-of which represents the dynamic aspects of objects. These dynamic aspects can be correctly modeled neither by the generic relationships offered by UML nor by its own concept of roles involved in collaboration diagrams. To fill this gap, this work proposes an extension of UML by role-of. New metaclasses and OCL rules are added to the basic metamodel of UML to capture the semantics of roles.

**Mots-clés**

rôles, métamodélisation, UML, langage de définition des contraintes, OCL

**Keywords**

roles, metamodeling, UML, object constraint language, OCL

# 1. Introduction

La modélisation conceptuelle (Mylopoulos, 1998) consiste à construire des représentations abstraites de certains aspects des systèmes physiques et sociaux et de leur environnement dans le monde qui nous entoure.

Des progrès en modélisation conceptuelle consistent à définir des mécanismes d'abstraction de modélisation simples, puissants et de haut niveau, qui permettent de réduire la distance entre les concepts familiers aux acteurs dans les domaines d'application et leur représentation dans les modèles. Les associations génériques (Dahchour et al., 2005) sont des exemples typiques de tels mécanismes d'abstraction. Les associations génériques les plus connues sont association, spécialisation/généralisation, classification/instanciation, et agrégation/décomposition. D'autres associations génériques ont été identifiées dans la littérature, telle role-of (Dahchour et al., 2002) (Wieringa et al., 1995), (Gottlob et al., 1996), (Kristensen, 1996), (Wong et al., 1997), (Steimann, 2000(b))

L'association role-of associe une classe de rôles (par exemple, Employé, Etudiant) à une classe de base dite classe d'objets (par exemple, Personne). Les instances de la classe de rôles représentent des rôles joués par les instances de la classe d'objet. Par exemple, Ali, initialement instance de Personne peut devenir plus tard une instance de Etudiant et/ou une instance d'Employé. Ainsi, pour role-of il est possible pour un objet, instance d'une classe, de changer son état et devenir, en séquence ou simultanément, une instance d'une autre classe.

L'association role-of est proche de la généralisation associant une sous-classe (par exemple, Voiture, Bus) à une superclasse (par exemple, Véhicule) mais elle en diffère par son aspect dynamique. En effet, contrairement à role-of, dans le contexte de la généralisation, un objet est une instance statique d'une classe. En effet, une instance de Voiture, ne peut jamais devenir une instance de la classe Bus.

Le modèle de données d'UML (OMG, 2004(a)), le standard de modélisation objet, comprend trois associations génériques principales : association, généralisation, et dépendance existentielle. A partir de ces trois associations de base, UML définit un certain nombre de catégories d'associations par son mécanisme de stéréotype.

Actuellement, il n'y a pas d'équivalent de l'association role-of dans UML. Par conséquent, quand on utilise UML pour modéliser une application on se trouve contraint de représenter les aspects dynamiques des objets par les associations disponibles dans UML, notamment la généralisation. Ceci résulte en une modélisation incorrecte avec toutes les conséquences que cela peut avoir sur les étapes de conception et d'implémentation.

Notons que UML (OMG, 2004(a)) définit un certain concept de rôle qui est complètement différent de celui associé à l'association role-of. En effet, UML distingue entre les rôles statiques et les rôles dynamiques. Les premiers sont hérités du modèle entité-association. Ils caractérisent la participation d'une classe dans une association (Chu et al., 1997). Les rôles dynamiques sont hérités des méthodes à base d'objets telle que OORAM (Reenskaug et al., 1996). Ce sont les objets qui participent dans les diagrammes de collaborations d'UML. Une étude détaillée sur les rôles d'UML et les problèmes qu'ils posent a été réalisée par (Steimann, 2000(a)). Elle sera discutée dans la section sur les travaux connexes.

L'objectif du présent travail est d'étendre UML par l'association role-of définie dans (Dahchour et al., 2002). Ce travail étend celui présenté dans (Dahchour et al., 2006).

Cet article est structuré comme suit. La section 2 présente notre modèle des rôles. La section 3 décrit l'architecture à quatre couches d'UML. La section 4 décrit le processus d'intégration des rôles dans UML. La section 5 discute les travaux connexes. La section 6 conclut le travail.

# 2. L'association Role-Of

Dans cette section nous présentons de manière très succincte la sémantique de l'association role-of. Les détails de cette sémantique sont définis dans (Dahchour et al., 2002), (Dahchour et al., 2004).

## 2.1. Structure générale

L'association role-of (schématisée par O ○←R) relie une classe, appelée classe d'objet (O), et une autre classe, appelée classe de rôles (**R**), qui décrit les rôles dynamiques pour la classe d'objets **O**. On dit que les instances de la classe d'objets **O** gagnent des rôles (aussi qu'ils jouent des rôles), lesquels sont des instances de la classe de rôles **R**. Les classes d'objet sont utilisées pour représenter les propriétés statiques des entités alors que les classes de rôles sont utilisées pour modéliser leurs aspects dynamiques. Une entité non évolutive est représentée comme une instance permanente et exclusive d'une classe. Une entité évolutive, en plus d'être une instance permanente et exclusive d'une classe d'objet, est représentée par un ensemble d'instances de la classe de rôles. Quand une entité gagne un nouveau rôle, une nouvelle instance de la classe des rôles appropriée est créée. Si elle perd un rôle, l'instance de la classe des rôles correspondante est supprimée. Par la suite, les instances de la classe d'objets s'appelleront des objets, tandis que les instances de la classe des rôles s'appelleront des rôles.
La Figure 1 montre deux associations role-of qui relient la classe d'objets **Personne** aux classes de rôles **Etudiant** et **Employé**. La classe **Personne** définit les propriétés permanentes des objets personne: *nom, adresse,* et *téléphone*. La classe **Etudiant** définit certains de leurs propriétés transitoires en tant qu'étudiants : *univ, etud#, spécialité,* et *cours* qui est un attribut multi-valué. De la même façon, la classe **Employé** définit certains de leurs propriétés transitoires en tant qu'employés: *Dept, emp#, fonction,* et *salaire*.

