UML ou ERA : quel modèle pour l'analyse de l'information ?

Résumé

UML est de plus en plus souvent proposé comme *le* modèle universel de spécification de systèmes d'information, destiné à remplacer, selon ses promoteurs, les approches dites *classiques*. Dans le domaine de la modélisation des bases de données, majoritairement basée sur l'une ou l'autre variante du modèle Entité-association (EA), le modèle de classes d'UML est parfois utilisé comme un formalisme d'expression de schémas conceptuels, voire même des schémas logiques ou physiques de bases de données. L'objectif de cet article est de présenter et d'analyser les principaux concepts du modèle de classes d'UML. Nous les comparerons aux concepts similaires du modèle EA, et nous tenterons de dériver du modèle UML une version qui satisfait aux exigences de la modélisation conceptuelle des données.

1. Qu'est-ce qu'UML ?

Unified **M**odeling **L**anguage, ou UML, est un ensemble de **notations** permettant de représenter divers aspects d'une application orientée-objet (dite le *système*). Par extension, ces notations peuvent décrire certaines applications non OO et même certains aspects d'un domaine d'application.

UML est un langage, et plus précisément une notation, et non une méthode. Plusieurs méthodes peuvent être développées sur la base d'UML. Les textes de base insistent sur les aspects syntaxiques plus que sur la signification des symboles.

1.1 Origine et caractéristiques

UML est né de la fusion des concepts de trois approches : OMT [Rumbaugh, 1991], OOSE/Objectory [Jacobson, 1992] et OOAD [Booch, 1994]. Il s'agit d'une démarche essentiellement commerciale (via la société *Rational Software*, commercialisant l'atelier *Rational Rose*) et non scientifique, contrairement au développement du modèle EA.

La sémantique d'UML est *intuitive* et *informelle*, ce qui présente un avantage, celui de permettre une grande liberté d'interprétation. L'inconvénient majeur est qu'il n'existe pas de consensus ni standardisation évidents quant à l'interprétation de chacun des concepts définis. Certains diront qu'UML n'a pas de sémantique, et donc n'est pas un modèle au sens strict. En pratique, la sémantique la plus fiable est celle de l'outil qu'on utilise. Dans certains cas, on ne peut découvrir l'interprétation d'une construction d'un schéma qu'à partir du code généré par l'outil. Il faut remarquer que de nombreuses recherches sont actuellement dédiées à la définition d'une sémantique formelle pour certaines notations. En caricaturant, on pourrait dire qu'*UML propose un langage dont il reste à définir ce qu'il exprime.*

1.2 Les vues et *modèles* d'UML

Les spécifications UML proposent 9 types de diagrammes (ou *modèles*) regroupés en 5 vues. Chaque vue et chaque modèle apporte un *éclairage* particulier sur le système ou sur une phase de la construction du système.

Vue utilisateur	diagramme de cas d'utilisation (*use case*)
Vue structurelle	**diagramme de classes** diagramme d'objets
Vue comportementale	diagramme de séquence diagramme de collaboration diagramme d'états (*statechart*) diagramme d'activités
Vue d'implémentation	diagramme de composants
Vue environnementale	diagramme de déploiement

Figure 1 - Les modèles (ou types de diagrammes) d'UML

Le présent exposé est limité au **sous-ensemble** du **modèle de classes** (diagrammes de classes) relatif à la **modélisation conceptuelle des bases de données**.

1.3 Mécanismes d'extensibilité

Avant même de décrire les concepts de base du modèle de classes d'UML, nous dirons quelques mots sur les mécanismes qui permettent d'assouplir quelque peu la rigidité du modèle. En effet, comme dans la plupart des modèles offerts aujourd'hui par les outils CASE, UML propose un ensemble figé de concepts et de règles d'assemblage de ceux-ci. Certains domaines de modélisation pourraient réclamer d'autres concepts, mieux à même de décrire les situations qui leur sont propres. UML propose deux concepts génériques qui, associés à un objet d'un schéma peuvent compléter le modèle lorsque celui-ci apparaît insuffisant. Manifestement conçus à

destination des développeurs d'outils CASE en raison de leur faible coût de mise en oeuvre, ils permettent de particulariser un modèle à peu de frais. Leur généricité implique qu'ils ont une sémantique *user-dependent*, et non prise en charge par UML ni par les outils CASE (sauf programmation spécifique). Nous décrirons brièvement les tags et les stéréotypes.

Un **tag** définit une propriété de l'objet auquel il est affecté. La signification de cette propriété n'est pas précisée dans le modèle, mais dépend de l'utilisateur du modèle modifié. Elle n'est donc pas standardisée. Sa forme générale est : keyword = value, ou simplement keyword. Par exemple, si l'attribut numClient est une *primary key*, on préfixera son nom du mot clé **PK**, arbitrairement choisi : " **PK** numClient "

Un **stéréotype** définit une catégorie remarquable dans une classe ou tout autre objet. Ici encore, l'interprétation est laissée à l'utilisateur du modèle. Les conventions UML suggèrent d'associer le nom du stéréotype à celui de l'objet. Par exemple, si la classe i_compte est du type *virtuel*, on associera au nom de la classe le mot clé **virtuel** comme suit : " **«virtuel»** i_Compte "

2. Le modèle de classes d'UML

Les diagrammes de classes d'UML, aussi appelés diagrammes structurels statiques, constituent le mode privilégié de description des structures de bases de données. Nous le désignerons sous le terme de *modèle de classes* d'UML. Un diagramme de classes a pour objectif de décrire les structures statiques d'un système sous la forme de *classes*, d'*associations*, d'*attributs* et d'*opérations* (Figure 2).

Le modèle de classes d'UML est dérivé du modèle de classes d'OMT [Rumbaugh 1991], qui lui-même est une variante du modèle Entité-association de P. Chen [Chen, 1976]. Certaines constructions pourtant essentielles n'ont pas été reprises, tels les identifiants. La raison est historique : OMT étant prioritairement destiné à permettre une description abstraire d'applications orientées-objet, seules les constructions directement exprimables en C++ ou SmallTalk ont été adoptées. L'intérêt pour la modélisation des structures de données persistantes (les bases de données) n'est venu que plus tard, à un moment où les conventions graphiques étaient largement fixées. UML a repris, en les modifiant légèrement, les bases du modèle de classes d'OMT.

Tout naturellement, le modèle de classes d'UML a été enrichi de diverses constructions permettant de représenter des aspects propres aux structures de données des langages OO (C++ et Java). Issu de la communauté *programmation OO* plus que de celle des *bases de données*, il présente donc des lacunes et des limitations dans son application à la modélisation conceptuelle, mais aussi à d'autres processus d'ingénierie (il est peu applicable à la rétro-ingénierie par exemple). Le modèle de classes inclut également des concepts qui sont sans utilité en modélisation des bases de données, et que nous ignorerons dans ce chapitre.

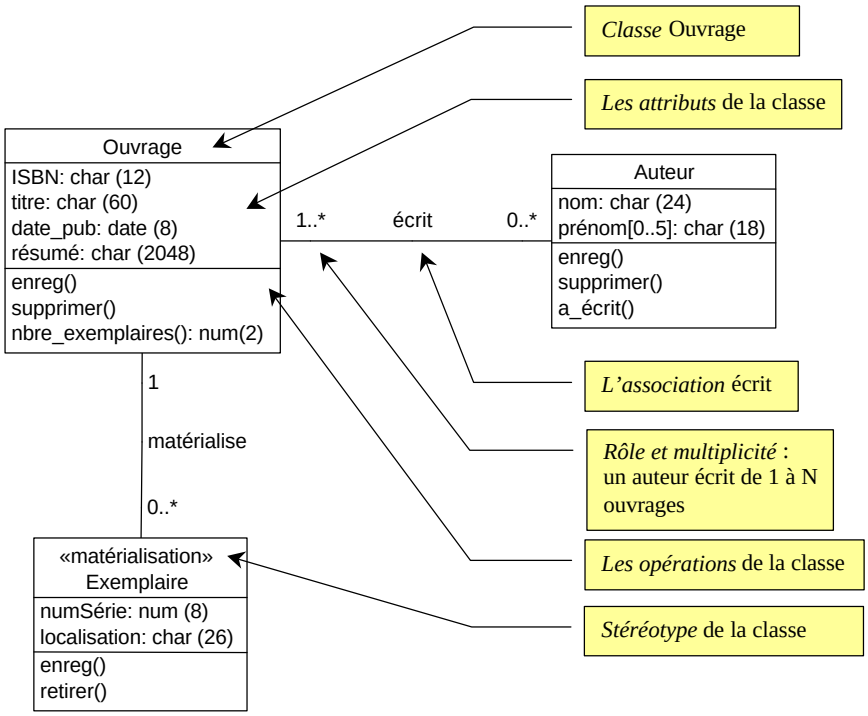

Figure 2 - Un schéma UML typique

Notons cependant une approche intéressante proposée par UML, les **profils** (ou *profiles* en anglais). Un profil définit un sous-modèle comme un sous-ensemble de concepts, augmenté de tags et stéréotypes, destiné à représenter un domaine de modélisation spécialisé. Par exemple, *UML Data Modeling Profile* [UML 2000], permet de représenter des schémas relationnels (encore immature pour l'instant).

Dans la suite de cet article, nous décrirons les concepts de *classe*, d'*association*, d'*attribut*, d'*opération* et leurs variantes Suite aux critiques que nous serons amenés à formuler, nous proposerons une variante d'UML, que nous appellerons **DB-UML**, constituée d'un sous-ensemble du modèle de classes d'UML, auquel nous ajouterons quelques extensions indispensables en modélisation conceptuelle.

3. Les classes d'objets

3.1 Principes

Une classe représente un concept ou un ensemble d'objets similaires du système modélisé. On la note sous la forme d'une boite décomposée en trois compartiments, spécifiant respectivement, le nom et le stéréotype de la classe, la liste des attributs de

la classe, les opérations applicables à la classe ou à ses instances (Figure 2). Il est théoriquement permis d'ajouter des compartiments additionnels, dits *user-defined*, mais dont on ne trouve pas d'exemples dans la littérature. Un compartiment peut inclure un titre.

On observe que les propositions de base ne suggèrent pas de compartiment *contrainte d'intégrité*. En particulier, il n'existe pas de notion d'*identifiant*, sauf via certaines associations qualifiées, comme nous le verrons dans la suite.

L'interprétation du concept de classe reste peu précise, notamment parce que le modèle n'est pas lié à un niveau d'abstraction défini. Une classe UML peut tout aussi bien modéliser un ensemble d'objets du domaine d'application qu'une classe C++. A ce titre, le modèle de classes peut être dit *à large spectre*. C'est au concepteur de la méthode (*method engineer*) utilisée par les analystes de préciser l'interprétation du concept de classe.

Dans le contexte de la modélisation conceptuelle, on décide, conformément à l'héritage historique du modèle, qu'une classe UML correspond à un **type d'entités** du modèle EA[1].

