Plan de transformation d'un schéma conceptuel en un schéma XML

Delcroix, Christine

Publication date:
2001

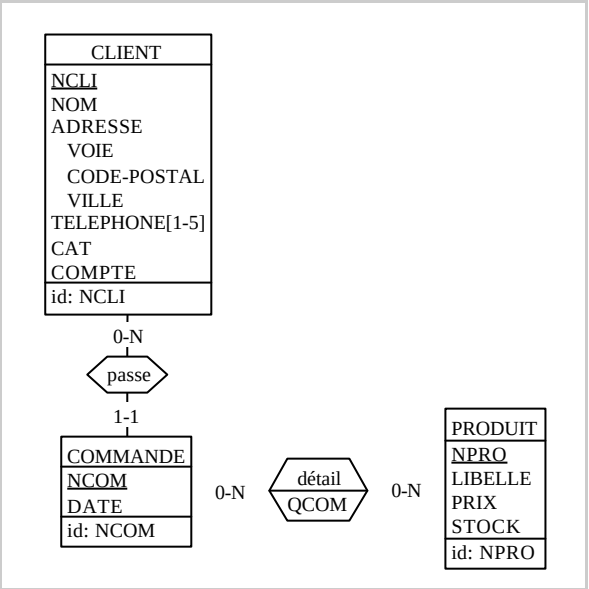
FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR - BELGIUM

Plan de transformation d'un schéma conceptuel en un schéma XML

Christine Delcroix
cde@info.fundp.ac.be

Introduction

Ce document propose un plan de transformation d'un schéma conceptuel en un schéma XML, représentant un DTD. Le plan de transformation comporte neuf étapes. Pour chaque étape, nous précisons l'objectif poursuivi et nous proposons un traitement permettant d'y parvenir. Il va s'en dire que l'utilisateur peut, s'il le souhaite, adopter une autre démarche pour atteindre l'objectif défini. Outre les fonctionnalités courantes de l'outil DB-Main, un certain nombre d'outils spécifiques ont été développés. Nous proposerons pour chaque étape un support logiciel. Enfin, en guise d'illustration, chaque étape sera appliquée sur le schéma conceptuel élémentaire figurant ci-dessous.



Notons que le présent document préconise que l'utilisateur ait pris connaissance des conventions de représentation d'un DTD sous la forme d'un schéma du modèle GER détaillées dans le document "Représentation de structures de données XML dans le modèle GER".

Terminologie :

Un DTD contient un certain nombre de déclarations de type d'éléments telles que :

```
<!ELEMENT CLIENT (Nom, Prénom?, Adresse, Téléphone+, COMMANDES*)>
```

On désigne par **sous-types** d'un type d'éléments, les types d'éléments qui apparaissent dans le contenu de ce type d'éléments. Par exemple, `Nom` est un sous-type de `CLIENT`. A l'inverse, on désigne par **sur-types** d'un type d'éléments, les types d'éléments dans le contenu duquel ce type d'éléments apparaît. Par exemple, `CLIENT` est un sur-type de `Adresse`.

Par abus de langage, au niveau du schéma, on utilise également les termes **sur-type** et **sous-type** pour désigner des relations entre types d'entités lorsque les types d'éléments leur correspondant entretiennent précisément ces relations entre eux.

Configuration :

Les scripts de transformation REMOVE_NON_CONFORM_GROUPS.tfs et TRANSFORM_REL_TYPES.tfs font appel au programme ERAtoXML_Lib.OXO qui doit se trouver dans le même répertoire.

1. Traitement des relations IS-A, des types d'associations complexes et N-N

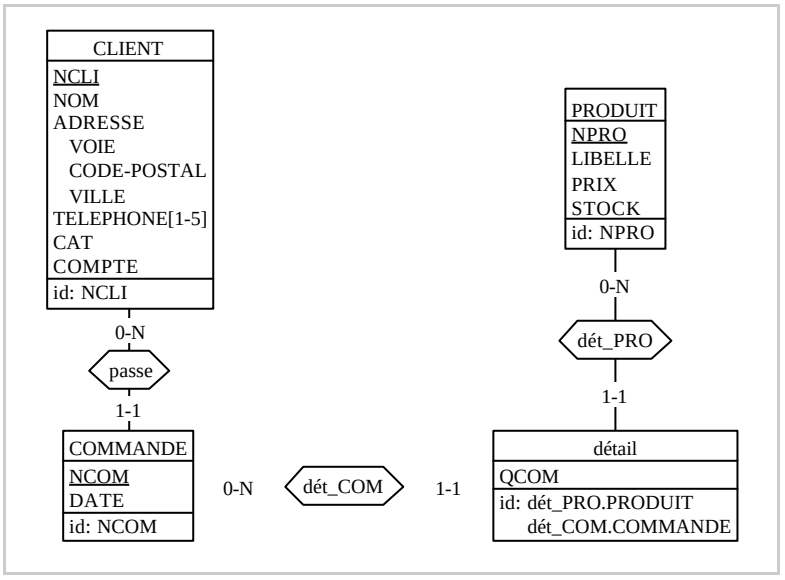
Objectif :
Le but de cette étape est d'éliminer les relations IS-A et les types d'associations complexes ou N-N du schéma. On entend par *type d'associations complexe*, un type d'associations qui comporte des attributs ou dont le nombre de rôles dépasse 2.

Traitement proposé :
Les relations IS-A peuvent être transformées en types d'associations. Les types d'associations complexes ou N-N, quant à eux, seront transformés en types d'entités.

Support logiciel :
L'étape est entièrement automatisée par le recours à l'assistant de transformations globales (Menu *Assist/Global transformation...*) dans lequel on sélectionne :

- *All Is-A* into *rel-types*;
- *Complex rel-types* into *Entity types*;
- *Binary N-N rel-types* into *Entity types*.