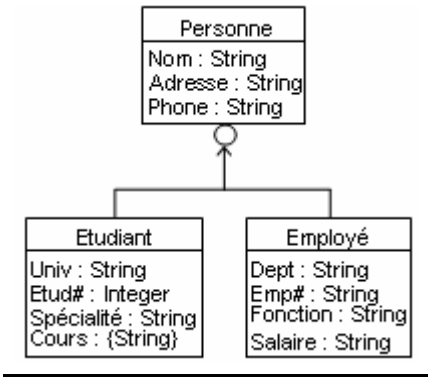


Figure 1. Un exemple de l'association role-of.

Notre modèle de rôles supporte également l'instanciation multiple de la même classe. En effet, un objet peut devenir une instance plus qu'une fois de la même classe, tout en perdant ou maintenant son appartenance à la classe. Par exemple, un étudiant peut être inscrit dans deux universités différentes. Il jouera donc deux rôles différents.

## 2.2. Changement dynamique de la classe

Soit une classe d'objets **O** en relation avec les classes des rôles $R_1$ et $R_2$. Les objets peuvent changer leur classification selon les différents cas décrits ci-dessus.

- **L'évolution** notée $R_1 \rightarrow^{ev} R_2$ pour les classes de rôles $R_1$ et $R_2$, avec $R_1$ distincte de $R_2$ : cette notation indique qu'un objet jouant le rôle r1 (instance de $R_1$) peut perdre r1 et gagner le rôle r2 (instance de $R_2$). En plus, les rôles de $R_2$ ne peuvent pas être créés indépendamment des rôles de $R_1$. Une instance de $R_2$ est créée nécessairement par évolution d'une instance de $R_1$. Les évolutions peuvent être

bidirectionnelles ou unidirectionnelles selon que le rôle perdu par $R_1$ peut être récupéré plus tard ou pas. Des exemples d'évolutions bidirectionnelles sont "Célibataire $\leftrightarrow^{ev}$ Marié" et "Employé $\leftrightarrow^{ev}$ Chômeur". Des exemples d'évolution unidirectionnelle sont " Employé $\rightarrow^{ev}$ Retraité" et "Etudiant $\rightarrow^{ev}$ Lauréat".

- **L'extension** notée $R_1 \rightarrow^{ext} R_2$, avec $R_1$ non nécessairement distincte de $R_2$, indique que l'objet qui joue le rôle r1 de $R_1$ peut gagner un nouveau rôle r2 de $R_2$ en retenant r1. Des exemples de ce type de transition sont "Etudiant $\rightarrow^{ext}$ Employé" et "Professeur $\rightarrow^{ext}$ ChefDept". Lorsque la classe source R et la classe cible sont les mêmes, cela autorise l'objet qui joue le rôle r1 de R de gagner un nouveau rôle r2 de la même classe R, tout en retenant r1. Les exemples suivants montrent cet aspect : "Etudiant $\rightarrow^{ext}$ Etudiant" quand une personne est simultanément étudiant dans plus d'une université, et "Employé $\rightarrow^{ext}$ Employé" quand une personne est simultanément employée dans plus d'une compagnie.
- **L'acquisition**, notée $\rightarrow$R, indique qu'un objet peut acquérir librement un rôle de R. Par exemple, "$\rightarrow$Etudiant" signifie que les personnes peuvent devenir des étudiants. Remarquez que "$\rightarrow$Etudiant" pourrait être noté comme "Personne $\rightarrow^{ext}$ Etudiant"**,** où la classe Personne est la classe d'objets pour la classe des rôles Etudiant.
- **La perte**, notée R$\rightarrow$, indique qu'un objet peut perdre librement un rôle de R. Par exemple, "Etudiant$\rightarrow$" signifie que les personnes cessent d'être des étudiants. Remarquez que "Etudiant$\rightarrow$" pourrait être noté comme "Etudiant$\overset{ev}{\rightarrow}$ Personne", où la classe Personne est la classe d'objets pour la classe des rôles Etudiant.

## 2.3. Prédicats de transition.

Soit une classe d'objets O en relation avec les classes des rôles R1 et R2. Un prédicat de transition est associé à R1 pour décrire les conditions nécessaires et/ou suffisantes sur la façon dont les objets jouant des rôles de R1 peuvent explicitement ou automatiquement acquérir des rôles dans R2. La classe source R1 peut être une classe des rôles (dans une évolution ou une extension) ou une classe d'objets (dans une acquisition).
Les prédicats de transition sont décrits en utilisant le langage déclaratif formel OCL (Object Constraint Language) d'UML (Clark et al., 2002). Dans ce qui suit, nous donnons quelques exemples de prédicats de transition :

- Employé $\leftrightarrow^{ev}$ Retraité : le prédicat *âge* $\geq$ *55* $\Lambda$ *TempsTravail* $\geq$ *20* associé aux employés qui deviennent des retraités.
- Employé $\leftrightarrow^{ev}$ Chômeur : le prédicat *contractExpirDate* $\leq$ *DateCourante* associé aux employés qui perdent automatiquement leur emploi lorsque la date de contrat est expirée. Un autre prédicat associé aux chômeurs qui peuvent devenir employés en signant un nouveau contrat de travail.
- Professeur$\rightarrow^{ext}$ ChefDept : le prédicat *grade=Professeur de l'Enseignement Supérieur* $\Lambda$ *expérience = 6 ans* associé aux professeurs qui deviennent des chefs de départements.
- Etudiant $\rightarrow^{ext}$ Etudiant : le prédicat *nbProgs* $\leq$ *max* déclare le maximum de programmes auxquels un étudiant peut s'inscrire.
- $\rightarrow$ Employé : le prédicat *âge* $\geq$ *18* attaché aux personnes, montre que seuls les adultes peuvent être des employés.