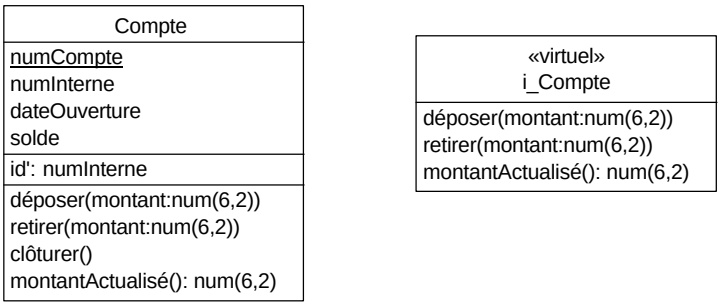

Figure 3 - Un schéma UML+ constitué de deux classes

3.2 Extension DB-UML

Afin d'adapter le modèle de classes d'UML à la modélisation conceptuelle nous ajoutons un nouveau compartiment, dit de **contraintes**, reprenant les contraintes applicables à la classe. En particulier, on y spécifiera les identifiants de la classe, en distinguant l'identifiant primaire (noté id) et les identifiants secondaires (notés id'). En outre, les attributs qui, à eux seuls, constituent un identifiant primaire seront soulignés[2] (Figure 3).

1. Dans cet article, nous utiliserons le modèle GER (generic Entity-relationship), modèle Entité-association relativement riche, qui inclut la plupart des concepts des modèles courants. L'essentiel du modèle GER a été implémenté dans l'atelier DB-MAIN. Tout comme UML, le GER est utilisé comme un modèle à large spectre.
2. On peut admettre qu'un tel identifiant ne soit pas spécifié dans le compartiment des contraintes. Cependant, ce raccourci ne sera pas admis dans les schémas logiques et physiques, puisque les identifiants participent à la définition des clés étrangères, et apparaissent dans la représentation graphique de celles-ci.

3.3 Hiérarchies de classes (généralisation/spécialisation)

Une classe peut être déclarée **super-classe** d'une ou plusieurs autres classes, appelées ses **sous-classes** (Figure 4). UML admet les propriétés habituelles de l'ensemble des sous-classes d'une super-classe, notées comme suit :

- {*disjoint*}, qui correspond au symbole de disjonction "**D**" du GER
- {*overlapping*}, ou non disjoint
- {*complete*}, correspondant au symbole de totalité "**T**" du GER[3]
- {*incomplete*}, ou non total.

Une partition sera notée {*disjoint,complete*}. Si on le désire, on peut indiquer le critère de distribution des instances dans les sous-classes sous la forme d'un **discriminateur**, qui est formé de l'attribut de la super-classe dont la valeur définit la sous-classe d'une instance.

Une classe est dite **abstraite** si la création de ses instances ne peut se faire que via la création d'instances de ses sous-classes. Dans le cas contraire, la classe est dite **concrète**. Ce concept est en fait lié à celui de totalité (*complete*/*incomplete*). La propriété *complete* implique que la super-classe est abstraite tandis qu'une super-classe concrète est définie par la propriété *incomplete*.

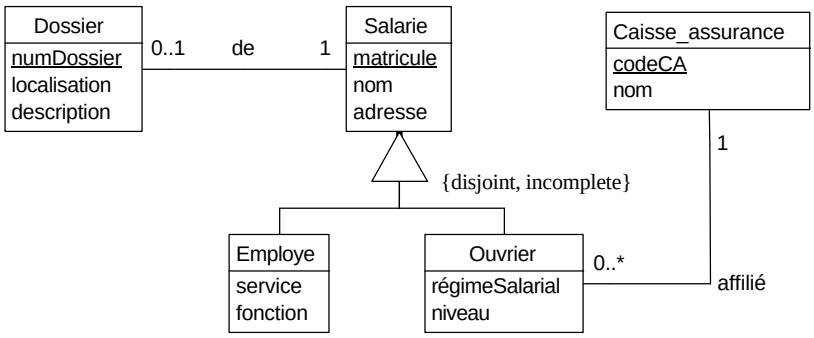

Figure 4 - Les deux classes *Employe* et *Ouvrier* forment une partition de *Salarie*.

4. Les attributs

4.1 Principes

Un *attribut* est une variable d'état associée à une classe. Il est décrit sous la forme d'une chaîne de caractères apparaissant dans le compartiment des attributs d'une classe. L'attribut UML est similaire à l'attribut du modèle EA.

3. Cette dénomination évoque le fait que la définition des sous-types n'est pas encore complète, signifiant implicitement par là que le statut normal de tout ensemble de sous-types devrait être *complete* dans un schéma achevé.

Un attribut peut être monovalué ou multivalué, obligatoire ou facultatif. Ces propriétés s'expriment par la *multiplicité* de l'attribut. Correspondant à la cardinalité d'attribut dans le modèle EA, la multiplicité exprime les valeurs autorisées du nombre d'instances (valeurs) de l'attribut pour une instance de la classe.

L'attribut est défini sur un type de base (numérique, caractères, etc.) ou sur un type défini par l'utilisateur (*user-defined data type*). Cependant, UML ne fixe pas l'ensemble des types de base possibles, ses auteurs estimant que ceux-ci sont *language-dependent*[4]. C'est donc aux développeurs d'outils CASE qu'incombe de faire le choix des types proposés aux utilisateurs.

UML associe à un attribut un certain nombre de propriétés qui ne sont pas toutes pertinentes pour la modélisation conceptuelle. On pourra en particulier spécifier le nom de l'attribut, sa visibilité (+ = *public*, # = *protected*, - = *private*), sa multiplicité (1 par défaut, sinon spécifié explicitement : [0..1], [3], [2..*]), son type (de base ou *user-defined*), sa valeur par défaut, sa (non-)modifiabilité (*frozen*), ainsi que d'autres propriétés à l'initiative de l'utilisateur (*user-defined*).

Exemples de définition : +prix : num(8) = 0; prénom[1..4] : string.

5. Les opérations

Une **opération** est un service que peut rendre toute instance d'une classe, ou la classe elle-même. Ce concept générique constitue une abstraction pour les *méthodes*, les *prédicats*, les *contraintes* ou autres *procédures* attachées à une classe.

Une opération peut être dotée d'une implémentation. Une opération d'une classe C est dite *abstraite* si ses implémentations sont fournies par les sous-classes de C.

Tout comme pour les attributs, UML associe à une opération des propriétés descriptives. On pourra en particulier spécifier le nom de l'opération, sa visibilité, la liste de ses paramètres (direction {in/out/inout}, nom, type, valeur par défaut), le type du résultat et sa valeur par défaut si l'opération est une fonction, ainsi que d'autres propriétés à l'initiative de l'utilisateur (*user-defined*).

Exemple de définition : calcul_prix(in taux_TVA : num(1,3) = 0.215) : num(8).

6. Les associations binaires

UML distingue les associations binaires des associations N-aires, celles-ci apparaissant comme des extensions des premières. Conformément à cette approche, nous les présenterons séparément.

4. Ce qui malheureusement ne contribue pas à la standardisation du modèle. En outre, ce type de commentaire illustre l'une des motivations principales des auteurs : proposer un modèle abstrait pour les langages de programmation OO.

6.1 Principes

Une association binaire représente une relation entre les populations de deux classes ou entre celle d'une classe et elle-même (Figure 5). Elle correspond au type d'associations binaire du modèle EA. Son nom n'est pas obligatoire.

Chaque extrémité est appelée **rôle**, qui peut être nommé Une association binaire possède deux rôles. Une association définie entre une classe et elle-même est dite *cyclique* (erronément dénommée *réflexive*[5]).

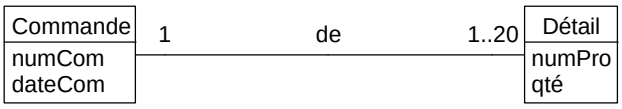

Figure 5 - Une association binaire

Chaque rôle est caractérisé par une **multiplicité**, spécifiée par l'expression d'un ensemble d'entiers non négatifs, le plus généralement sous la forme d'un intervalle [i..j] :

soit R (r1[i1..j1]:C1, r2[i2..j2]:C2) une association R de rôles r1(joué par C1) de multiplicité [i1..j1] et r2 (joué par C2) de multiplicité [i2..j2];

La **multiplicité** [i1..j1] de r1 exprime le fait qu'à partir de chaque instance de C2, *on peut voir*, au travers de R, de i1 à j1 instances de C1 (interprétation *look-across*, comme nous le verrons ci-après). Ou encore, chaque instance de C2 apparaît dans un nombre d'associations R compris entre i1 et j1.

UML propose des notations synthétiques intuitives, bien qu'irrégulières[6], pour exprimer les multiplicités d'un schéma :

2..4	:	de 2 à 4
0..1	:	de 0 à 1; au plus 1
1	:	= 1..1; de 1 à 1; exactement 1
*	:	= 0..*; de 0 à ∞; un nombre quelconque
1..*	:	de 1 à ∞; au moins 1
1,3..5	:	1, 3, 4 ou 5 (*rarement observé*).

On peut déclarer un rôle *ordonné*, signifiant par là qu'il est possible d'accéder, à partir d'un objet source (de l'autre rôle) aux objets jouant ce rôle dans un ordre déterminé. Bien que le concept d'ordre soit surtout pertinent pour des schémas logiques ou physiques, il peut à l'occasion s'avérer utile dans un schéma conceptuel.

5. Une relation R(A,A) est réflexive si, pour tout a ∈ A, (a,a) ∈ R.
6. L'expression "1" représente 1..1 mais l'expression "*" signifie 0..*.

6.2 Multiplicité UML et cardinalité EA

Il est important de noter que la multiplicité d'un rôle d'une association binaire UML n'est pas équivalente à la cardinalité d'un rôle d'un type d'associations EA, du moins dans les modèles qui, comme Merise et le GER adoptent la sémantique européenne.

soit R(r1[i1..j1]:C1, r2[i2..j2]:C2) un type d'association R de rôles r1(joué par C1) de cardinalité [i1..j1] et r2 (joué par C2) de cardinalité [i2..j2];

La **cardinalité** [i1..j1] du rôle r1 exprime le fait que chaque instance de C1 *apparaît* dans un nombre d'associations R compris entre i1 et j1 (interprétation *participation*).

Les interprétations *look-across* et *participation* s'opposent subtilement, puisque dans le cas des associations binaires, elle se présentent comme inversées l'une par rapport à l'autre, comme le montre la Figure 6.

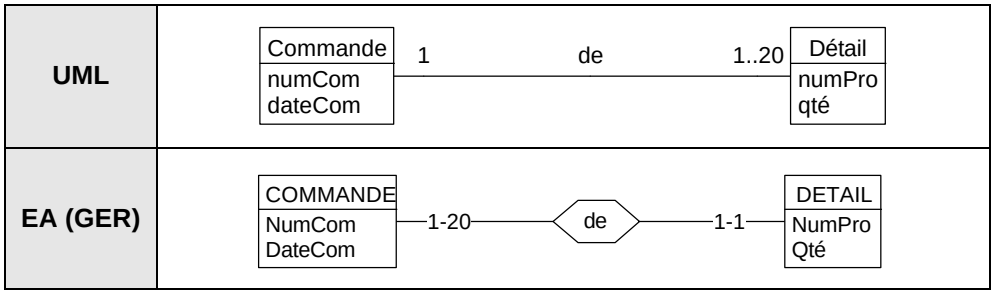

Figure 6 - Les multiplicités des rôles dans un schéma UML et les cardinalités des mêmes rôles dans le modèle EA sont inversées

Remarque

Le modèle EA auquel on fait référence dans cet article est une variante européenne, comme proposée par Abrial, Merise, Batini ou le GER, et non nord-américaine, comme proposée par Chen, Teorey ou OMT (dont UML a hérité).