Etude de cas :
Le type d'associations *détail* est complexe. Nous l'avons transformé en un type d'entités de même nom.



2. Construction de la structure hiérarchique

Objectif :
Il s'agit dans cette étape, d'élaborer la structure hiérarchique du futur schéma XML à partir du graphe que forme le schéma conceptuel. Au terme de cette étape, chaque type d'entités du schéma devra avoir un sur-type sauf si ce type d'entités est une racine.

Support logiciel :
Il est possible de vérifier automatiquement que l'objectif est rempli par l'exécution du programme VERIF_HIERARCHY.OXO.

Traitement proposé :
Nous avons isolé deux tâches constitutives de cette étape.

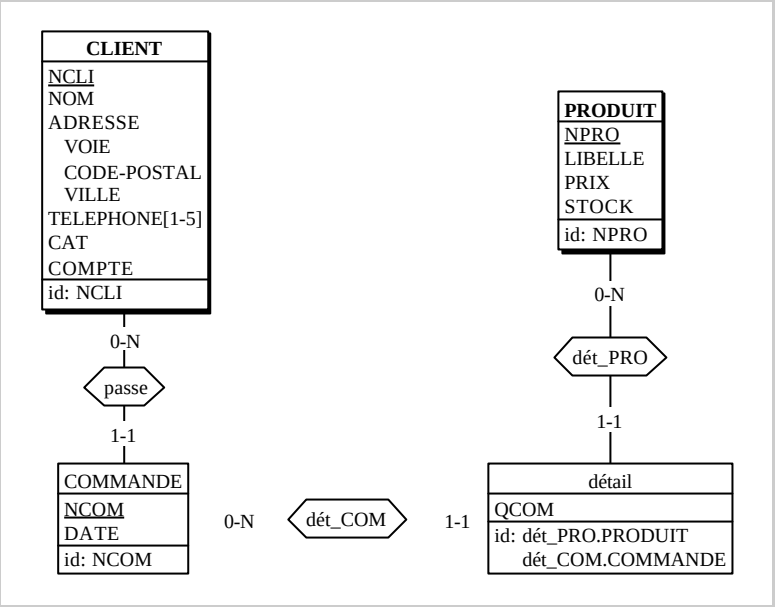
2.1. Déterminer les racines

Objectif :
L'objectif est de marquer les types d'entités racines c'est-à-dire, les types d'entités qui correspondent aux types d'éléments qui apparaîtront au premier ou au deuxième niveau dans le document XML.

Traitement proposé :
Le schéma conceptuel contient un certain nombre de types d'entités qui correspondent **obligatoirement** à des types d'entités racines. Ce sont les types d'entités qui ne jouent aucun rôle de cardinalité égale à 1-1. En effet, ces types d'entités-là ne peuvent être le sous-type d'aucun autre type d'entités et doivent donc nécessairement être une racine. Une première étape est de marquer ces types d'entités. Ensuite, l'utilisateur peut sélectionner d'autres types d'entités comme racine au fur à mesure qu'il construit la structure hiérarchique. Notons que l'étape 2.2. mettra en évidence d'autres racines, qu'il sera assez tôt de marquer à ce moment-là.

Support logiciel :
Les types d'entités racines obligatoires sont automatiquement marqués par le programme MANDATORY_ROOT.OXO. Le programme FACULTATIVE_ROOT.OXO fournit une liste de types d'entités candidats à être une racine. Il s'agit de types d'entités qui ont un certain nombre de sur-types potentiels mais aucun de ces sur-types n'a lui-même un sur-type ou n'est une racine. A l'utilisateur, s'il le souhaite de les marquer.

Etude de cas :
Les types d'entités *CLIENT* et *PRODUIT* n'ont aucun sur-type potentiel car ils ne jouent aucun rôle de cardinalité maximale égale à 1. Aussi, sont-ils tous deux des racines obligatoires. C'est ce qu'indiquent le marquage de ces deux types d'entités.



2.2. Construire la hiérarchie des types d'entités à partir des racines

Objectif :

Cette étape consiste à élaborer la structure hiérarchique du futur schéma XML. Nous connaissons déjà un certain nombre de racines. Il s'agit maintenant de déterminer pour chacune d'elles quels sont ses descendants. Cette étape peut mettre en évidence une ou plusieurs nouvelles racines que l'utilisateur devra marquer.

Traitement proposé :

Au niveau graphique, ce sont les types d'associations qui modélisent les liens entre un type d'entités et chacun de ses sous-types. Le sens du lien type/sous-type est indiqué par le nom d'un rôle selon la règle suivante : dans un type d'associations, le rôle joué par le sur-type porte le nom "f" (pour "father").

Construire la hiérarchie revient donc à nommer un certain nombre de rôles par "f" afin d'indiquer comment s'articulent les différents types d'entités entre eux et en respectant les règles suivantes :

- le rôle joué par un sous-type doit avoir une cardinalité égale à 1-1;
- dans la plupart des cas, un type d'entités n'a qu'un seul sur-type[1];
- un type d'entités marqué (racine) ne peut avoir de sur-type;

Tous les types d'associations du schéma conceptuel ne seront pas conservés dans le schéma final. En effet, si on considère le schéma actuel comme un graphe, l'étape décrite ici consiste, en fait, à extraire de ce graphe, une structure hiérarchique, ce qui implique l'élimination d'un certain nombre d'"arcs" (ici, types d'associations). Au terme de cette étape, les types d'associations qui n'ont aucun rôle nommé "f" correspondent précisément à ces

1. Si un type d'entités a plusieurs sur-types, les rôles que ce type d'entités jouent dans les types d'associations le liant à ses sur-types doivent être de cardinalité 0-1 et doivent être soumis à une contrainte exactly-1 indiquant que chaque élément de ce type apparaît dans le contenu d'un et d'un seul élément du type d'un de ses sur-types. Notons qu'en terme de DTD, cette situation correspond à celle d'un type d'éléments quiapparaît comme sous-type de plusieurs types d'éléments.