# 3. L'architecture d'UML

Cette section décrit l'architecture à quatre couches d'UML, les paquetages logiques permettant d'organiser les différents métamodèles du langage et le formalisme de spécification du métamodèle d'UML.

## 3.1. Métamodélisation

L'architecture d'UML est basée sur une structure à quatre niveaux d'abstraction: M0 (objets ou données utilisateurs), M1 (modèle), M2 (méta-modèle) et M4 (méta-méta-modèle) (OMG, 2004(a)). Chaque niveau Mi est vu comme "instance" du niveau supérieur Mi+1 (cf. figure 2). Certains chercheurs parlent plutôt de conformité entre les niveaux Mi et Mi+1[1].

- – M0 (objet) : C'est la couche où réside les objets factuels souvent appelés « données utilisateurs ». Cette couche d'abstraction est utilisée pour formaliser des expressions spécifiques à un objet donné.
- – M1 (modèle) : C'est la couche qui héberge les modèles UML développés pour spécifier un domaine d'application donné. Elle décrit la structure de la couche M0.
- – M2 (méta-modèle) : C'est la couche qui définit les éléments syntaxiques et sémantiques des modèles du niveau M1. Les concepts UML, "Class", "Attribute", "Instance", etc., sont définis à ce niveau.
- – M3 (méta-métamodel) : Cette couche définit un langage de haut niveau pour spécifier les métamodèles. Elle est instance d'elle même. Le métamodèle d'UML (niveau M2) en est une instance particulière. Ce langage est aussi connu sous le nom de MOF (Meta Object Facility) (OMG, 2004 (c)).

Nous nous intéressons dans ce travail à la couche M2 (méta-modèle) qui sera détaillée dans la sous-section suivante.

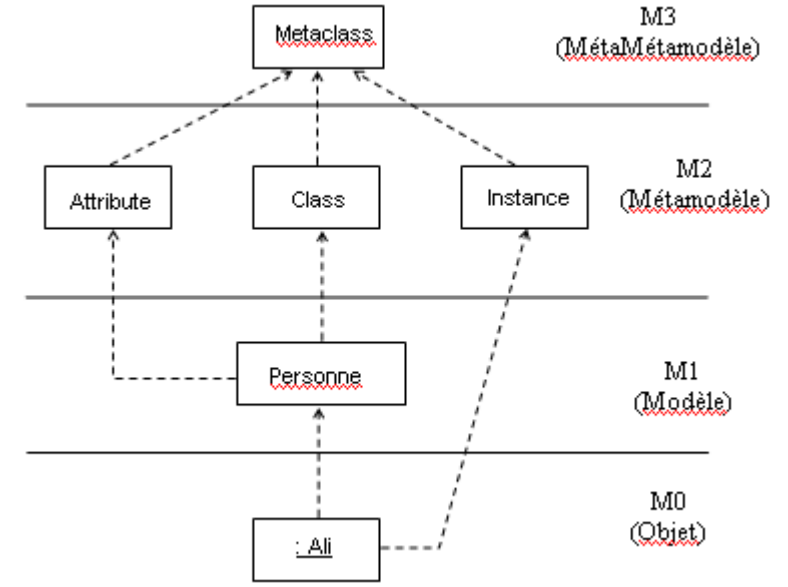


Figure 2. L'architecture à 4 niveaux d'UML

## 3.2. Les paquetages logiques du métamodèle d'UML

La complexité du métamodèle d'UML (niveau M2) est géré en organisant celui-ci en des paquetages logiques.

[1] Par exemple, (Bézivin, 2004) propose de remplacer la structure classique à quatre niveaux de l'OMG par une architecture 3+1 composée de niveaux reliés par les associations *representedBy* et *conformsTo* au lieu de *instanceOf*. La strate M0 correspond au système réel représenté par un modèle du niveau M1. Ce modèle est conforme à son métamodèle défini au niveau M2 et le métamodèle lui-même est conforme au métamétamodèle au niveau M3 . Le métamétamodèle est conforme à lui-même.