7. Agrégation et composition

UML identifie des types particuliers d'associations qui permettent de modéliser le fait qu'*un objet est constitué d'autres objets*, structure qui est perçue comme plus forte que la simple association établie entre une classe de composés et des classes de composants. Deux variantes sont proposées : l'agrégation et la composition. La plupart des modèles EA ignorent ces formes.

7.1 Association d'agrégation

Une association binaire d'**agrégation** indique que chaque objet de la classe source est un *agrégat* d'objets de la classe cible. Il s'agit d'une sémantique intuitive peu précise qui ne permet pas toujours de distinguer une agrégation d'une association binaire

ordinaire. L'exemple de la Figure 7 illustre ce concept : *une équipe est faite d'au moins un joueur; un joueur intervient dans un nombre quelconque d'équipes*

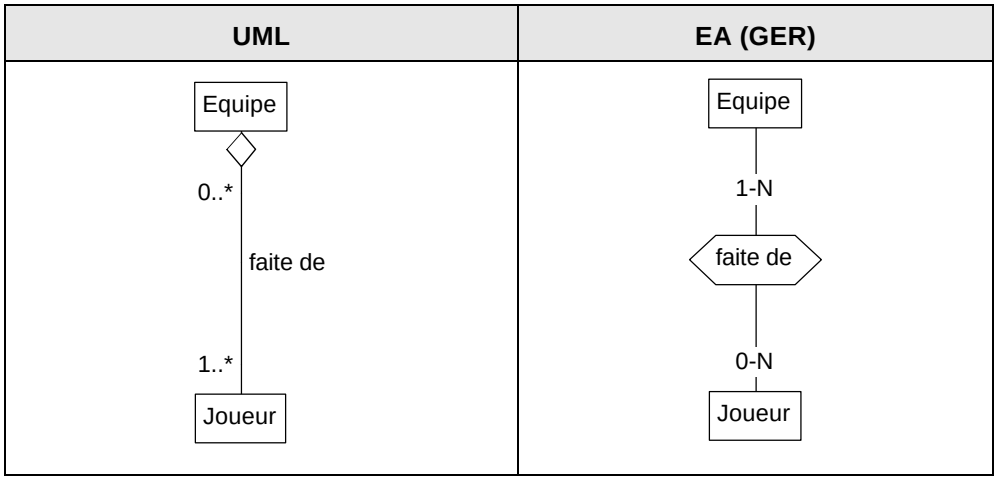

Figure 7 - L'association d'agrégation est utilisée pour indiquer qu'une équipe est faite d'un ensemble de joueurs

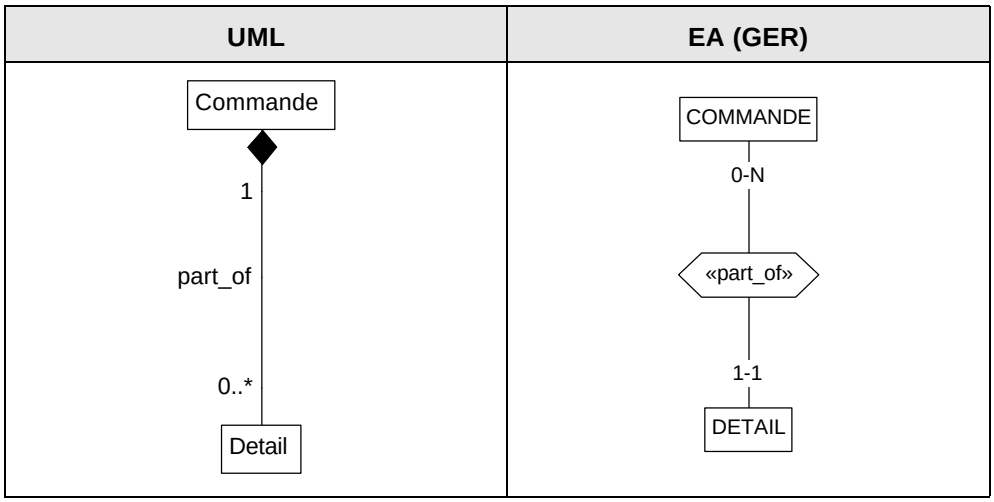

Figure 8 - L'association de composition part_of et son équivalent dans le modèle EA

7.2 Association de composition

Une association binaire de **composition** est définie comme une agrégation *forte*, l'existence d'un composant étant liée à celle de son agrégat (Figure 8). Bien que la définition soit également imprécise, le concept semble plus clair que l'agrégation, du moins à la lumière des exemples que propose la littérature. A titre d'illustration, la

figure 8 définit une *commande comme constituée d'un certain nombre de détails, lesquels n'existent que comme constituants d'une commande*.

Le modèle EA n'inclut pas de construction équivalente. Nous proposons de simuler la composition par le stéréotype part_of dans le GER (figure 8), ou par un type d'associations doté d'un nom générique (*part_of*, *composant*, etc.) dans les autres modèles EA.

Malgré l'absence de définition précise, le concept de composition s'avère très utile dans l'analyse conceptuelle pour distinguer certaines variantes d'association. L'exemple de la Figure 9 montre clairement la différence entre une composition (une commande est *composée* de détails) et une association binaire ordinaire (un client passe des commandes mais n'est *pas composé* de commandes). Or les deux structures correspondent, dans le modèle EA, à des types d'associations identiques.

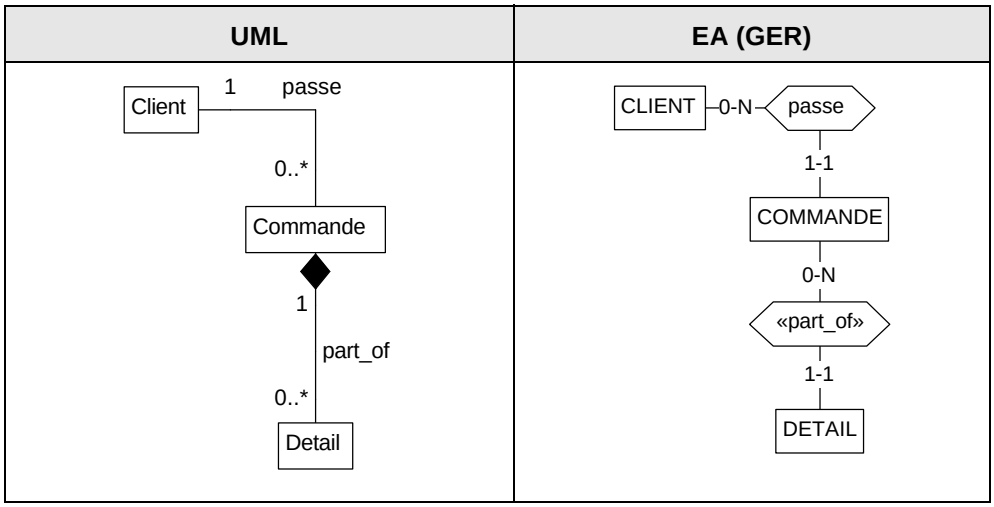

Figure 9 - Association de composition et association simple

8. Les associations qualifiées

8.1 Principes

Relativement à une association binaire entre une classe source A et une classe cible B, un **qualifieur** est une collection d'attributs de cette association dont les valeurs **partitionnent** l'ensemble des objets B associés à un même objet A via l'association. Une **association qualifiée** est une association dotée d'un qualifieur. Les attributs du qualifieur appartiennent à l'association et sont attachés au rôle source. C'est par ce mécanisme d'association qualifiée que nous pouvons modéliser les situations suivantes, toutes deux inspirées d'exemples extraits des documents de spécification UML 1.3.

1. Une banque a des clients (qui sont des personnes), mais, dans une banque, à un numéro de compte déterminé correspond un seul client (Figure 10, gauche).
2. Un échiquier comporte des cases; dans un échiquier, une colonne et une ligne désignent une case bien déterminée (Figure 10, droite).

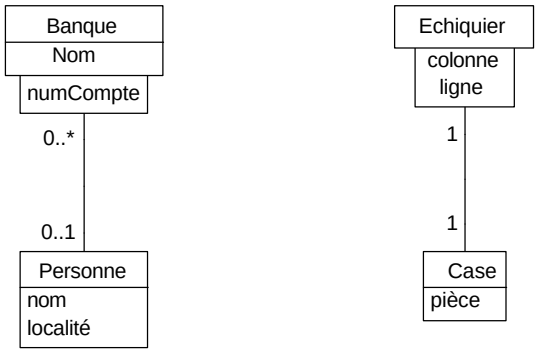

Figure 10 - Deux associations qualifiées

La multiplicité du rôle cible (Personne ou Case) de l'association qualifiée doit être interprétée d'une manière particulière : elle indique combien d'instances cibles correspondent à *1 instance source + une valeur du qualifieur.* La multiplicité du rôle source est indépendante du qualifieur.

Deux cas sont à distinguer, selon la multiplicité modifiée du rôle cible.

1. Chaque membre de la partition contient un seul élément. La multiplicité du rôle cible **dégénère** donc en [0..**1**] ou [**1**]. Il s'agit du cas le plus fréquent.
2. Chaque membre de la partition peut contenir plus d'un élément. Ce cas de figure est rarement illustré dans la littérature. On trouvera dans [Blaha, 1998] un des seuls exemples connus, malgré tout peu convaincant.

8.2 Analyse du concept

Le concept d'association qualifiée, en apparence simple et d'utilité évidente du moins dans sa première variante, pose cependant plus de problèmes qu'il n'en résout.

1. Le modèle UML propose **deux constructions et notations différentes** et incompatibles pour représenter une *association dotée d'attributs* : la *classe-association* (voir plus loin) et l'*association qualifiée*.
2. Une association *un-à-plusieurs* qualifiée qui dégénère définit une dépendance fonctionnelle (Echiquier,colonne,ligne → Case), ce qui correspond à un cas particulier d'**identifiant hybride** (identifiant comportant au moins un rôle) dans le modèle EA (GER).
3. Le qualifieur ne permet cependant pas d'exprimer d'autres cas d'identifiants hybrides pourtant très utiles : {attribut + 2 rôles}, {2 rôles}.