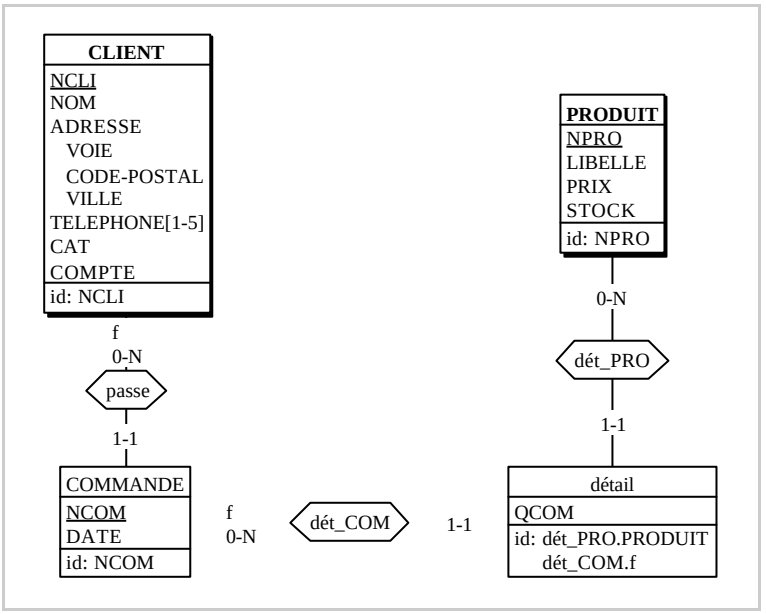
arcs éliminés.

Notons enfin que le mode opératoire décrit dans cette section est en réalité, fort restrictif. L'utilisateur peut appliquer d'autres transformations pour remplir cet objectif, tel par exemple, la transformation d'un type d'entités en attribut, ...

Support logiciel :
Deux programmes permettent de faciliter cette étape :

- MARK_SUBTYPES.OXO part d'un type d'entités sélectionné et cherche tous ses sous-types possibles en respect des règles décrites ci-dessus. Il marque les rôles adéquats par "f" afin d'indiquer la hiérarchie qu'il a établie.
- MARK_DESCENDANT.OXO travaille de manière similaire au programme précédent, si ce n'est qu'il cherche tous les descendants et non simplement les sous-types de premier niveau.

Etude de cas :
Nous décidons d'affecter *CLIENT* comme sur-type de *COMMANDE*. *détail*, quant à lui, peut être le sous-type de *COMMANDE* ou de *PRODUIT*. Nous avons choisi d'associer *COMMANDE* et *détail.* Nous verrons plus loin que le lien entre *détail* et *PRODUIT* sera tout de même conservé, quoique un peu différemment.



3. Raffinement de la structure hiérarchique

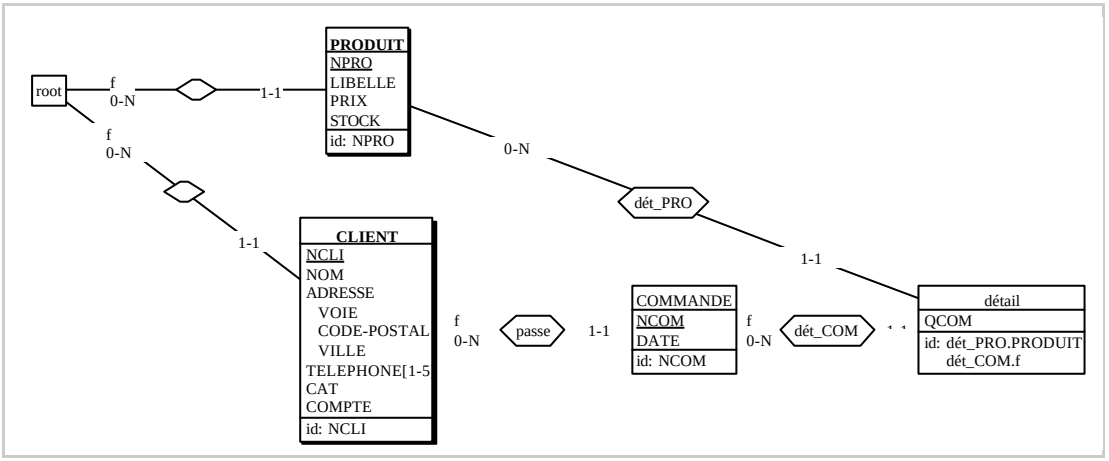
Objectif :
L'objectif est de disposer d'un schéma à racine unique et dont la disposition graphique met en évidence la structure hiérarchique.

Traitement proposé :
Lors de l'étape précédente, plusieurs types d'entités ont pu être choisis comme racines. Cependant, la norme XML impose qu'il n'y ait qu'un seul type d'éléments racine. Aussi,

si tel n'est pas le cas, il faut créer une racine unique au schéma. Cette nouvelle racine est définie comme sur-type des anciens types d'entités racines.
Le schéma peut être retracé afin de mettre en évidence sa structure hiérarchique à racine unique.
Maintenant que la structure hiérarchique du schéma apparaît plus clairement, c'est peut-être l'occasion pour y apporter les dernières modifications.

