Date de dernière modification *: 30/6/2015*

Annexe **17**

Les diagrammes de classes UML

A17.1MULTIPLICITÉ MINIMALE D'UN RÔLE

<complément de la section 17.6.2>

La multiplicité [1..*] du rôle exporte.Pays a des effets qui malheureusement ne se limitent pas à cette interprétation. Observons d'abord que la multiplicité minimum non nulle signifie que tout couple (r, p) de Produit x Usine apparaît dans au moins une association exporte, ou encore, selon la notation algébrique du chapitre 3 :

exporte[Produit, Usine] = Produit x Usine

Ce fait entraîne des contraintes dérivées complexes dont nous décrivons quelques-unes.

- Si l'une des classes opposées, Produit ou Usine, est vide, alors les deux autres classes de l'association ne sont soumises à **aucune contrainte**.
- En revanche, si, comme ce sera généralement le cas, les deux classes opposées Produit et Usine sont non vides, alors toute instance de chacune de ces deux classes doit apparaître dans une instance de l'association. En d'autres termes, chacun des deux rôles opposés est obligatoire (au sens du modèle Entité-association) pour sa classe. Les rôles exporte.Produit et exporte.Usine sont donc **conditionnellement obligatoires**, ce qui constitue un concept nouveau et complexe : une instance d'une classe peut devenir illégale lorsqu'une instance est créé dans une autre classe.
- Si **deux** des rôles d'une association n-aire ont une multiplicité dont min = 1, alors **tous les rôles** de l'association **sont obligatoires** dès que n-1 des classes participantes sont non vides. Toutes les classes participantes sont simultanément soit

non vides soit vides. Pour créer une instance d'une classe participante, il faut simultanément créer une instance de chacune des autres classes participantes et une instance de l'association.

- Si les classes Produit et Usine sont non vides, alors la classe Pays est aussi non vide. Si Usine est vide, alors Pays ou Produit (ou les deux) doivent être vides.
- Chaque instance de Produit est associé via exporte, non seulement à au moins une instance de Usine, mais à **toutes les instances** de Usine. En d'autres termes, **chaque usine exporte tous les produits** et **chaque produit est exporté par toutes les usines**. Ceci constitue une contrainte extrêmement forte.

De telles dépendances entre les classes participantes vont compliquer considérablement la gestion des données. D'une part, lors de la création d'un objet Usine, si Produit est non vide, il faut qu'il existe au préalable au moins un objet Pays ; de même pour la création d'un objet Produit lorsque Usine est non vide. D'autre part, la création d'un objet Usine (Produit) doit s'accompagner de la création simultanée d'au moins autant d'instances de exporte qu'il y a d'objets Produit (Usine). Ainsi, si la base de données contient 500 objets Usine, l'introduction d'un nouvel objet Produit entraînera l'insertion de 500 instances de exporte. Les opérations de modification et de suppression d'objets sont aussi soumises à des contraintes complexes dont la joie de la découverte est laissée au lecteur.

On peut légitimement douter que tous les analystes qui spécifient une multiplicité min > 0 sont conscients de ces dépendances parasites et de leurs conséquences. Il sera sans doute fréquent que ces analystes interprètent la multiplicité minimale comme une propriété du rôle auquel elle est attachée (comme dans les modèles Entité-association européens). Dans notre exemple, l'analyste sera tenté de lire dans le schéma que « *toute usine exporte au moins un fois* ». Non seulement cette interprétation est fausse, mais en outre elle n'est pas exprimable en UML via une association n-aire. Un tel système de contraintes ne peut être pris en compte que par une programmation transactionnelle complexe dont on peut raisonnablement douter qu'elle soit effectivement mise en œuvre.

Le standard 1.3 reconnaît prudemment que : « *Multiplicity for N-ary associations may be specified, but it is less obvious than binary multiplicity* »[1]. On notera que dans ce cas, les multiplicités ne sont pas considérées comme obligatoires. En UML 2, la recommandation est plus précise mais pas plus courageuse : « *For N-ary associations, the lower multiplicity of an end is typically 0* »[2]. Plusieurs modèles nord-américains expriment la participation obligatoire des instances d'une classe à une instance d'association par un symbole spécifique associé au rôle obligatoire dans les notations du type *crow-feet* par exemple.

Il est intéressant de noter que la plupart des auteurs ignorent les associations n-aires [D'Souza, 1998]. Certains de ceux qui les prennent en compte suggèrent de

1. La multiplicité des associations n-aires peut être spécifiée mais est moins évidente que la multiplicité des associations binaires.