4. Lorsque la multiplicité du rôle source est [1] (cas le plus fréquent), les attributs du qualifieur apparaissent manifestement comme des **propriétés de la classe cible** et non de l'association : *colonne et ligne sont des caractéristiques d'une case* plus que de l'association d'appartenance d'une case à un échiquier. Cette position éloignée pose d'ailleurs le problème de la spécification des autres aspects de ces attributs : types, valeurs par défaut, stéréotypes, tags, multiplicité, etc.
5. La notation rend difficile l'intervention des attributs du qualifieur dans d'autres spécifications (contraintes par exemple).
6. On notera que tout **attribut monovalué obligatoire** de la classe cible partitionne les instances de celle-ci au même titre qu'un qualifieur. *Exemples* : l'attribut *localité* de la classe *Personne* définit naturellement une partition des personnes selon leur localité; il en serait de même de *pièce* de *Case*, qui indique le nom de la pièce qui occupe la case. Un qualifieur n'est donc pas nécessaire pour définir une partition.
7. On ne peut définir qu'**un seul qualifieur par extrémité d'une association**. Or les exemples ne manquent pas où plusieurs identifiants hybrides comportant le même rôle source seraient utiles.
8. Si un qualifieur intervient dans plus d'une association (cas de figure tout à fait plausible), il doit **apparaître plusieurs fois** dans le schéma, qui perd en lisibilité. Faut-il déclarer les propriétés de ces qualifieurs plusieurs fois ? Comment indiquer que ces qualifieurs ne forment qu'un seul attribut ?
9. Le qualifieur modifie la multiplicité initiale du rôle cible. Malheureusement, le modèle ne permet pas d'indiquer quelle était cette **multiplicité d'origine**, qui est donc perdue : il devient impossible de préciser qu'*un échiquier **a 64 cases*** ou encore que *toute personne est cliente dans **au moins une banque***.
10. Les spécifications UML [UML 1.3] précisent que le qualifieur est *un **index** pour les objets cibles.* De quelle nature est cet index : technique, conceptuelle, critère de sélection fréquemment utilisé ? D'ailleurs, pourquoi cette *indexation*, à supposer que ce concept soit pertinent, devrait-elle s'exprimer uniquement dans le cadre d'une association ? Ne pourrait-on pas indexer les *Personnes* sur *localité* et les *Cases* sur *pièce*?
11. Lorsqu'un qualifieur reprend tous les attributs d'une classe, cette classe est-elle réputée n'avoir aucun attribut ? Ce cas de figure a-t-il un sens dans un schéma conceptuel ?
12. Un qualifieur d'une association *plusieurs-à-plusieurs* peut cacher un concept important, tel que *Compte* dans l'exemple ci-dessus. Dans ce cas, un qualifieur est l'indice d'une mauvaise modélisation.

Le concept d'*association qualifiée* est complexe, limité et irrégulier. Il prête à confusion et peut induire une modélisation erronée. Il ne présente qu'un seul intérêt : celui de spécifier un cas particulier d'identifiant hybride. Il s'agit d'ailleur du seul cas d'identifiant explicitement permis en UML. En enrichissant UML du concept naturel d'identifiant, celui d'association qualifiée pourraient disparaître, pour le plus grand bénéfice d'UML et de ses utilisateurs.

8.3 Equivalence UML / EA

L'interprétation EA (GER) d'une association qualifiée peut servir de sémantique à ce concept (Figure 11). On notera que le schéma EA est plus riche, car il préserve les cardinalités d'origine, qu'on suppose respectivement [1-N] et [64-64] dans les deux exemples que nous avons utilisés.

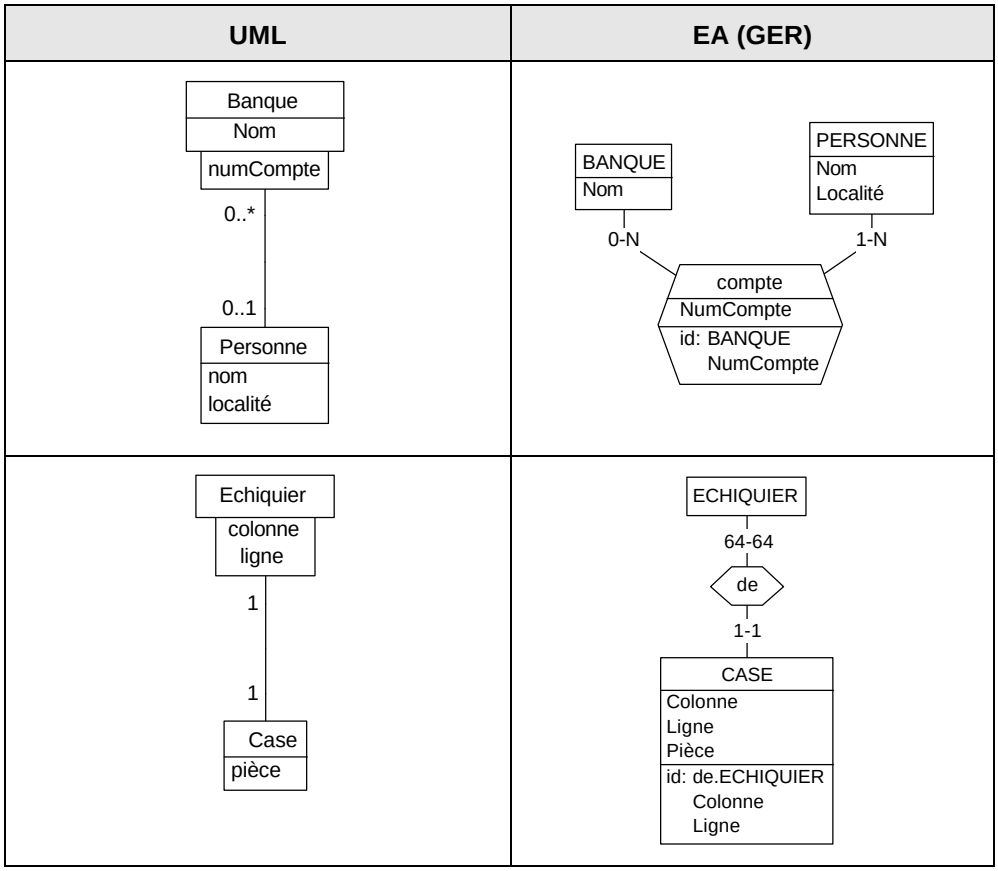

Figure 11 - Expression d'associations qualifiées dans le modèle EA (GER). Le schéma supérieur (tiré de [UML 1.3]), dans les deux modèles, mériterait d'être modifié de manière à mettre en évidence le concept de COMPTE[7].

7. L'exemple des comptes bancaires est en fait plus complexe qu'il n'y paraît. Le schéma GER laisse supposer que plusieurs associations compte peuvent exister entre une même banque et un même client (*règle* : si un identifiant explicite est spécifié pour un type d'associations, alors l'identifiant implicite, soit ici l'ensemble des rôles, doit être explicité). Cette interprétation est parfaitement réaliste, un client peut en effet posséder plusieurs comptes dans une même banque. Bien que la spécification d'UML n'en parle pas, on peut supposer qu'il n'en va pas de même dans ce modèle. Le type d'associations *compte* du schéma EA devrait alors être doté d'un deuxième identifiant, {BANQUE,PERSONNE}, indiquant qu'une personne ne peut ouvrir qu'un seul compte dans une même banque. Ceci confirme que l'exemple exprime une mauvaise modélisation.

9. Les associations N-aires

9.1 Principes

Une association N-aire correspond à une relation entre les populations de plus de deux classes, non nécessairement distinctes (Figure 12). Ce concept est présenté comme une extension de l'association binaire à plus de deux rôles, approche typique des modèles EA nord-américains (voir la présentation de Chen dans [Chen 1976] par exemple[8]).

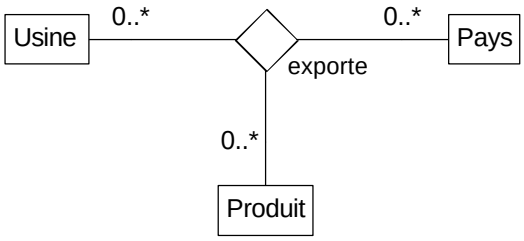

Figure 12 - Une association ternaire : *les usines exportent des produits dans des pays*

Assez curieusement, les associations N-aires sont dotées d'une notation et d'un comportement différents de ceux des associations binaires. Cette différence est cependant inutile car il est possible d'en donner une définition compatible avec celle de l'association binaire.

Une association N-aire peut porter un nom. Chaque rôle, qui peut porter un nom explicite, est caractérisé par une **multiplicité**, qui est un ensemble d'entiers non négatifs, le plus généralement un intervalle [i..j].

soit R (r1[i1..j1]:C1, r2[i2..j2]:C2, r3[i3..j3]:C3) une association ternaire de rôles r1, r2, r3 entre les classes C1, C2 et C3. La multiplicité [i1..j1] de r1 indique qu'à partir de chaque combinaison d'instances de C2 et de C3, *on peut voir*, au travers de R, de i1 à j1 instances de C1.

Si la définition paraît simple, nous verrons cependant qu'elle induit de nombreux problèmes qui la rendent impropre à l'utilisation, sauf dans des cas simples. Selon certains auteurs, la notion est adoptée sans restriction, mais est rarement utilisée[9]. Selon d'autres, on ne conserve que la borne supérieure, d'autres enfin suggérant de s'en méfier [UML 1.3], de l'ignorer [Blaha 1998], voire même d'ignorer des association N-aires. Le malaise est donc réel.

8. Dans les modèles européens (à l'exception du modèle *Data Semantics* d'Abrial, qui est purement binaire), les types d'associations binaires forment un cas particulier des types d'associations N-aires. Cette divergence est à l'origine des problèmes que posent les associations N-aires, et que nous allons étudier.
9. La plupart des ouvrages qui présentent les associations N-aires n'en contiennent qu'un seul exemple, celui qui illustre le concept.

9.2 Multiplicité UML et cardinalité EA

Nous analyserons de manière plus approfondie le concept de multiplicité dans la section suivante. Il est cependant utile de comparer ce concept avec celui de cardinalité dans le modèle EA, qui semble très voisin de la multiplicité.

En fait, les multiplicités d'une *association N-aire* (UML) et les cardinalités du type d'associations (selon les variantes européennes) équivalent sont (largement) indépendantes. Elles ne sont malheureusement pas dérivables l'une de l'autre, contrairement à ce qui se passait pour les associations binaires.

A titre d'exemple, les deux schémas de la Figure 13, décrivant le même domaine d'application, et exprimés respectivement en UML et en EA, sont, en dépit des apparences, parfaitement compatibles. Par exemple, le fait qu'une usine ne puisse exporter plus de 10 produits dans un même pays (schéma UML) n'est pas incompatible avec le fait qu'un produit ne puisse être exporté plus de 20 fois, quels que soient l'usine et le pays (schéma EA).