Support logiciel :
Le programme ADD_UNIQUE_ROOT.OXO analyse le schéma afin de détecter combien de type d'entités sont racines (sont marqués). Si il y en a plus qu'un, il crée une nouvelle racine qu'il relie au reste du schéma en la spécifiant comme sur-type des anciens types d'entités racines. Le nom de la nouvelle racine ainsi que les cardinalités des rôles qu'elle joue peuvent être modifiés après exécution.
Le programme DRAW_SCHEMA.OXO réalise le repositionnement graphique de manière à mettre en évidence la structure hiérarchique du schéma.
Si des modifications ont été apportées à la structure hiérarchique, il est prudent de réexécuter le programme VERIF_HIERARCHY.OXO afin de s'assurer que la structure du schéma est bien de nature hiérarchique.

Etude de cas :
A l'étape précédente, nous avons déterminé deux racines. C'est pourquoi, à ce stade-ci, on ajoute une racine unique *root*. Le schéma est redessiné de sorte que les types d'entités de même niveau soient positionnés sur la même colonne.



4. Spécifier le type de contenu

Objectif :
Chaque type d'entités du schéma représente un type d'éléments du DTD. Le contenu d'un type d'éléments au sein du DTD est constitué d'un certain nombre de sous-types qui sont organisés selon deux modes : *sequence* ou *choice.* Maintenant que les sous-types de chaque type d'entités ont été identifiés, il est important de spécifier leur mode d'organisation.

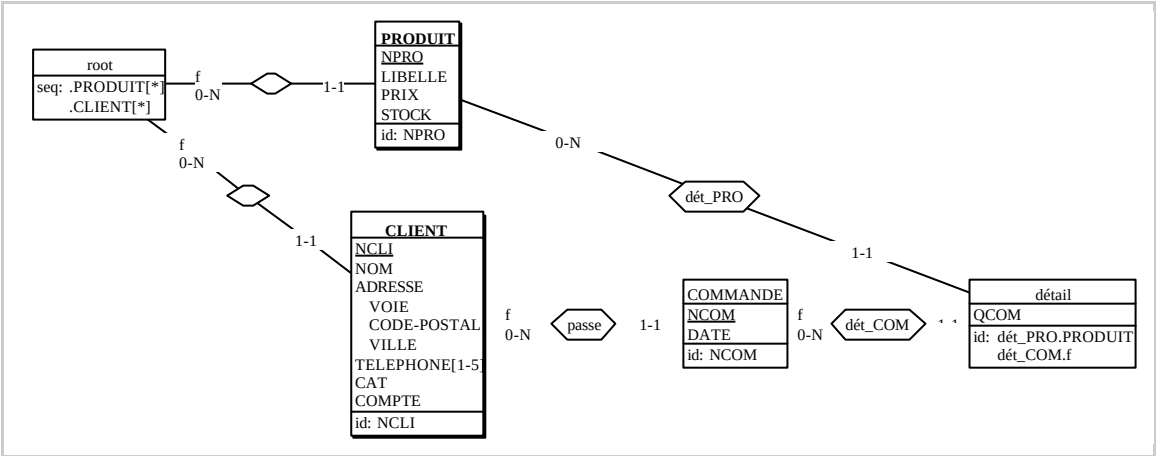
Traitement proposé :
Par convention, le mode d'organisation des sous-types d'un type d'entités est représenté par le biais d'un groupe attaché à ce type d'entités, soumis à une user-constraint *seq* ou

choice et contenant, dans un ordre précis, le rôle joué par chacun de leur sous-type. La présence de ce groupe n'est pas requise lorsque le type d'entités ne possède qu'un seul sous-type. Cette étape consiste donc à créer dans chaque type d'entités contenant au moins un sous-type, un groupe contenant les rôles joués par les sous-types.

Support logiciel :
Le programme CREATE_SEQ.OXO crée dans chaque type d'entités ayant plus d'un sous-type, un groupe soumis à une user-constraint *seq* et contenant le rôle joué par chacun de ses sous-types. L'ordre des rôles dans le groupe est arbitraire.

Etude de cas :
Seul le type d'entités root a deux sous-types, *PRODUIT* et *CLIENT*. *root* se voit donc affecté d'un groupe soumis à une user-constraint *seq* contenant les rôles joués par les deux sous-types. Au niveau du DTD, ceci correspond à la déclaration :

<!ELEMENT root (PRODUIT*, CLIENT*)>



5. Eliminer les types d'associations non marqués

Objectif :
A ce stade, le schéma contient des types d'associations dont un rôle est nommé "f" et d'autres ne vérifiant pas cette condition. Ces derniers doivent disparaître. C'est précisément l'objectif de cette étape. Au terme de cette étape, tous les types d'associations du schéma ont un rôle nommé "f" et sont définitifs.

Traitement proposé :
XML offre deux mécanismes permettant de lier des éléments entre eux. Le premier est l'imbrication d'un élément dans un autre. Au niveau de la structure, cela se traduit par le fait qu'un type d'éléments soit le sous-type d'un autre et au niveau graphique, ce type de lien est modélisé par la présence d'un type d'associations joignant les types d'entités correspondants aux types d'éléments.