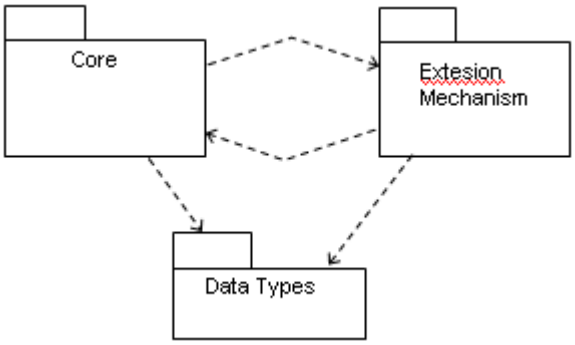


Figure 3. Les sous-paquetages du paquetage Foundation

Ces paquetages regroupent des métaclasses qui représentent une forte cohésion interne et un faible couplage avec les métaclasses d'autres paquetages. Cette couche comprend, entre autres, le paquetage Foundation. Ce paquetage est l'infrastructure du langage UML qui définit la structure statique des modèles du niveau M1. Il est composé de trois sous-paquetages : Core, Extension Mechanisms et Data Types (cf. figure 3).
Le paquetage Core définit les concepts de base pour la construction d'un modèle objet. On y distingue deux types de constructions, abstraites et concrètes. Les constructions abstraites ne sont pas instanciables et sont généralement employées pour réifier les constructions de base et organiser le méta-modèle d'UML. Les constructions abstraites définies dans le paquetage Core incluent ModelElement, Relationship et Classifier. Un élément quelconque du modèle est une sous-classe directe ou indirecte de ModelElement, héritant de lui l'attribut name, qui confère un nom à n'importe quel élément existant dans UML. Les constructions concrètes sont, par contre, instanciables. Elles incluent Class, Attribute, Operation et Association.
Le paquetage Extension Mechanisms offre des mécanismes permettant d'ajouter de nouveaux éléments dans UML sans devoir changer sa structure de manière substantielle.
Le paquetage Data Types définit les types de données de base utilisés dans la spécification des autres paquetages.
Une extension substantielle d'UML affecte le paquetage Core. Dans le cadre de notre travail, la plupart des concepts requis pour la définition des rôles seront incorporés dans ce paquetage.

## 3.3. Formalisme de spécification du méta-modèle d'UML

Nous rappelons ci-dessous les éléments du formalisme de spécification du métamodèle d'UML que nous respectons dans nos définitions. Dans UML, un méta-modèle est décrit d'une façon semi-formelle en utilisant les trois vues suivantes :

- Syntaxe abstraite (Abstract syntax),
- Règles de bonne conception (Well-formedness rules)
- Sémantique (Semantics).

### 3.3.1. Syntaxe abstraite

La syntaxe abstraite est présentée à l'aide d'un diagramme de classes UML. Elle définit les constructions et leurs liens avec d'autres constructions. Elle peut être accompagnée d'une description non formelle en langage naturel décrivant chaque méta-classe et chaque association du diagramme.

### 3.3.2. Règles de bonne conception

Il s'agit d'un ensemble de règles définissant la sémantique statique des différentes constructions du diagramme de classes défini dans « la syntaxe abstraite » ci-dessus. Chacune de ces règles est définie par une expression logique en OCL (Object Constraint Language)

(Clark et al., 2002), (OMG, 2004(b)) accompagnée éventuellement de clarifications exprimées en langage naturel.

### 3.3.3. Sémantique

Elle décrit la sémantique dynamique des différentes constructions définies dans « la syntaxe abstraite ». Elle est décrite principalement en langage naturel, mais peut inclure une certaine notation additionnelle, selon le modèle décrit.

# 4. Intégration des rôles dans UML

Nous présentons ci-dessous le processus d'intégration de l'association role-of dans UML en respectant les éléments du formalisme de spécification présentés dans la Section 3.

## 4.1. Syntaxe abstraite

Nous définissons ici le métamodèle de l'association role-of présentée dans la section 2.

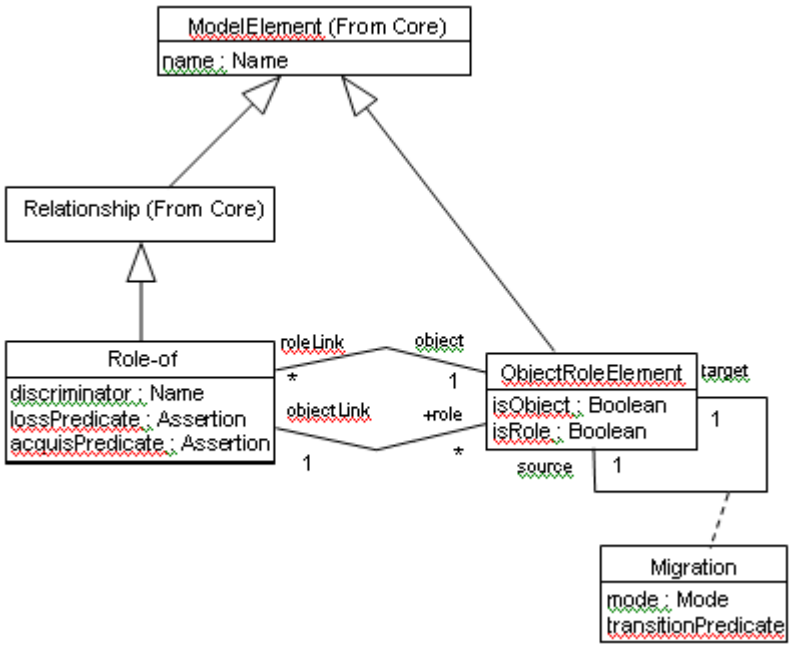


Figure 4. Le métamodèle de l'association role-of.

Nous définissions role-of comme sous-classe directe de Relationship et ObjectRoleElement comme sous-classe directe de ModelElement. La métaclasse ObjectRoleElement représente les classes d'objets et les classes de rôles impliquées dans une association role-of. Notre métamodèle est reporté à la Figure 4. Nous allons détailler dans la suite chacun de ses éléments. Pour mieux clarifier les éléments du métamodèle de la figure 4, la figure 5 présente une association role-of reliant deux classes A et B.

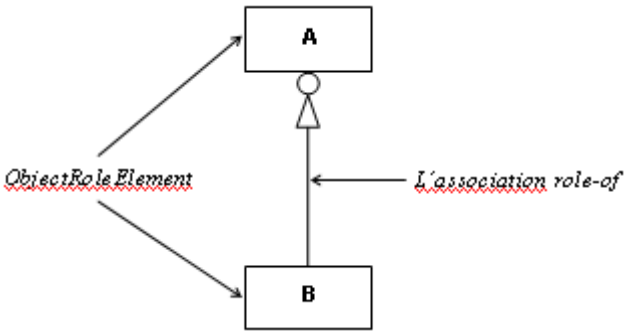


Figure 5. Exemple de role-of.