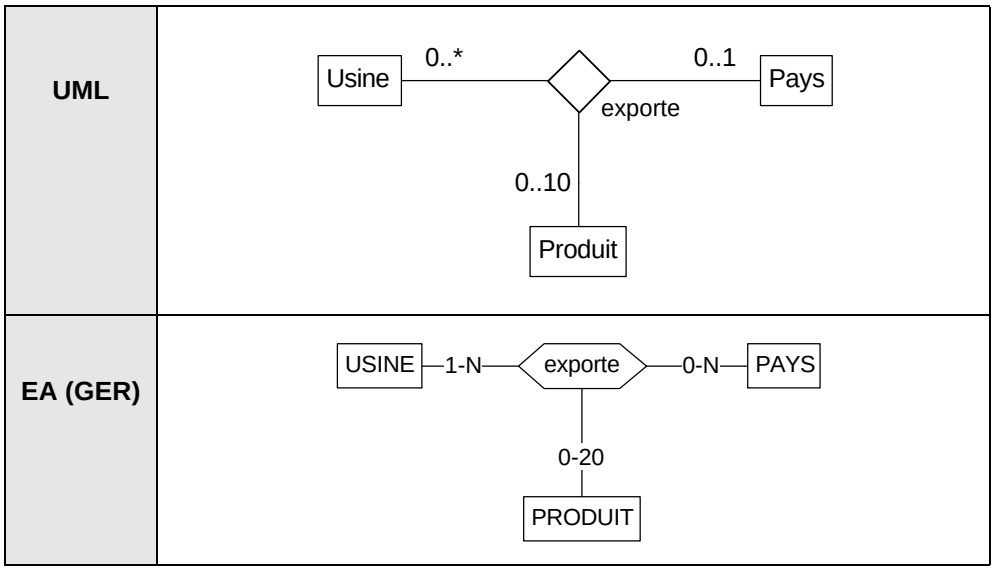

Figure 13 - Ces deux schémas, sans être équivalents, sont pourtant compatibles, malgré les apparences

10. Le concept de multiplicité

Etant donné son importance et son expressivité, le concept de multiplicité mérite une étude spécifique, ce que nous allons développer dans cette section.

10.1 Associations binaires et N-aires : une définition unique

Bien que les documents UML présentent les associations N-aires comme distinctes des associations binaires, il est possible d'en donner une définition commune.

Une association de degré N ($2 \leq N < \infty$)

$$R\ (r1[i1..j1]:C1,\ r2[i2..j2]:C2,\ ...,\ rN[iN..jN]:CN)$$

représente une relation définie sur les classes C1, C2, ..., CN, non nécessairement distinctes. Une association représente donc un ensemble de N-uplets $\{<o1,o2,...,oN> \in C1xC2x...xCN\}$.

Multiplicité : la multiplicité [ik..jk] du rôle rk de R indique les valeurs possibles du nombre d'associations R dans lesquelles toute combinaison d'instances des classes jouant les N-1 autres rôles {r1,...,rk-1,rk+1,...,rN} apparaît :

$$\forall <o1,...,ok-1,ok+1,...,oN> \in C1x...xCk-1xCk+1X...xCN:$$

$$ik \leq |\{ok \in Ck: <o1,...,ok-1,ok,ok+1,...,oN> \in R\}| \leq jk$$

Cette définition est valable pour les associations tant binaires que N-aires. Dans le premier cas, les *classes jouant les autres rôles* se réduisent à l'unique classe jouant le rôle opposé.

La *cardinalité* d'un rôle telle qu'elle est adoptée dans les modèles EA (variantes européennes telles que Merise ou GER) est différente :

Cardinalité : la cardinalité [ik..jk] du rôle rk de R indique les valeurs possibles du nombre d'associations R dans lesquelles toute entité Ck joue le rôle rk :

$$\forall ok \in Ck: ik \leq |\{r \in R: r[rk] = ok\}| \leq jk$$

L'interprétation UML est dénommée ***look-across*** car elle mesure le nombre d'instances qu'on peut *voir via* l'association, alors que l'interprétation du modèle EA (européen) est dénommée ***participation*** car elle mesure le nombre d'associations auxquelles une instance *participe*.

L'interprétation look-across est typique des modèles d'origine nord-américaine, tels que ceux de Chen [Chen 1976], Teorey [Teorey 1999], Elmasri [Elmasri 2000], OMT [Rumbaugh 1991], UML [UML 1.3][10]. Dans ces approches, les associations binaires ont été décrites et formalisées, puis ultérieurement étendues aux associations N-aires.

L'interprétation participation s'observe surtout dans les modèles mis au point en Europe : Abrial [Abrial 1974], Merise [Nancy 1996], Batini [Batini 1992], Elmasri [Elmasri 2000], Ceri [Ceri 1997], Corba [OMG 2000], Coad [Coad 1993], GER.

L'apparente origine géographique de ces interprétations est purement fortuite. Les problèmes que nous allons évoquer étaient certainement considérés comme mineurs à cette époque, si tant est qu'ils aient été perçus correctement par l'ensemble de la communauté scientique et professionnelle.

10. Egalement proposé dans la variante UML de l'atelier DB-MAIN.

Il apparaît rapidement que les bornes minimum (i) et maximum (j) d'une multiplicité définissent des contraintes indépendantes de nature différente. Ce fait a été reconnu depuis longtemps. Dans certains modèles d'ailleurs, elles s'expriment par des attributs graphiques indépendants. Dans la mesure où ils posent également des problèmes de nature différente, on les analysera séparément avant de formuler des conclusions et des recommandations générales. Cette analyse comprendra une comparaison avec le concept de cardinalité du modèle EA.

10.2 Borne maximale *j* de la multiplicité

Afin de focaliser la discussion sur les aspects essentiels, nous distinguerons trois valeurs types de la multiplicité maximale, soit: j = 1, j = * et j = m (valeur finie > 1)

- **j = 1** La multiplicité [ik..1] du rôle rk indique que les autres rôles {r1,...,rk-1,rk+1,...,rN} **constituent un identifiant de R**.

  R (a[0..*]:A, b[0..*]:B, c[0..**1**]:C) ⇔ R (**a, b**, c)

  Dans le cas d'une association binaire, cette propriété se traduit par le fait qu'une instance de la classe de l'autre rôle n'apparaît que dans une seule instance de l'association (association *un-à-plusieurs* ou *un-à-un*).

- **j = *** La multiplicité [ik..*] du rôle rk indique que les autres rôles {r1,...,rk-1,rk+1,...,rN} ne **constituent pas un identifiant de R**. Dans le cas d'une association binaire, cette propriété se traduit par le fait qu'une instance de la classe de l'autre rôle peut apparaître dans plusieurs instances de l'association.

- **j = m > 1 (m ¹ *)** Les autres rôles ne constituent pas un identifiant. En outre, on connaît précisément le nombre maximum d'instances de rk observées à partir d'une combinaison d'instances des autres rôles.

Qu'en est-il des cardinalités maximales du modèle EA ?

Avant de tenter de comparer multiplicité UML et cardinalité EA, il est sans doute utile de rappeler les propriétés de cette dernière.

- **j = 1** La cardinalité [ik-1] du rôle rk indique que ce rôle **constitue un identifiant de R**.

  R (a[0-N]:A, b[0-N]:B, c[0-**1**]:C) ⇔ R (a, b, **c**)

- **j = N** La cardinalité [ik-N] du rôle rk indique que ce rôle **ne constitue pas un identifiant de R**.

- **j = m > 1 (m ¹ N)** Ce rôle ne constitue pas un identifiant. En outre, on connaît précisément le nombre maximum d'associations R dans lesquelles toute instance de Ck peut jouer le rôle rk.

Equivalence entre cardinalité maximale et multiplicité maximale

Un type d'associations N-aire peut-il être complètement traduit dans un schéma UML ? L'équivalence n'est assurée que pour les associations binaires et pour les associations N-aires dont toutes les multiplicités sont [0..*]. Considérons l'exemple suivant : *une machine peut être empruntée par un employé pour le compte d'un projet; une machine ne peut être empruntée qu'une seule fois (à un moment déterminé)*. La Figure 14 donne les *meilleurs schémas* d'association N-aire selon les modèles UML et EA.

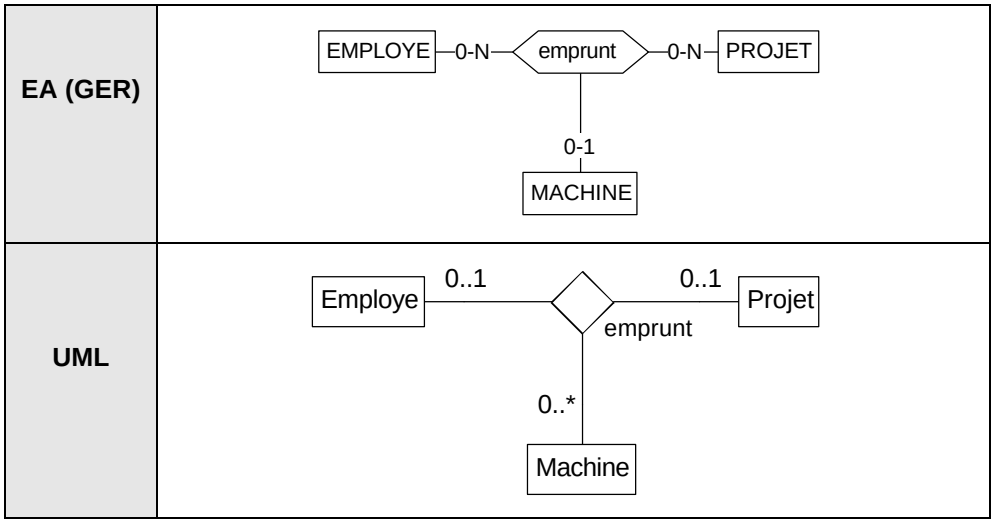

Figure 14 - Expression d'un identifiant d'association composé d'un seul rôle

Le schéma UML ne traduit qu'imparfaitement le fait qu'*une machine ne peut faire l'objet que d'un seul emprunt* : Machine étant l'identifiant de emprunt, et UML ne permettant d'exprimer, dans ce schéma, que des identifiants de 2 rôles, on en est réduit à déclarer deux identifiants non minimaux {Machine,Projet} et {Machine,Employe}.

On pourrait objecter qu'un tel rôle identifiant permettrait de transformer l'association ternaire en deux associations binaires *un-à-plusieurs*, respectivement entre Employe et Machine et entre Projet et Machine. Cette pratique, parfaitement licite, génère cependant une contrainte de coexistence (si une machine est empruntée, elle l'est pour le compte d'un projet, et inversement). En outre, ceci ne règle pas la question des associations dotées d'attributs et de la situation où plus d'un rôle est identifiant.

Remplaçons l'association N-aire emprunt en une classe Emprunt accompagnée de 3 (c'est à dire N) asssociations binaires, selon la transformation d'un type d'associations en type d'entités communément utilisée en ingénierie des bases de données. On obtient ainsi le schéma de la Figure 15, dans lequel il est possible de représenter les identifiants d'associations comprenant un seul rôle. L'association pour matéria-

lise le rôle emprunt.Projet. Elle peut exprimer l'identifiant d'Emprunt mono-composant {Machine} sous la forme de la multiplicité [0..1].