Le second mécanisme consiste à attacher au premier type d'éléments, un attribut IDREF dont chaque valeur référence celle d'un attribut ID attaché à l'autre type d'éléments. Ce mécanisme rappelle le mécanisme de clef étrangère. C'est pourquoi **les types d'associa-**

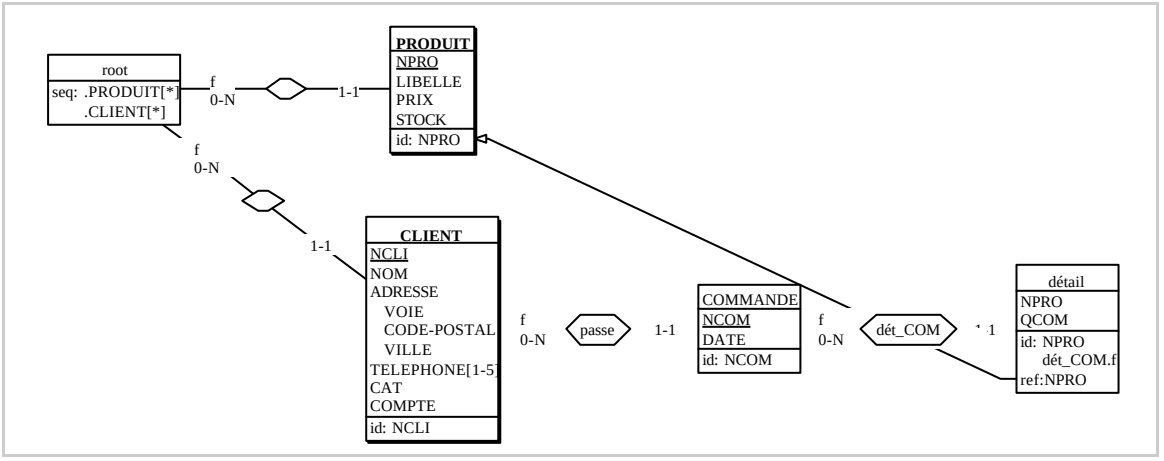
tions non marqués peuvent être transformés en clef étrangère pour, dans un second temps, être transformés en attribut ID/IDREF. Notons que seuls les groupes référentiels constitués d'un seul attribut simple pourront être dérivés en attributs IDREF au sens de XML. De plus, les contraintes référentielles qui possèdent une contrainte d'égalité ne pourront traduire cette caractéristique en terme de XML. Notons enfin que certains types d'associations ne peuvent être transformés en clef étrangère parce qu'aucun type d'entités associé ne possède d'identifiant. Il suffit dans ce cas d'ajouter un identifiant technique à un des deux types d'entités.

Au lieu de transformer un type d'associations non marqué en clef étrangère, on peut choisir de l'**éliminer** purement et simplement. Enfin, une troisième alternative existe dans le cas où un des deux types d'entités liés par le type d'associations non marqué ne joue qu'un seul rôle et ne possède qu'un seul attribut simple, en d'autres mots, dans le cas où un des deux types d'entités est pendant. On peut dans ce cas, **transformer ce type d'entités en attribut**.

Support logiciel :
Le script de transformation globale TRANSFORM_RELTYPES.tfs (à lancer à partir du menu *Assist/Advanced global transformations.../Load*) transforme en clef étrangère, tous les types d'associations non marqués possibles.

Le programme VERIF_ALL_RT_CONFORM.OXO permet de vérifier que l'objectif de l'étape a été atteint en affichant les types d'associations non marqués présents dans le schéma.

Etude de cas :
Le schéma ne contient qu'un seul type d'associations non marqué, *dét-PRO* entre le type d'entités *détail* et *PRODUIT.* Nous voulons garder une trace du lien entre *détail* et *PRODUIT.* Aussi, nous choisissons de transformer *dét-PRO* en clef étrangère.



6. Modifier la cardinalités des rôles des types d'associations

Objectif :
Dans un DTD, un type d'éléments, lorsqu'il apparaît dans le contenu d'un autre peut être soumis à un opérateur de cardinalité. Cependant, le DTD ne permet pas de définir des cardinalités aussi précises que 1-5, 3-7 ou 5-N. Les seules cardinalités que peut exprimer le DTD sont 0-1, 1-1, 0-N, 1-N. Au niveau du schéma, ces cardinalités correspondent à la cardinalité des rôles marqués "f".
L'objectif de cette étape est d'obtenir un schéma où la cardinalité de chaque rôle est valide par rapport au modèle XML, à savoir 0-1, 1-1, 0-N et 1-N pour les rôles "f" et 1-1 pour les autres.

Traitement proposé :
La manière de procéder jusqu'à maintenant nous assure que la cardinalité des rôles non "f" est valide, sauf si elle a été modifiée manuellement.
Ramener la cardinalité des rôles "f" à l'une des quatre cardinalités admises revient à faire la démarche suivante :

- si la cardinalité minimale est supérieure à 1, elle est ramenée à 1;
- si la cardinalité maximale est supérieure à 1, on lui affecte la valeur N.

Avec cette démarche, les cardinalités 1-5, 3-7 et 5-N sont toutes trois transformées en 1-N.

Support logiciel :
Le programme CARDI_ROLE.OXO modifie la cardinalité des rôles "f" selon le traitement proposé ci-dessus.
Le programme VERIF_CARDI_ROLE.OXO assure que l'objectif de l'étape soit rempli.

Etude de cas :
Le schéma ne présente aucun rôle dont la cardinalité est non conforme au modèle XML. Aussi, nous nous abstenons de tout changement.

7. Traitement des groupes

Objectif :
Les seuls groupes admis dans un schéma XML sont ceux soumis à une user-constraint *seq*, *choice*, *gid* ou *idref*. Au terme de cette étape, le schéma ne contiendra plus que de tels groupes.

Support logiciel :
Le programme VERIF_ALL_GROUPS_CONFORM.OXO vérifie que l'objectif de l'étape soit rempli.

Traitement proposé :
Nous avons identifié trois étapes pour remplir cet objectif.