Les classes A et B sont des instances de la métaclasse ObjectRoleElement. Par abus de langage on dira que A et B sont des ObjectRoleElement avec les précisions suivantes :

- A (la classe d'objets) est un ObjectRoleElement avec l'attribut isObject=True et isRole=False.
- B (la classe de rôles) est un ObjectRoleElement avec l'attribut isRole=True et isObject=False.

Donc A et B de la figure 5 représentent, respectivement, les qualificateurs object et role de la figure 4. L'association role-of entre A et B, peut être décomposée en deux sous-liaisons : objectLink (reliant un rôle à un objet) et roleLink (reliant un objet à un rôle) comme le montre la figure 4.

### 4.1.1. Signification des cardinalités

- **1** du côté du qualificateur *object* : un seul ObjectRoleElement de type *classe d'objects* (c-à-d un ObjectRoleElement avec les attributs isObject=true) participe dans l'association role-of.
- **1** du côté du qualificateur *role* : un seul ObjectRoleElement de type *classe de rôles* (c-à-d un ObjectRoleElement avec les attributs isRole=true) participe dans l'association role-of.
- **1** du côté du qualificateur *objectLink* : un ObjectRoleElement ne peut avoir qu'une seule liaison envers la classe d'objets.
- ***** du côté du qualificateur *roleLink* : un ObjectRoleElement peut avoir plusieurs liaisons envers des classes de rôles.

### 4.1.2. ObjectRoleElement

C'est une sous-classe de « ModelElement » qui peut participer dans une association role-of. Un ObjectRoleElement est une méta-classe abstraite.
Attributs :

- **isObject :** il est de type Booléen. Il a la valeur True pour une classe d'objets.
- **isRole :** il est de type Booléen. Il a la valeur True pour une classe de rôles.

Pour les classes intermédiaires telle que R1 dans la composition de role-ofs R2 →○R1→○ O, les attributs isObject et isRole auront tous les deux la valeur « vrai », puisque R1 est à la fois une classe d'objets et une classe de rôles.
Associations :

- **RoleLink :** c'est la liaison partant de la classe d'objets vers la classe de rôles.
- **ObjectLink :** c'est la liaison partant de la classe de rôles vers la classe d'objets.

### 4.1.3. L'association role-of

Dans le méta-modèle, role-of est une sous classe directe de Relationship. Elle représente le lien entre une classe d'objets et une classe de rôles.

Attributs :

- **discriminateur :** indique la partition à la quelle le lien de role-of appartient. Il désigne tous les liens de role-of qui partagent la même classe d'objets. Chaque partition représente une dimension orthogonale qui regroupe un ensemble de classes de rôles correspondant à une classe d'objets. Les classes de rôles de la même partition ont tous le même nom de discriminateur.
- **lossPredicate :** une assertion décrivant les conditions nécessaires et/ou suffisantes sur la façon dont les objets peuvent explicitement ou automatiquement perdre des rôles.
- **acquisPredicate :** une assertion décrivant les conditions nécessaires et/ou suffisantes sur la façon dont les objets peuvent explicitement ou automatiquement gagner des rôles.

Associations :

- **object :** désigne un *ObjectRoleElement* qui peut jouer un rôle.
- **role :** désigne un *ObjectRoleElement* qui représente un état possible d'un objet.

### 4.1.4. Classe-association Migration

L'association Migration est une classe-association récursive qui relie une classe d'ObjectRoleElement appelée classe source et une classe d'ObjectRoleElement appelée classe cible (target). Elle permet de spécifier les notions d'évolution et d'extension (voir section 2.2) ainsi que les prédicats de transition d'objets (voir section 2.3).

Attributs :

- **mode** est un type énuméré, le rôle de cet attribut est de définir le mode de migration, il peut prendre une des deux valeurs : « evolution » ou « extension ».
- **transitionPredicate** est de type Assertion. Il décrit les conditions pour qu'une instance de la classe source puisse migrer vers la classe destination.

Associations :

- **source** désigne une classe de rôles (c-à-d une classe ObjectRoleElement avec l'attribut isRole=true) qui va subir les règles de l'évolution ou d'extension.
- **target** désigne la classe des rôles destination après la vérification des conditions de transition sur la classe source.

## 4.2. Les règles de bonne conception

### 4.2.1. Opérations additionnelles

En plus des opérations prédéfinies dans OCL, nous définissons quelques autres opérations qui seront utilisées dans les règles OCL ci-dessous.

- *object* : fonction qui retourne l'object de ObjectRoleElement qui l'a appelé.
- *object* = self.objectLink.object
- *role* : fonction qui retourne l'ensemble de rôles de ObjectRoleElement qui l'a appelé.
- *role* = self.roleLink.role
- *getId()* : fonction qui retourne l'identifiant de ObjectRoleElement concerné.
- *destroy()* : fonction qui détruit l'ObjectRoleElement qui l'a appelé.
- *make()* : fonction qui crée un rôle pour l'ObjectRoleElement qui l'a appelé.

### 4.2.2. ObjectRoleElement.

- Cardinalités : chaque instance de la classe des rôles est en relation avec exactement une instance de sa classe d'objets.

***Context ObjectRoleElement inv***
***Self.objectLink.object→ size() = 1***

De même, chaque instance de la class d'objet peut être en relation avec plusieurs instances de la classe des rôles, ceci n'exprime aucune contrainte à devoir écrire avec OCL.

- Identité de l'objet : une instance de la classe des rôles a son propre rid (role-identity), différent de toutes les autres instances de la classe des rôles et aussi de toutes les instances de la classe d'objets.

***Context inv ObjectRoleElement***
***ObjectRoleElement.allInstances → forAll(p1,p2 \| p1<>p2 implies p1.getId()<>p2.getId())***

- Un ObjectRoleElement qui est racine (c.à.d isObject=true et isRole=false) ne peut avoir aucun objectLink.