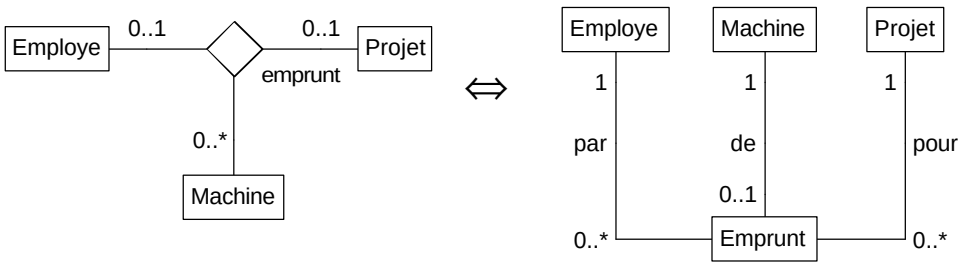

Figure 15 - La réduction d'une association N-aire en associations binaires permet d'exprimer un identifiant d'association mono-rôle sans perte d'information

En revanche, toute association N-aire UML peut être traduite sans perte en un type d'association N-aire, du moins dans les modèles EA qui offrent le concept d'identifiant de type d'associations. Les exemples de la figure 16 expriment qu'*un client ne peut acheter un produit que chez un seul représentant*. L'association achète est dotée d'un identifiant constitué des rôles {Client, Produit}, qui traduit la multiplicité 0..1 du rôle joué par Représentant dans le schéma UML et par un identifiant explicite du type d'associations achète dans le schéma EA.

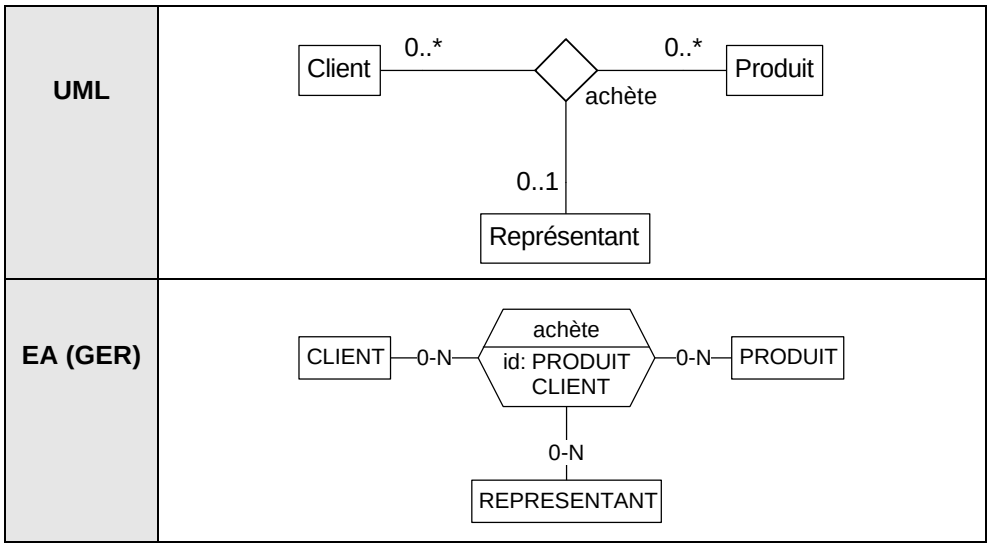

Figure 16 - Expression d'un identifiant d'association composé de deux rôles

Borne maximale d'une multiplicité : premières conclusions

L'analyse qui précède nous permet de tirer les premières conclusions relatives au concept de multiplicité.

1. En UML, pour les valeurs les plus fréquentes (*1* et *), la borne maximale d'une multiplicité d'un rôle d'une association R de degré N indique quels sont les **identifiants de** R **constitués de N-1 rôles**. Il n'existe pas de notation pour indiquer les autres identifiants de R. On mentionnera la proposition de M. Blaha, coauteur de la méthode OMT, qui suggère de remplacer les multiplicités des associations N-aires par la mention explicite de ses identifiants [Blaha 1998].
2. Dans le modèle EA, pour les valeurs les plus fréquentes (*1* et *N*), la borne maximale d'une cardinalité d'un rôle d'un type d'associations R de degré N indique quels sont les **identifiants de** R **constitués de 1 rôle**. Les autres identifiants de R sont indiqués de manière explicite.
3. On observe que la multiplicité maximale d'un rôle rk *ne représente pas une propriété du rôle* (ou de la classe) auquel elle est affectée, mais bien *une contrainte affectant les autres rôles*. Si le concept n'est pas réellement gênant pour les associations binaires, il s'avère souvent d'interprétation complexe pour les associations N-aires. En revanche, la cardinalité maximale d'un rôle rk dans le modèle EA *représente une propriété du rôle* (ou de la classe) auquel elle est attachée.
4. La conversion de notation entre UML et EA sans perte d'information d'une association n'est possible que pour les associations binaires et pour les associations N-aires dont tous les rôles sont de multiplicité 0..*. [11] Dans le cas d'une association binaire, N-1 = 1, de sorte que les deux notations expriment les mêmes identifiants.
5. Pour les associations de degré N > 2, il est possible d'exprimer une multiplicité UML [0..1] par un identifiant EA équivalent. Inversement, une cardinalité EA [0-1] ne peut se traduire par une multiplicité UML équivalente. Les multiplicités N-aires [i..j] où $1 < j < \infty$ ne peuvent se traduire en cardinalités, ni inversement.
6. Si on exprime une association N-aire sous la forme d'une classe accompagnée d'associations binaires (figure 15), alors toutes les formes de cardinalités EA peuvent être traduites sans perte d'information. Cependant, après transformation, les multiplicités N-aires [i..j] où $1 < j < \infty$ ne peuvent plus se traduire en cardinalités équivalentes.

10.3 Borne minimale i de la multiplicité

On distinguera également trois valeurs types : i = 0, i = 1 et i = m (valeur finie > 1)

- **i = 0** La multiplicité [**0**..jk] du rôle rk indique qu'une combinaison quelconque d'instances des classes des autres rôles {r1,...,rk-1,rk+1,...,rN} ne doit pas nécessairement apparaître dans une instance de R. Dans le cas d'une association binaire, cette propriété se traduit par le fait que l'autre rôle est facultatif pour sa classe.

11. La borne minimale doit être 0, comme nous le verrons plus loin.

- **i = 1** La multiplicité [1..jk] du rôle rk indique que toute combinaison quelconque d'instances des classes des autres rôles {r1,...,rk-1,rk+1,...,rN} doit apparaître dans une association R.

  $$R\ (a[0..*]:A,\ b[0..*]:B,\ c[\mathbf{1}..*]:C) \ \Leftrightarrow\ R\ [a,\ b]\ =\ AxB$$

  Dans le cas d'une association binaire, cette propriété se traduit par le fait que l'autre rôle est obligatoire pour sa classe.

- **i = m > 1 (m ¹ *)** L'interprétation est la même que pour i=1. En outre, on connaît précisément le nombre minimum d'instances de rk observées à partir d'une combinaison d'instances des autres rôles.

Discussion

La multiplicité [0..jk], qui n'impose en fait aucune contrainte, ne pose pas de problèmes particuliers. Considérons ensuite les conséquence de i > 0, et plus particulièrement, pour simplifier la discussion, de i=1, dans le cas des associations de degré N > 2. Nous appuyerons l'analyse sur l'exemple de la Figure 17.

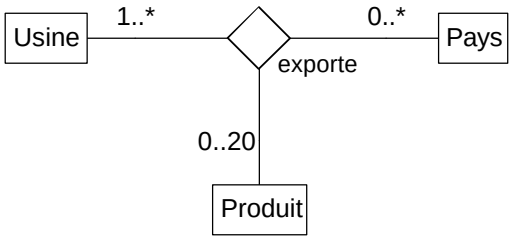

Figure 17 - Exemple d'une multiplicité minimale non nulle.

Cherchons à interpréter la multiplicité [1..*] du rôle exporte.Usine. Celle-ci indique qu'à partir de tout couple <Produit,Pays>, on *voit*, via l'association exporte, de 1 à * instances de Usine. Autrement dit, tout couple <Produit,Pays> apparaît dans au moins une association exporte :

$$exporte[Produit,Pays] = Produit\ x\ Pays$$

Ce fait entraîne des conséquences importantes mais peu intuitives.

1. Si la classe Produit est vide, les éventuels objets de la classe Pays ne sont soumis à aucune contrainte; il en va de même des objets de Pays si Produit est vide.
2. Si la classe Produit contient au moins un objet, alors toute instance de Pays apparaît dans au moins une instance de exporte; le rôle exporte.Pays est donc obligatoire pour Pays quand Produit n'est pas vide. Symétriquement, si la classe Pays contient au moins un objet, toute instance de Produit apparaît dans au moins une instance de exporte; le rôle exporte.Produit est donc obligatoire pour Produit quand Pays n'est pas vide. En synthèse : si, parmi les classes jouant les rôles autres que exporte.Usine (porteur la la multiplicité 1..j), l'une n'est pas vide, alors chacun de ces rôles est obligatoire pour sa classe.

3. Si deux des rôles d'une association N-aire ont une multiplicité 1..j, alors **tous** les rôles de l'association sont obligatoires dès qu'une des classes participantes n'est pas vide.

4. Chaque objet Produit est associé via exporte, non seulement à au moins un objet Pays, mais à tous les objets Pays. Il en va de même de Pays vis-à-vis de Produit. Ainsi, si la base de données contient 100 objets Pays, l'introduction d'un nouvel objet Produit entraînera l'insertion de 100 instances de exporte. On peut exprimer cette contrainte de la manière suivante : *si un produit est exporté, il l'est dans tous les pays connus; si on exporte un produit dans un pays, on y exporte aussi tous les produits répertoriés.*

5. Si les classes Pays et Produit comprennent chacune au moins une instance, alors la classe Usine doit inclure elle aussi au moins une instance. Si Usine est vide, alors Pays ou Produit (ou les deux) sont également vides.

6. Si deux des rôles d'une association N-aire ont une multiplicité 1..j, alors **toutes** les classes participante sont simultanément soit non vides soit vides.

De telles propriétés vont compliquer considérablement la gestion des données. D'une part, lors de la création d'un objet Pays, si Produit est non vide, il faut qu'il existe au moins un objet Usine; de même pour la création d'un objet Produit lorsque Pays est non vide. D'autre part, la création d'un objet Pays doit s'accompagner de la création simultanée d'au moins autant d'instances de exporte qu'il y a d'objets Produit; de même pour Produit par rapport à Pays. Les opérations de suppression d'objets sont aussi soumis à des contraintes dont l'expression est laissée à l'initiative du lecteur.

On peut légitimement se demander si tous les analystes qui spécifient i > 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. Dans notre exemple, l'analyste sera tenté de lire dans le schéma que "*toute usine exporte au moins un produit vers un pays*". Non seulement cet 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 oeuvre.

[UML 1.3] reconnaît lucidement que : "*Multiplicity for N-ary associations may be specified, but it is less obvious than binary multiplicity*". On notera que dans ce cas, les multiplicités ne sont pas considérées comme obligatoires. Comme déjà signalé, [Blaha 1998] suggère de remplacer les multiplicités des associations N-aires par des identifiants d'association, ignorant par là même l'effet de la borne minimale. 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 (voir les notations du type *crow-feet* par exemple [Teorey 1999] et annexe de [Elmasri 2000]).