7.1. Traitement des groupes référentiels

Le schéma contient, à ce jour, des groupes référentiels dont la cible est un groupe identifiant. La norme XML définit un mécanisme similaire aux contraintes référentielles par le biais d'attributs IDREF référençant les valeurs d'un attribut de type ID. Bien que la portée de l'identifiant dans le langage XML soit plus globale que celle du paradigme entité-association, on admet qu'il est possible de globaliser un tel identifiant afin qu'il soit valide dans la norme XML. Aussi, les groupes référentiels, lorsqu'ils ne sont composés que d'un attribut simple (éventuellement multi-valué ou optionnel) peuvent-ils être transformés en un groupe soumis à une user-constraint nommée *idref*. Le groupe identifiant ciblé par ce groupe référentiel, si n'est plus la cible d'aucun autre groupe référentiel peut, quant à lui, être transformé en groupe soumis à une user-constraint *gid*.
Une alternative au traitement proposé des groupes référentiels est la suppression.

Support logiciel :
Le programme TRANSFORM_ID_REF.OXO transforme tous les groupes référentiels et leurs groupes identifiants cibles en groupes soumis à une user-constraint *idref* et *gid*.

7.2. Traitement des identifiants

Nous l'avons déjà dit, les identifiants simples du schéma peuvent être transformés en attributs identifiants au sens de XML. Graphiquement, il s'agit de transformer un identifiant en un groupe soumis à une user-constraint *gid*. Cependant, un type d'éléments XML ne peut se voir attacher qu'un seul attribut de type ID. Aussi, cette transformation n'est permise que sur les types d'entités n'ayant pas encore de groupe soumis à une user-constraint *gid*.

Notons qu'une alternative à cette transformation est la suppression de la contrainte identifiante sur un attribut. C'est ce qu'il faudra de toute façon appliquer sur les identifiants qui ne sont pas constitués d'un seul attribut simple.

Support logiciel :
Le programme TRANSFORM_ID.OXO transforme tous les groupes identifiants valables en groupes soumis à une user-constraint *gid*.

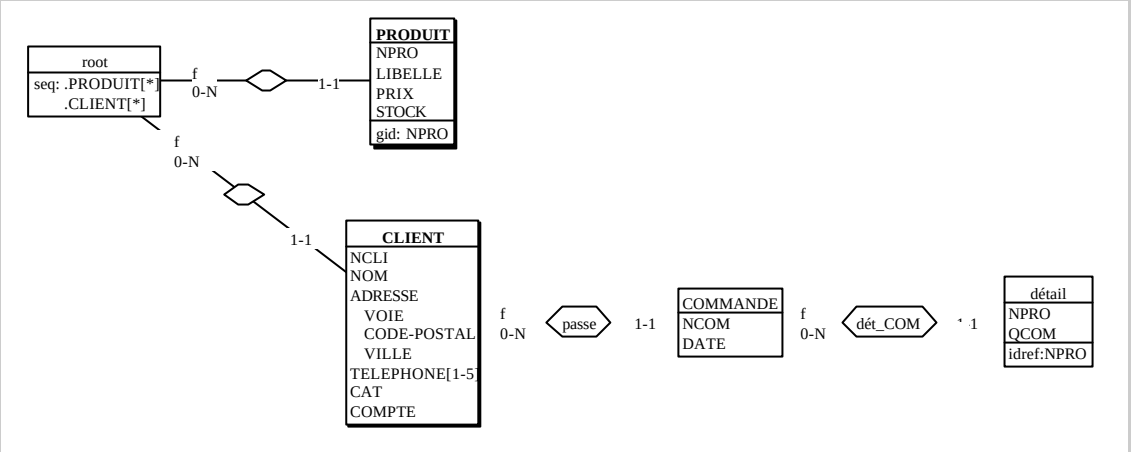
7.3. Traitement des autres groupes

Il s'agit de traiter les groupes non conformes au modèle XML. Dans certain cas, ils pourraient être transformés en groupes conformes tels certains groupes *exactly-1* peuvent avoir un groupe *choice* équivalent. Cependant, à ce jour, aucun traitement spécifique n'a été développé et on propose simplement de les supprimer.

Support logiciel :
Le script de transformation REMOVE_NON_CONFORMES_GROUPS.tfs (à lancer à partir du menu *Assist/Advanced global transformations/Load/...*) supprime tous les groupes non soumis à une user-constraint *gid, idref, seq* ou *choice*.

Etude de cas :
Le groupe référentiel contenu dans le type d'entités *détail* ne contient qu'un seul attribut simple. Il peut être transformé en groupe soumis à une user-constraint *idref*, afin d'être i*n*

fine traduit en attribut XML de type IDREF. Le groupe identifiant cible de *PRODUIT*, quant à lui, est transformé en groupe soumis à une user-constraint *gid*. Cette transformation est obligatoire car, dans XML, le domaine de valeurs d'un attribut IDREF est la réunion des domaines de valeurs de tous les attributs ID.
Les autres groupes identifiants, quant à eux, sont supprimés purement et simplement même si, à l'exception de celui contenu dans *détail*, nous aurions pu les transformer en groupe *gid*.



8. Traitement des attributs

Objectif :
Un schéma XML admet un certain nombre d'attributs. Ces derniers sont en correspondance directe avec les attributs au sens de XML ou bien sont des attributs techniques tels que les attributs nommés *#pcdata* ou *#any*. L'objectif de cette étape est de transformer les attributs du schéma afin que les structures qui en découlent soient toutes conformes au modèle XML. Notons que les attributs appartenant à un groupe soumis à une user-constraint *gid* ou *idref* sont déjà conformes au modèle XML et ne nécessitent donc plus aucun traitement.