***Context ObjectRoleElement inv***
***Self.isObject and (not self.isRole) implies self.objectLink→ isEmpty()***

- Un ObjectRoleElement qui est Leaf (c.à.d isObject=false et isRole=true) ne peut avoir aucun roleLink.

***Context ObjectRoleElement inv***
***Self.isRole and (not self.isObject) implies self.roleLink→ isEmpty()***

- On ne peut pas créer un rôle sans avoir créer au préalable l'objet correspondant.

***ObjectRoleElement inv***
***Self.isRole implies self.objectLink.object→ notEmpty()***

- Des changements (suppression, création) dans la classe des rôles n'affecte pas la classe d'objets, mais l'inverse n'est pas vrai, notamment dans le cas où on supprime l'objet tous les rôles de cet objet sont automatiquement supprimés.

***Role-Of inv***
***self.object.destroy() implies***
***self.role→forAll ( r \| r.destroy())***

### 4.2.3. Role-of

- La perte d'un rôle est contrôlée par l'attribut lossPredicate du modèle role-of. Quand l'assertion lossPredicate est satisfaite, on détruit le rôle de cette liaison. La règle exprimant cette action est comme suit.

***ObjectRoleElement inv***
***Self.roleLink→forAll(p \| p.lossPredicate.isSatisfied***
***implies p.role.destroy())***

- L'acquisition d'un rôle est contrôlée par l'attribut acquisPredicate du modèle role-of. Quand l'assertion acquisPredicate est satisfaite on construit un nouveau rôle pour cette liaison. La règle exprimant cette action est la suivante:

***ObjectRoleElement inv***
***Self.roleLink→forAll(p \| p.acquisPredicate.isSatisfied***
***implies p.role.make ())***

### 4.2.4. Classe-association Migration

L'attribut Mode de la classe migration est de type Mode, qui est un stéréotype du type Enumeration. Il définit l'ensemble des modes de transition entre la classe source et la classe destination, il y a deux cas possibles : évolution et extension (cf. Figure 6).

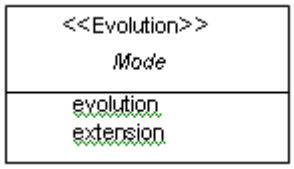


Figure 6. Définition du type Mode.

Règles associées aux modes « évolution » et « extension ».
A la classe Migration nous associons deux règles régissant les modes « évolution » et « extension » des rôles définis dans la section 2.2.

Pour l'évolution, la règle doit vérifier les conditions suivantes :

- Le mode de migration est 'évolution'.
- Les classes source et destination sont des classes de rôles.
- Satisfaction du prédicat **transitionPredicate** de la classe migration.
- Le résultat de la règle est la destruction du rôle de la classe source et la création du rôle de la classe destination.

La règle OCL exprimant ces contraintes est la suivante :

***Context Migration inv :***
***(Self.mode=Mode::evolution and self.source.isRole=true and self.target.isRole=true and self.transitionPredicate.isSatisfied ())***
***Implies (self.source.destroy() and self.target.make())***

Pour l'extension, la règle doit vérifier les conditions suivantes :

- Le mode de migration est 'extension'.
- Les classes source et destination sont des classes de rôles.
- Satisfaction du prédicat **transitionPredicate** de la classe migration.
- Le résultat de la règle est la création du rôle de la classe destination. La règle OCL exprimant ces contraintes est la suivante :

***Context Migration inv :***
***(self.mode=Mode::evolution and self.source.isRole=true and self.target.isRole=true and self.transitionPredicate.isSatisfied())***
***implies (self.target.make() )***

### 4.2.5. Eléments standard

Dans ce paragraphe on présentera le nouveau type de données qu'on a déjà utilisé dans notre modèle qui est Assertion (cf. Figure 7).

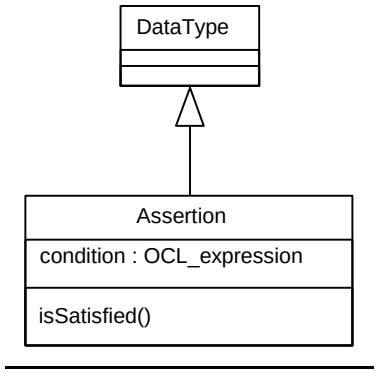


Figure 7. Le type Assertion défini comme sous-classe de DataType

Ce type est utilisé pour définir les prédicats tels que acquisPredicate, lossPredicate et transitionPredicate. Ce type est constitué par l'attribut condition et la méthode isSatisfied() :

- **condition** : est une expression écrite en OCL composée par un ensemble de clauses définissant les conditions d'acquisition ou de perte d'un rôle, ainsi que les contraintes liées à la migration vers une autre classe de rôles.
- **isSatisfied()** : fonction qui vérifie le respect de l'attribut condition, et qui retourne les deux valeurs true ou false.

# 5. Travaux connexes

Nous ne traitons pas dans cette section des travaux, très nombreux, concernant les rôles et les concepts similaires. Nous les avons largement discutés dans (Dahchour et al., 2004). La plupart des travaux visant à étendre UML par de nouveaux concepts recourent aux mécanismes des stéréotypes (Gogolla et al., 2002). A titre d'exemple, cette approche a été utilisée par (Luján-Mora et al., 2002) pour étendre UML par les concepts relatifs à la modélisation multidimensionnelle.