Qu'en est-il des cardinalités minimales dans les modèle EA européens ?

Tout comme nous l'avons fait pour la borne maximale de la cardinalité, il est utile de rappeler la définition de la borne minimale de cette dernière.

- **i = 0** La cardinalité [0..jk] du rôle rk indique que ce rôle est facultatif pour le type d'entités Ck. Une entité Ck peut exister sans apparaître dans une association R.
- **i = 1** La cardinalité [1..jk] du rôle rk indique que ce rôle est obligatoire pour le type d'entités Ck. Toute entité Ck doit apparaître dans au moins une association R.

$$R\ (a[0..*]:A,\ b[0..*]:B,\ c[\mathbf{1}..*]:C) \quad \Leftrightarrow \quad C = R\ [c]$$

- **i = m > 1** Ce rôle est obligatoire. En outre, on connaît précisément le nombre minimum d'associations R dans lesquelles toute instance de Ck doit jouer le rôle rk.

On observe ici encore que la cardinalité d'un rôle rk représente une propriété du type d'entités (ou classe) qui joue ce rôle. Le concept de rôle obligatoire, qui spécifie qu'une entité d'un type doit participer à au moins une association, ne peut être spécifié en UML, sinon comme effet de bord comme on vient de le voir.

Equivalence entre cardinalité minimale et multiplicité minimale

On vient de voir que l'équivalence n'est assurée que pour les associations binaires, ainsi que pour les associations N-aires dont toutes les multiplicités sont [0..j]. Cependant, la substitution d'un schéma UML purement binaire à l'association N-aire d'origine permet d'exprimer les rôles obligatoires comme l'illustre la Figure 18.

Borne minimale d'une multiplicité : premières conclusions

Nous pouvons tirer des conclusions supplémentaires relatives au concept de multiplicité.

1. Dans le modèle UML, pour les valeurs les plus fréquentes (0 et 1), la borne minimale d'une multiplicité d'un rôle d'une association R de degré N indique si chaque combinaison d'instances des classes des N-1 autres rôles *doivent* (1) ou *peuvent ne pas* (0) *apparaître* dans R. Dans une association binaire, elle indique si l'autre rôle est facultatif ou obligatoire. Il n'existe pas de possibilité de spécifier qu'un rôle d'une association N-aire est obligatoire. Toute valeur i > 0 de la borne minimale entraîne des dépendances complexes indésirables dans les associations N-aires.
2. Dans le modèle EA, pour les valeurs les plus fréquentes (0 et 1), la borne minimale d'une cardinalité d'un rôle d'un type d'associations R indique si ce rôle est facultatif ou obligatoire. Elle n'induit aucune contrainte sur les autres rôles. La cardinalité minimale d'un rôle rk représente une propriété du type d'entités qui joue ce rôle.

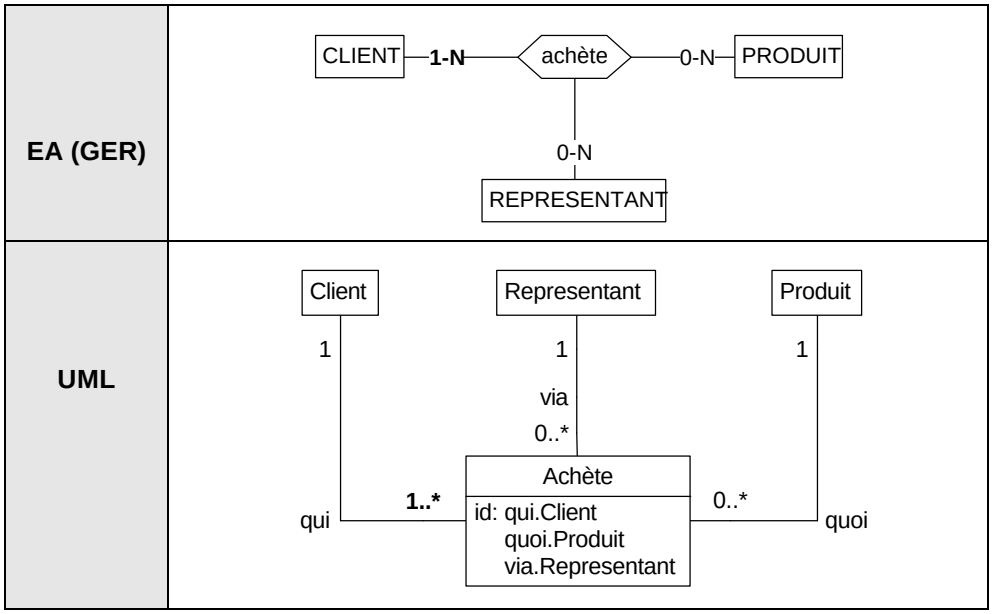

Figure 18 - Une cardinalité minimale supérieure à 0 peut se traduire en UML à condition de transformer l'association N-aire en une classe et N associations binaires

3. Observons encore que la multiplicité minimale d'un rôle rk ne représente pas une propriété du rôle (ou de la classe) auquel elle est attachée, mais bien une contrainte affectant les combinaisons des instances des classes des autres rôles.
4. La conversion de notation d'UML vers EA sans perte de sémantique d'une association n'est possible que pour les associations binaires ou pour les associations N-aires dont tous les rôles sont de multiplicité 0..j.
5. Pour les associations de degré N > 2, il n'est pas possible de dériver complètement une cardinalité d'une multiplicité. Inversement une cardinalité [1-m], ou rôle obligatoire, n'est pas exprimable en UML, sauf via une transformation en associations binaires. Une cardinalité EA [0-n] implique que tous les autres rôles sont de multiplicité UML [0..m].

10.4 Multiplicité : conclusions générales

Afin de simplifier la formulation de ces conclusions, nous distinguerons les associations binaires des associations N-aires.

a) Associations binaires

La multiplicité d'un rôle exprime une contrainte sur les instances de la classe jouant l'autre rôle, ce qui ne facilite pas toujours leur utilisation. Par exemple, une multiplicité [1..j] indique que **l'autre rôle est obligatoire** pour sa classe. L'implémentation

traduira ces contraintes en mécanismes (*check, triggers*) attachés à la table/classe associée au rôle opposé.

Cependant, multiplicité UML et cardinalité EA ont même puissance d'expression.

b) Associations N-aires

La multiplicité d'un rôle joué par une classe exprime une contrainte sur l'ensemble des classes jouant les autres rôles. Une difficulté supplémentaire apparaît : le raisonnement concerne, non pas les instances d'une classe, mais les combinaisons d'instances d'autres classes.

Une multiplicité minimale non nulle entraîne des dépendances parasites particulièrement problématiques non seulement à interpréter, mais aussi à implémenter. Elle ne peut en aucun cas exprimer le simple concept de rôle obligatoire.

Multiplicité et cardinalité expriment des propriétés différentes. La multiplicité maximale permet de définir les identifiants de N-1 rôles, alors que la cardinalité maximale définit les identifiants composés d'un seul rôle. La multiplicité minimale permet de définir la participation obligatoire de toutes les combinaisons d'instances des N-1 autres rôles, tandis que la cardinalité minimale définit plus simplement les rôles obligatoires.

Recommandation

Il tombe sous le sens qu'il faut éviter l'utilisation des associations N-aires en UML. Leur logique et le comportement qu'elles induisent sont trop complexes pour constituer une construction de modélisation fiable et réaliste. Leur réduction sous la forme d'associations binaires sans perte d'information et la possibilité de définir des identifiants de toute nature pour une classe permettent d'ailleurs d'exprimer toutes les propriétés des associations N-aires.

11. Les classes-associations

UML propose un concept hybride qui relève à la fois des classes et des associations. Une classe-association est une association binaire ou N-aire qui jouit des caractéristiques d'une classe : *attributs, opérations, jouer un rôle* dans une autre association. Elle est représentée par une association à laquelle une classe est attachée.

Si la classe-association ne sert qu'à affecter des attributs ou des opérations à une association, la classe peut ne pas être nommée. Dans le modèle EA, on l'exprimera sous la forme d'un type d'asociation avec attributs (Figure 19).

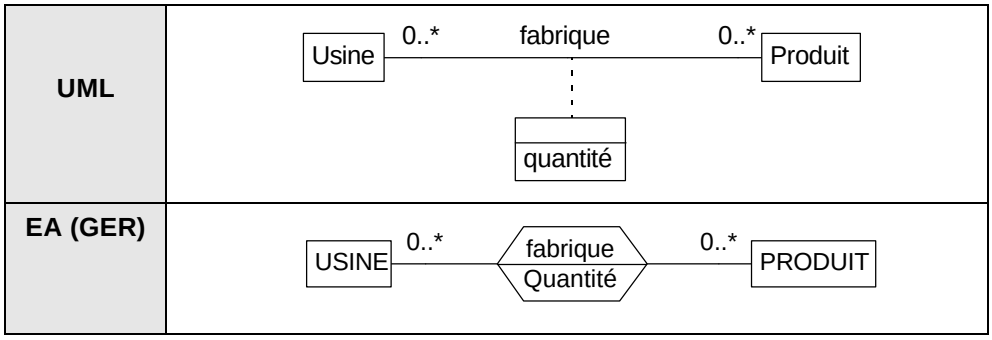

Figure 19 - Une classe-association non nommée et son interprétation dans le modèle EA

Si la classe-association a pour but d'impliquer l'association dans une autre association, alors la classe sera nommée au détriment de l'association (figure 20). Son équivalent dans le modèle EA est un type d'entités *association* résultant de la transformation du type d'associations en type d'entités.

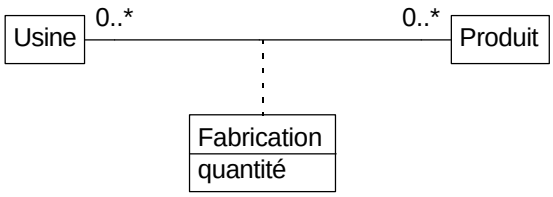

Figure 20 - Une classe-association nommée

12. Les contraintes

Le modèle de classes d'UML est particulièrement pauvre en contraintes. En effet, UML offre une seule contrainte prédéfinie : la contrainte **xor** (*ou exclusif*) entre associations relatives à une même classe, qui correspond à une contrainte *exactement-un* (exact-1) dans le modèle EA. Dans l'exemple de la Figure 21, un compte appartient à une personne ou à une société, mais pas aux deux.

Il existe cependant une autre contrainte importante, qu'on peut définir indirectement via une association qualifiée dont la multiplicité dégénère en [1], et qui représente, comme on l'a vu, une variante limitée d'*identifiant hybride*.

Toute autre contrainte est dite *user-defined*, et n'a donc pas de sémantique standard. UML propose un langage d'expression de contraintes (OCL), qui permet d'exprimer des propriétés sur des ensembles d'objets ou d'instances d'association.