Traitement proposé :
Un **attribut simple et mono-valué** de premier niveau peut, au choix de l'utilisateur, être modifé pour correspondre à un attribut XML de type CDATA ou à un type d'éléments dont le contenu est de type textuel pur #PCDATA. La première alternative nécessite de changer le type de l'attribut afin qu'il soit du type *user-defined* nommé *cdata*. La seconde alternative implique de transformer l'attribut en type d'entités (via la transformation *Attribute* $\rightarrow$ *Entity type/instance representation*), de renommer l'attribut par *#pcdata* et de faire les modifications d'usage afin d'exprimer que le nouveau type d'entités est sous-type du type d'entités qui contenait l'attribut avant transformation (à savoir mise à jour du groupe *seq* ou *choice* et de nommer le rôle joué par le sur-type par *f*). Eventuellement, un groupe identifiant est créé par la transformation et il exige d'être traité comme décrit dans la section précédente.

Un **attribut simple et multi-valué** se ramène à une liste d'attributs simples et mono-valués par la transformation *Multi* $\rightarrow$ *List single*. Ces attributs simples doivent à leur tour

subir un des traitements proposés ci-dessus.
Une deuxième manière de traiter un attribut simple et multi-valué est de le transformer directement en un type d'éléments de contenu textuel pur #PCDATA via la transformation *Attribute → Entity type/instance representation.* Les même mises à jour que pour l'attribut simple et mono-valué doivent être faites. De plus, il y a lieu de vérifier que la cardinalité des rôles ainsi créés soit conforme au modèle XML.

Un **attribut composé et mono-valué** se ramène au cas d'attributs simples en le désagrégeant via la transformation *Disaggregation.* Les attributs simples doivent à leur tour être traité selon un des traitements préconisés ci-dessus.
Un attribut composé et multi-valué peut être également transformé en type d'entités de même nom en appliquant la transformation *Attribute → Entity type/instance representation*. Le nouveau type d'entités contient à son tour des attributs simples qu'il convient également de traiter. De plus, la transformation nécessiste ici aussi d'être complétée par les ajustements décrits plus haut.

Un **attribut composé et multi-valué** peut, comme son précédent, être transformé en type d'entités de même nom. Aux ajustements décrits plus haut s'ajoute celui de vérifier la cardinalité des nouveaux rôles.

Support logiciel :
Le traitement des attributs étant extrêmement libre, nous proposons d'adopter la démarche suivante :

- marquer un ensemble d'attributs devant subir le même traitement;
- déclencher un programme qui traite l'ensemble des attributs marqués.

Le marquage des attributs n'a pas fait l'objet d'un développement spécifique puisque l'assistant de transformations (Menu *Assist/Advanced global transformations...*) est suffisamment riche. En effet, les transformations primitives *MARK* et *UNMARK* permettent de marquer et de démarquer les objets graphiques. La sélection des objets graphiques se fait principalement en utilisant les prédicats suivants, qui peuvent être combinés ensemble :

- attributs de premier niveau : DEPTH_of_ATT(1 1);
- attributs simples : SUB_ATT_per_ATT(0 0);
- attributs ne faisant partie d'aucun groupe : PART_of_GROUP_ATT (0 0)

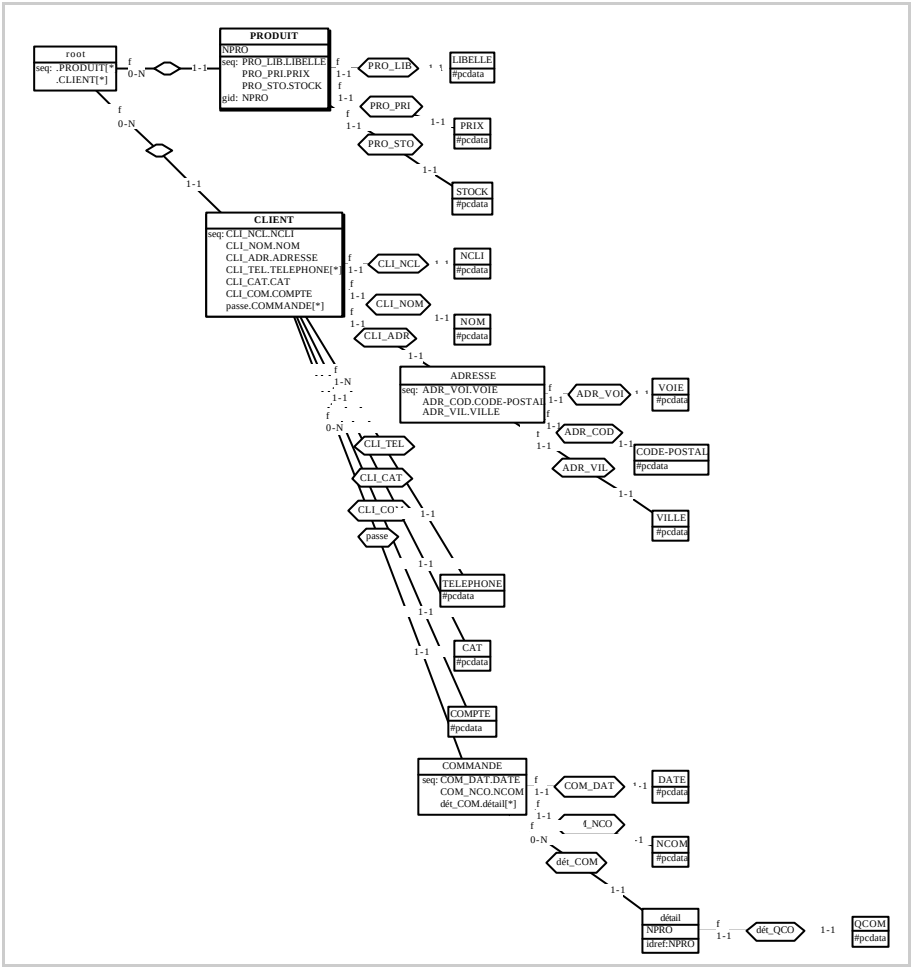
Il peut être utile de désélectionner tous les attributs nommés "#pcdata". Cela peut-être fait très facilement manuellement en utilisant la vue textuelle triée.
Quant aux transformations, nous avons développer deux programmes agissant sur l'ensemble des attributs marqués du schéma de la manière suivante :