Notons que les stéréotypes ne permettent pas de changer les caractéristiques fondamentales des éléments auxquels ils sont appliqués. A titre d'exemple, si nous appliquons un stéréotype à une classe, l'élément résultant reste toujours une classe, avec la sémantique et tous les aspects structuraux attribués aux classes. On peut, néanmoins, ajouter de nouvelles restrictions aux éléments du modèle. Par exemple, nous pourrions créer un stéréotype applicable à une classe et définir une restriction ne permettant pas l'instantiation des classes auxquelles le stéréotype est appliqué. Dans ce cas-ci, l'élément résultant serait une classe abstraite.

Ainsi les stéréotypes ne devraient être utilisés que pour étendre UML par des concepts dont la sémantique est légèrement différente des éléments prédéfinis dans le métamodèle d'UML. Pour les concepts qui sont substantiellement différents de ces derniers, il y a de plus en plus de travaux qui, au lieu d'utiliser les stéréotypes, proposent d'étendre le métamodèle même d'UML. C'est le cas par exemple de (Hachani et al., 2005) qui étend UML par les concepts relatifs à la programmation par aspects, (Filho et al., 2002) qui étend UML par un modèle de propriétés (Features Model) ou encore (Clarke et al., 2002) qui étend UML par un modèle de composition. Notre approche pour intégrer l'association role-of dans UML s'inspire de ces derniers travaux.

Nous discutons dans la suite les travaux, très limités, qui traitent de l'intégration des rôles dans UML.

Nous nous accordons avec (Depke et al., 2000) sur le fait que les rôles associés aux diagrammes de collaboration d'UML ne sont pas appropriés pour capturer la sémantique sous-jacente aux rôles de l'association role-of ainsi que sur les limites des stéréotypes à intégrer ces derniers dans UML. Nous divergeons sur les points suivants.

D'abord, les modèles de rôles respectifs que nous proposons d'intégrer dans UML ne sont pas les mêmes. Le travail de (Depke et al., 2000) se base essentiellement sur les travaux (Gottlob et al., 1996) et (Kristensen, 1996) que nous avons déjà discuté dans (Dahchour et al., 2004). Par conséquent, les métamodèles correspondant ne sont pas les mêmes. A titre d'exemple, dans le métamodèle de la Figure 6 nous définissons les notions de lossPredicate, aquisPredicate et tansitionPredicate relatives aux contraintes d'abandon et d'acquisition des rôles totalement absentes dans le métamodèle des rôles de (Depke et al., 2000).

Par ailleurs, (Depke et al., 2000) définit la métaclasse Role-of relationship comme sous classe de la métaclasse Association et considère ainsi role-of comme un cas particulier de l'association. Cela crée une ambiguïté inutile similaire à celle induite en considérant l'agrégation/composition comme un cas particulier de l'association. Cette approche a été largement critiquée dans la littérature (voir par exemple (Bruel et al., 2001)). Notre métamodèle des rôles définit la métaclasse Role-of comme sous classe de la métaclasse Relationship et considère ainsi role-of comme association générique à part entière au même titre que la généralisation et l'agrégation telle qu'elle a été redéfinie dans (Bruel et al., 2001).

(Steimann, 2000(a)) critique sévèrement le concept des rôles adopté par UML et suggère de le remplacer par son propre modèle de rôles (Steimann, 2000(b)) en agissant sur le métamodèle d'UML. Néanmoins, (Steimann, 2000(a)) donne juste la syntaxe abstraite de son métamodèle de rôles. Il n'aborde pas les deux autres parties du formalisme de spécification

d'UML. Ainsi, aucune règle OCL n'a été définie pour exprimer la sémantique des rôles. Le même auteur discute dans (Steimann, 2001) une version de rôles qui peut être confondue avec la notion d'interfaces d'UML. Par ailleurs, l'auteur propose dans (Steimann et al., 2002) un noyau de langage de modélisation objet beaucoup plus réduit que la version actuelle d'UML. Dans la liste des concepts de base constituant ce noyau figure le concept de type qui peut être un type naturel appelé classe ou type de rôle appelé rôle. Ce dernier représente la version des rôles proposée dans (Steimann, 2001).

(Guizzardi et al., 2004), de son côté, propose un profile ontologique bien fondé autour des classeurs[2] d'UML. Il propose notamment un ensemble de stéréotypes (kind, phase, role, mixin) de la classe Class permettant une analyse très fine des différents types de classeurs. Dans le même ordre d'idée (Li, 2005) propose dans son projet de thèse de revoir complètement les fondements ontologiques des constructeurs d'UML notamment le concept du rôle qu'il propose de redéfinir à la lumière des nombreux travaux traitant de ce concept. L'objectif visé étant de proposer une version d'UML plus appropriée pour la modélisation conceptuelle.

# 6. Conclusion

Dans cet article, nous avons présenté l'intégration d'un modèle de rôles dans le langage UML. Tout d'abord, nous avons présenté l'association role-of sur laquelle se base ce travail. Ensuite, nous avons présenté la structure et l'architecture du langage UML notamment les paquetages de base qui sont affectés par l'intégration de notre association role-of. Enfin, nous avons présenté notre approche pour étendre UML par le concept de rôles en suivant la même démarche utilisée par UML lui-même pour définir ses propres méta-modèles. Cette extension permettra d'utiliser l'association role-of dans la phase de modélisation au même titre que les associations génériques déjà offertes par UML.

Bézivin J. (2004), "Sur les principes de base de l'ingénierie des modèles", *RSTI-L'objet*, Octobre, p. 145-156.

Bruel J.M, Henderson-Sellers B., Barbier F., Le Parc A., France R.B.(2001), "Improving the UML Metamodel to Rigorously Specify Aggregation and Composition", *Proc. of the 7th Int. Conf. on Object Oriented Information Systems*, OOIS'01, 27-29 August, Calgary, Canada, Springer, p. 5-14.