2. Pour les associations N-aires, la multiplicité minimale d'une extrémité [= rôle] est typiquement 0.

s'en tenir au degré 3 et d'ignorer les multiplicités, ou de les remplacer par des identifiants d'association [Blaha, 1998].

Il paraît assez clair que les associations n-aires doivent être évitées. Lorsque de telles structures s'avèrent nécessaires, on recommandera de les traduire sous la forme d'une classe et de n associations binaires, comme illustré à la figure A17.1. On observe que cette forme permet d'exprimer simplement les rôles obligatoires et facultatifs, les cardinalités à l'européenne (par exemple, un produit fait l'objet de 1 à 20 exportations) et que les multiplicités max = 1 sont exprimées sous la forme d'un identifiant, concept nouveau mais indispensable sur lequel nous reviendrons[3].

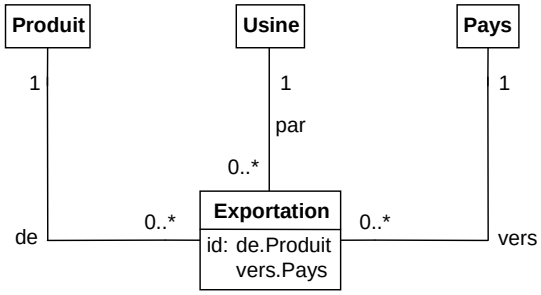


Figure A17.1 - Substitut recommandé d'une association ternaire en DB-UML

On trouvera dans les références [Génova,2002] et [Mc Allister,1998] des études plus approfondies du concept de multiplicité.

Remarque

On observe donc que l'interprétation de la valeur min provoque chez la plupart des auteurs qui adoptent les associations n-aires un malaise évident. Ajoutons aux problèmes logiques décrits ci-avant les interprétations très personnelles de certains auteurs. Citons en deux, trouvées dans des ouvrages largement répandus consacrés à la conception de bases de données.

La première consiste à interpréter min selon le modèle Entité-association : min = 1 indique que le **rôle est obligatoire**, notion qu'UML ne permet malheureusement pas de représenter pour les associations n-aires.

La seconde est plus surprenante car elle viole la définition mathématique de la relation au sens du chapitre 3[4] : s'il existe dans exporte un couple (Produit,Usine), alors ce couple est obligatoirement associé à un objet Pays. À suivre cette interprétation, il existerait donc aussi des instances de exporte dans lesquelles Usine ou Produit seraient inconnus, puisque pour ces deux rôles, min = 0 ! Rappelons que le caractère obligatoire d'un rôle représente une propriété du type d'entités participant et non une contrainte du type d'associations.

3. Cette forme ne permet pas d'exprimer la multiplicité min = 1, mais nous avons vu que cette faiblesse est plutôt un avantage.

4. Une relation est un ensemble de n-uplets, chacun composés de n valeurs tirées chacune d'un domaine. Sachant en outre que la valeur null ne fait partie d'aucun domaine.

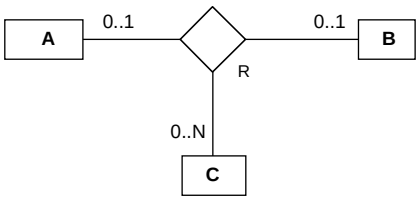
Références

GÉNOVA ET AL., The meaning of multiplicity of N-ary assiciations in UML, Software Systems Modeling (2002), 1:86-97, Special issue UML 2001

MC ALLISTER, A., Complete rules for N-ary relationship cardinality constraints, DKE 27(3), pp. 255-288, 1998

A17.2EXERCICES DU CHAPITRE 17

A17.1 Le schéma ci-dessous est-il normalisé, au sens du chapitre 3 ?



Solution

En admettant que la sémantique interne de R soit la relation R(A,B,C), les multiplicités [0..1], qui chacune définissent un identifiant de R, s'expriment sous la forme de deux dépendances fonctionnelles :

$AB \longrightarrow C$
$AC \longrightarrow B$

Bien que formant un circuit, ces dépendances ne sont pas anormales. La relation R, et donc aussi l'association UML R, sont normalisées.

A17.2 Exprimer en OCL les contraintes manquantes dans les schémas 17.11 et 17.12.

A17.3 Proposer un méta-schéma des diagrammes de classes de DB-UML.

A17.4 Proposer un méta-schéma des diagrammes de classes d'UML.