- ATT_TO_ET.OXO transforme tous les attributs marqués en un type d'entités de même nom. Il effectue les ajustements préconisés ci-dessus. Ce programme n'accepte pas de transformer des attributs qui sont inclus dans un groupe de type "gid" ou "idref". Il ne prend pas en charge le traitement des nouveaux groupes qui ont pu être créés par la même occasion;

- ATT_TO_CDATA.OXO transforme tous les attributs marqués en attributs correspondants aux attributs XML de type CDATA. Le programme agit donc sur chaque attribut marqué mais sous réserve que l'attribut soit de simple, mono-valué et de premier niveau. Si le type *user-defined cdata* n'existait pas, il prend en charge sa création.

Etude de cas :

Nous avons choisi de ne conserver sous la forme d'attribut, que ceux qui avaient été définis dans les étapes précédentes comme étant de type *gid* ou *idref*. Les autres attributs sont tous transformés en type d'entités correspondants à des types d'éléments de contenu textuel pur.



9. Renommination

Objectif :

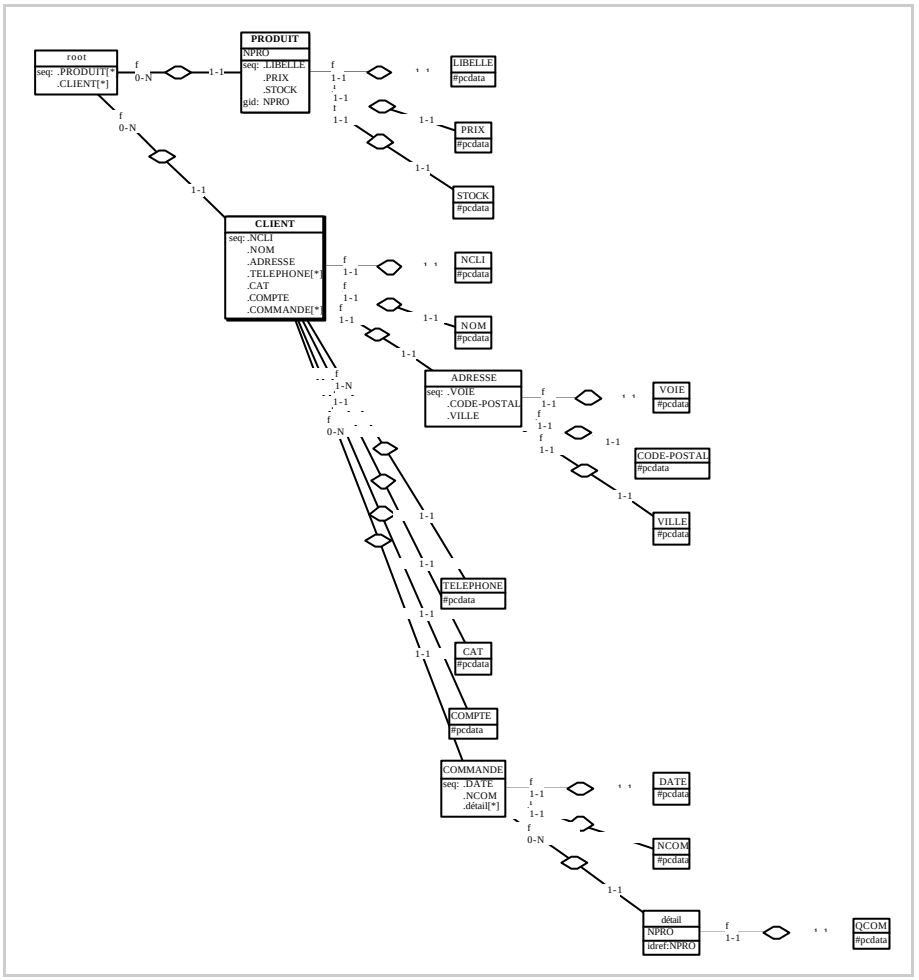
Le nom de certains objets est superflu ou peu explicite, aussi peut-on le changer pour plus de lisibilité.

Traitement proposé :

Le nom des types d'associations est superflu. Aussi, pour qu'il n'apparaisse plus à l'écran, peut-on le préfixer par "|". Les rôles dont le nom diffère de "f" peuvent rester sans nom.

Support logiciel :
Le programme RENAMING.OXO exécute automatiquement le traitement proposé ci-dessus.

Etude de cas :
Pour clôturer cette étude de cas, nous avons appliqué à la lettre le traitement préconisé plus haut.



10. Validation du schéma conformément au modèle XML

Objectif :
L'objectif est de s'assurer que le schéma est conforme au modèle XML

Support logiciel :
Le script d'analyse XML.ana, lancé à partir de l'assistant d'analyse (Menu *Assist/Schema analysis/Load)* permet de remplir cet objectif. Cependant, afin que l'exécution se déroule sans encombre, il faut charger préalablement la librairie de prédicats, XML.anl, utilisée par ce script via le bouton *Edit library/Load library.* Cette librairie fait appel au programme XML.OXO qui doit être enregistré dans le même répertoire qu'elle.