Chu W.W., Zhang G. (1997), "Associations and roles in object-oriented modeling", *Proc. of the 16th Int. Conf. on Conceptual Modeling, ER'97*, Los Angeles, California, LNCS,Vol.1331, Springer, Berlin, p. 257-270.

Clarke S. (2002), "Extending standard UML with model composition semantics", *Science of Computer Programming*, 44 (1), p. 71-100.

ClarkT., Warmer J.B. (2002), "Object Modeling With the OCL: The Rationale Behind the Object Constraint Language", LNCS, Vol. 2263, Springer, Berlin.

Dahchour M., Pirotte A., Zimányi E. (2002), "A Generic Role Model for Dynamic Objects", *Proceedings of the 14th Int. Conf. on Advanced Information Systems Engineering, CAiSE 2002*, Toronto, Canada, May 27-31, p. 643-658.

Dahchour M., Pirotte A., Zimányi E. (2004), "A role model and its metaclass implementation", *Information Systems Journal*, volume 29, p. 235-270, Elsevier.

Dahchour M., Pirotte A., Zimányi E. (2005), "Generic relationships in information modeling". *Journal on Data Semantics IV, p. 1-34*, Springer-Verlag

Dahchour M., Rayd H., Lakhrissi Y., Kriouile A. (2006), "Extension d'UML par les rôles", *Proc. of the 9th Maghrebian Conference on Information Technologies (MCSEAI 2006)*, 7-9 December, Agadir, Morocco.

[2] Le terme « classeur » est la traduction en français du terme anglais « classifier » de la terminologie UML. C'est un mécanisme qui décrit les caractéristiques comportementales et structurelles. Les classeurs incluent les interfaces, les classes, les types de données et les composants.

Depke R., Engels G., Küster J. M. (2000), "On the Integration of Roles in the UML." *Technical Report No. 214*, University of Paderborn, August.
Filho I. M.,, de Oliveira T. C., Pereira de Lucena C. J. (2002), "A Proposal for the Incorporation of the Features Model into the UML Language", *Proc. of the 4st Int. Conf. on Enterprise Information Systems, ICEIS'02*, Ciudad Real, Spain, April 2-6, p. 594-601.
Gogolla M., Henderson-Sellers B. (2002), "Analysis of UML Stereotypes within the UML Metamodel." *Proc. of the 5th Conf. Unified Modeling Language, UML'02*, Dresden, Germany, September 30 - October 4, p. 84-99.
Gottlob G., Schrefl M., Röck B. (1996), "Extending object-oriented systems with roles", *ACM Trans. Office Information Systems*, 14 (3), p. 268-296.
Guizzardi G., Wagner G., Guarino N., van Sinderen M. (2004), "An Ontologically Well-Founded Profile for UML Conceptual Models", *Proc. of the 16th Int. Conf. on Advanced Information Systems Engineering, CAiSE 2004*, Riga, Latvia, June 7-11, p. 112-126.
Hachani O., Bardou D. (2005), "Modélisation par aspects et transformation vers AspectJ et Hyper/J", *RSTI-L'objet*, LMO'O5, p. 127-142.
Kent W. (1991), "A rigorous model of object reference, identity, and existence", *Journal of Object-Oriented Programming*, 4 (3), p. 28-36.
Kristensen B. B. (1996), "Object-oriented modeling with roles", *Proc. of the Int. Conf. on Object Oriented Information Systems, OOIS'95*, Dublin, Ireland, Springer, Berlin, p.57-71.
Li X. (2005), "Using UML in Conceptual Modeling: Towards an Ontological Core", *Proceedings of the 12th Doctoral Consortium, CAiSE'05*, June, Porto, Portugal.
Luján-Mora S., Trujillo J., Song I.Y. (2002), "Extending the UML for Multidimensional Modeling", *Proc. of the 5th Int. Conf. on the Unified Modeling Language, UML'02*, Dresden, Germany, p. 290-304.
John Mylopoulos (1998), "Information Modeling in the Time of the Revolution", *Information Systems Journal*, 23(3-4), p.127-155.
OMG (2004 a), "Unified Modeling Language", v1.5, March, www.omg.org.
OMG (2004 b), "UML 2.0 OCL Specification", April, www.omg.org.
OMG (2004 c), "Meta-Object Facility (MOF)", www.omg.org.
Reenskaug T. (1996), Wold P., Lehene O. A.. "Working with Objects—The OORAM Software Engineering Method", Addison-Wesley/Manning.
Steimann F. (2000), "A radical revision of UML's role concept", *Proc. of the 3rd Int. Conf. on the Unified Modeling Language, UML'00*, Springer, p. 194–209.
Steimann F. (2000), "On the representation of roles in object-oriented and conceptual modelling", *Data & Knowledge Engineering*, vol. 35(1), p. 83–106.
Steimann F. (2001), "Role = Interface: a merger of concepts", *Journal of Object-Oriented Programming*, vol. 14(4), p. 23–32.
Steimann F., Kühne T. (2002), "A radical reduction of UML's core semantics", *Proc. of the 5th Int. Conf. on the Unified Modeling Language, UML'02*, Springer, p. 34–48.
Wieringa R.J., de Jonge W. (1991), "The identification of objects and roles: object identifiers revisited", *Technical Report IR-267*, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, December.
Wieringa R.J., De Jonge W., Spruit P. (1995), "Using dynamic classes and role classes to model object migration", *Theory Practice Object Systems*, vol. 1 (1), p. 61-83.
Wong R.K., Chau H.L., Lochovsky F.H. (1997), "A data model and semantics of objects with dynamic roles", *Proc. of the 13th Int. Conf. on Data Engineering, ICDE'97*, Birmingham, UK, IEEE Computer Society, p. 402-411.