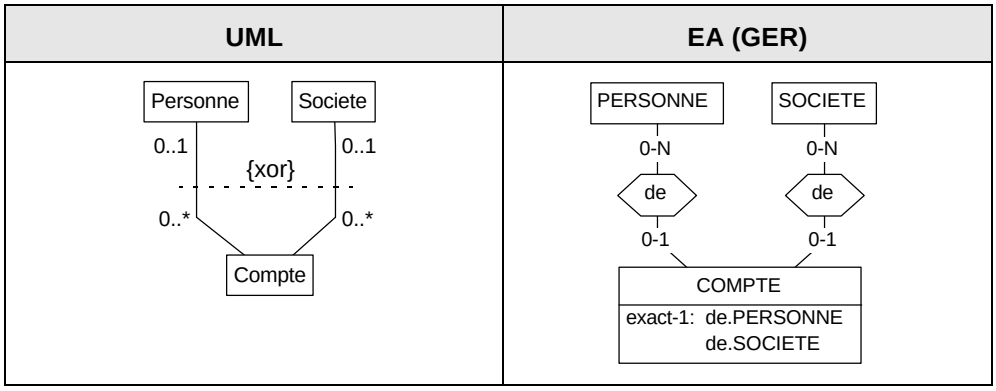

Figure 21 - La contrainte **xor** permet de définir l'exclusion de rôles pour une classe

Malgré son origine (le modèle de Chen), le modèle de classes d'UML n'inclut donc pas les contraintes les plus répandues dans les modèles de bases de données, telles que les *identifiants*, les *contraintes d'existence* autres que l'exclusion, les contraintes d'*inclusion*, etc. La raison le plus communément invoquée est historique : *ces concepts n'existent pas en C++*. Il est donc essentiel de réintroduire ces contraintes indispensables, par exemple via le compartiment des contraintes dans la représentation d'une classe, comme évoqué dans la section 2 (Figure 22).

13. Conclusions

Nous conclurons cet article dans deux directions. Nous tenterons d'abord de comparer les modèles EA et UML, en examinant comment chaque construction de l'un peut s'exprimer dans l'autre. Ensuite, nous définirons une variante du modèle de classes d'UML qui résout les problèmes mis en évidence ci-avant.

13.1 Correspondances UML / EA

Comparer deux modèles est un exercice périlleux en toute généralité : en effet, un concept absent dans un modèle peut souvent être exprimé ou simulé par des constructions induisant des comportements similaires ou par des constructions génériques. D'où l'importance, pour chacun des modèles, des méta-constructions pour représenter l'*inexprimable* :

- stéréotypes, tags et *user-defined properties* en UML;
- stéréotypes, méta-propriétés et noms génériques en GER;
- annotations (purement documentaire) dans tous les modèles.

L'inconvénient de ces constructions est que le modèle n'en définit pas la sémantique, tâche qui est laissée à l'initiative de l'analyste. Ainsi, le stéréotype d'une association

UML peut servir à préciser son niveau d'abstraction, son mode d'implémentation, voire le service de l'entreprise qui est responsable de ses instances. Nous avons utilisé ce concept dans un schéma GER pour simuler la composition (Figure 9).

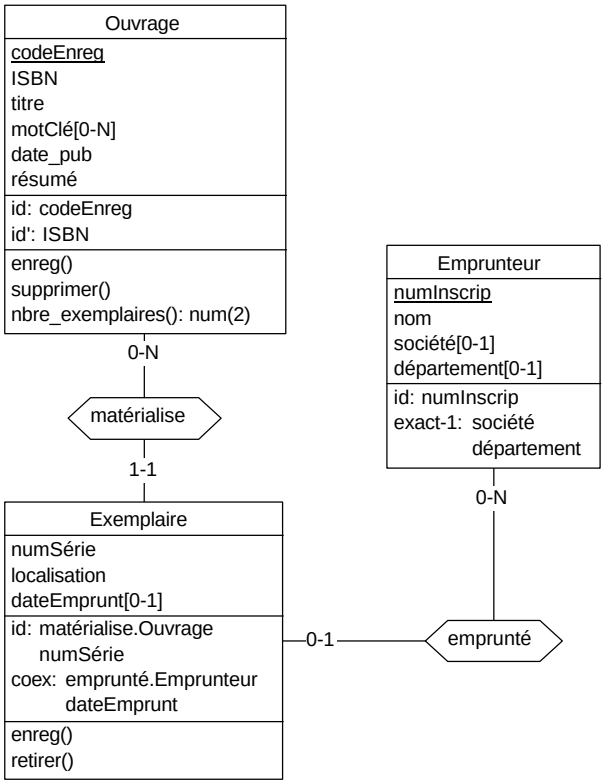

Figure 22 - Un schéma UML illustrant l'importance du compartiment des contraintes

Le tableau ci-dessous établit les principales équivalences entre les concepts du modèle de classes d'UML et ceux du modèle EA. Les abréviations utilisées sont les suivantes : TE = type d'entités, TA = type d'associations, ISA = relation is-a entre un surtype et un sous-type.

UML	EA (GER)	commentaire
classe	type d'entités	
généralisation	relation ISA	*propriétés des sous-classes (T, D) identiques*
discriminateur	-	*EA : via méta-propriété*
attribut simple	attribut atomique	

attribut multivalué	attribut multivalué	*via multiplicité/cardinalité*
-	attribut composé	*UML : via user-defined data type; voir UML+*
attribut facultatif	attribut facultatif	*via multiplicité/cardinalité*
association qualifiée dégénérée	identifiant TE (attributs + 1 rôle)	*EA : par effet de bord*
-	identifiant TE (autres)	*voir UML+*
association qualifiée générale	-	*concept non pertinent*
association binaire	type d'associations binaire	*équivalence complète*
association N-aire	type d'associations N-aire	*équivalence partielle*
multiplicité d'attribut	cardinalité d'attribut	*équivalence*
multiplicité de rôle (binaire)	cardinalité de rôle (binaire)	*équivalentes, inversion*
-	cardinalité de rôle (N-aire)	*voir UML+*
multiplicité de rôle [i..1] (N-aire)	identifiant de TA (N-1 rôles)	
multiplicité de rôle [i..m>1] (N-aire)	cardinalité de groupe	
rôle opposé obligatoire (binaire)	rôle obligatoire (binaire)	
-	rôle obligatoire (N-aire)	
-	identifiant de TA (autres)	*voir UML+*
-	identifiant d'attribut	*voir UML+*
association d'agrégation	-	*sémantique peu claire; EA : via stéréotype ou nom générique*
association de composition	-	*EA : cardinalité [1-1] des composants; stéréotype ou nom générique*
classe-association	TA avec attributs/méthodes; TE-association	*EA : correspond à un TA avec attributs/méthodes; ou à un "TE-association"*
contrainte d'association xor	contrainte "exactly one" entre 2 rôles	

-	contraintes d'existence (autres)	*voir UML+*
-	contraintes (autres)	*voir UML+*
opération	unité de traitement	

13.2 Le modèle de classes d'UML pour l'analyse conceptuelle d'une base de données

Nous avons montré que le modèle de classes d'UML présente un certain nombre d'inconvénients lorsqu'on cherche à l'utiliser pour élaborer des spécifications conceptuelles. En écartant les constructions problématiques ou douteuses, et en ajoutant un minimum de concepts indispensables, on peut obtenir une variante d'UML qui, pour être quelque peu différente de la version communément acceptée, n'en respecte pas moins les spécifications générales qu'on peut trouver dans [UML 1.3].

Posons donc que le modèle de classes d'UML est un support adéquat pour la modélisation conceptuelle pour autant que :

1. on réintroduise le concept d'identifiant de classe comme construction explicite; toutes les configurations d'identifiants de classes (composés d'attributs et/ou de rôles) doivent être autorisées;
2. on adopte le compartiment des Contraintes pour l'expression des contraintes habituelles des bases de données, y compris les identifiants;
3. on retienne éventuellement l'association de composition, mais pas l'association d'agrégation, dont la signification est obscure;
4. on évite l'usage des associations N-aires;
5. on évite l'usage des associations qualifiées.

C'est ce modèle que nous avons désigné sous le terme d'DB-UML. Tout en étant conforme aux recommandations de l'OMG, ce modèle est strictement équivalent à un sous-ensemble du modèle EA (variante GER). Par ailleurs, la possibilité d'appliquer à un schéma UML ou EA des transformations à sémantique constante rend les deux modèles équivalents quant à leur puissance de modélisation conceptuelle, même si le modèle EA, plus riche en possibilités d'expression, permet de construire des schémas plus concis et plus expressifs.

Pour terminer, ajoutons encore qu'une analyse similaire devrait être effectuée pour définir une variante d'UML capable d'exprimer l'essentiel des constructions logiques et physiques des bases de données. Un tel modèle est indispensable si on veut utiliser UML comme support à la rétro-ingénierie, à la réingénierie, à la migration ou à l'interopérabilité des bases de données. La proposition qu'on trouvera dans [UML 2000] est quant à elle une première tentative dans ce sens. Adéquate pour représenter de petits schémas élémentaires, elle s'avère cependant tout-à-fait insuffisante en pratique.

14. Bibliographie

[Abrial 1974] Abrial, J-R., Data Semantics, in *Data Base Management*, North-Holland, 1974.

[Batini 1992] Batini C., Ceri S. et Navathe S., B. - *Conceptual Database Design - An Entity-Relationship Approach*, Benjamin/Cummings, 1992.

[Blaha 1998] Blaha M. et Permerlani W. - *Object-Oriented Modeling and Design for Database Applications*, Prentice Hall, 1998.

[Booch 1994] Booch G. - *Object-Oriented Analysis and Design with Applications*, Benjamin Cummings, 1994.

[Ceri 1997] Ceri S. et Fraternali P. - *Designing Database Applications with Objects and Rules - The IDEA Methodology*, Addison-Wesley, 1997.

[Chen 1976] Chen P., The Entity-Relationship Model - Towards a Unified View of Data, in *ACM TODS*, Vol. 1, No 1, pp. 9-36, 1976.

[Coad 1993] Coad, P., Nicola, J., Object-Oriented Programming, Yourdon, 1993.

[Elmasri 2000] Elmasri R. et Navathe S. - *Fundamentals of Database Systems*, 3rd Edition, Addison-Wesley, 2000.

[Jacobson 1992] Jacobson I., Christerson M., Jonsson P. et Övergaard G. - *Object-Oriented Software Engineering*, Addison-Wesley, 1992.

[Nancy 1996] Nancy D. et Espinasse B. - Ingénierie des systèmes d'information Merise, Deuxième génération, Sybex, 1996.

[OMG 2000] *Relationship service specification*, Version 1.0, April 2000, OMG.

[Rumbaugh 1991] Rumbaugh J., Blaha M., Premerlani W., Eddy F. et Lorensen F. - *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[UML 1.3] *OMG Unified Modeling Language Specification*, Version 1.3, OMG, March 2000.

[UML 2000] *The UML and Data Modeling*, A Rational Software Whitepaper, TP-180 3/00, 2000 (disponible sur le site www.rational.com).