Transposition relationnelle d'un modèle objet objet par prise en compte des contraintes d'intégrité de niveau instance

Guy Pierra, Hondjack Dehainsala, Nadège Ngabiapsi Negue, Mounira Bachir

Transposition relationnelle d'un modèle objet par prise en compte des contraintes d'intégrité de niveau instance

Guy Pierra, Hondjack Dehainsala, Nadège Ngabiapsi Negue, Mounira Bachir

LISI-ENSMA 86961 Futuroscope Cedex, France

{pierra,hondjack,ngabiapsi,mounira.bachir}@ensma.fr

RÉSUMÉ. *Nous proposons dans cet article une nouvelle approche de transposition des modèles objets dans un univers relationnel. Notre approche se fonde sur un raffinement des contraintes usuellement représentées dans les modèles conceptuels. Ce raffinement consiste à identifier les différentes structures possibles des instances. Ces contraintes affinées servent de base pour définir de nouvelles règles de transposition d'un modèle objet. Le modèle résultant, moins contraint que le modèle de départ, possède une capacité de représentation de l'information au moins égale à celle du modèle de départ, et son exploitation relationnelle est beaucoup plus efficace que celle d'un modèle resultant d'une méthode usuelle de transposition. La correspondance entre modèles est elle-même représentée comme un modèle. Cela permet alors, par les méthodes classiques d'ingénierie des modèles, d'écrire de façon générique tous les programmes de transformation, tels que la migration d'instances. La mise en œuvre de l'approche proposée est également présentée.*

ABSTRACT. *We propose in this paper a new approach for mapping object models onto relational schemas. Our approach is based on a refinement of the constraints usually represented in conceptual models making it possible to identify the various possible structures of instances. These refined constraints are used for defining new rules for transformation of object models into models very easy to implement in a relational database. Exploitation of resulting models are considerably more efficient than relational models resulting from usual approaches. The mapping between models is itself represented as a model. This allows to write in a generic way all the transformation programs, such as instances migration, using model management technics. We finally present the results of an implementation of our approach for the mapping of a large size object model.*

MOTS-CLÉS : *Base de données, modélisation objet, ingénierie des modèles, ingenierie dirigées par modèles*

KEYWORDS: *Database, Object modeling management*

1. Introduction

Les Bases de Données (BD) occupent une place essentielle dans beaucoup de systèmes informatiques et ont beaucoup évoluées ces dernières années. Les bases de données relationnelles (BDR) s'imposent de plus en plus par rapport aux BDs objets pour des raisons à la fois de performances et de simplicité. Par contre, l'inverse se produit dans l'univers des formalismes de modélisation conceptuelle. En effet, les formalismes objets (UML, OMT) ayant un pouvoir d'expression (héritage, agrégation, polymorphisme, etc.) beaucoup plus riche que les formalismes de type Entité-Relation, ont tendances à se généraliser. Pour pallier à cette divergence des formalismes de modélisation et d'implantation dans le domaine des BDs, de nouvelles approches de bases de données ont été proposées visant à concilier les deux univers. Ces approches peuvent être classées en deux catégories. La première consiste à *étendre* le système relationnel par l'ajout de concepts objets (classes, agrégation, collection, polymorphisme) [LIU 91]. Cette approche a conduit récemment à la définition d'une nouvelle norme SQL99, dite relationnel objet [ISO 99]. La seconde consiste à *transposer* les concepts objets en relationnel. Pour chaque concept objet, une correspondance est définie avec les concepts du relationnel pour permettre sa représentation en termes relationnels [ERI 04][RAH 00].

La première approche, n'est actuellement implémentée que très partiellement par les fournisseurs de SGBD. Oracle n'implémente pas, par exemple, l'héritage de tables, à l'inverse de PostgreSQL. Les schémas logiques de ces systèmes ne sont donc pas portables. D'où l'intérêt majeur de l'approche basée sur les transpositions de modèles ou "mappings"(Hibernate[HEU 03], iBatis[KLA 04],OntoDB [PIE 05], etc.) Actuellement, aucune approche systématique de transposition n'est satisfaisante dans tous les cas. La complexité de représentation de certains concepts (héritage + polymorphisme, agrégation + polymorphisme) peut engendrer de mauvaises performances (coût de jointure, nombre de tables, etc.) dans les BDRs selon les données à traiter [RAH 01][ERI 04][RAH 00]. Une solution souvent utilisée pour améliorer la performance est alors de définir, ou de modifier, la transposition de façon empirique afin d'optimiser le modèle cible compte tenu de la connaissance que l'on a du domaine modélisé. Mais on perd alors tous les avantages qu'apporte la représentation formelle du mapping pour les traitements ultérieurs : migration d'instances du modèle objet vers le modèle relationnel par exemple.

Le but de cet article est de proposer une nouvelle approche basée sur l'exploitation fine des contraintes d'intégrité existant au niveau des instances de chaque modèle objet pour transformer de façon systématique un modèle objet avant de définir sa représentation relationnelle. Par contrainte de niveau instance, nous signifions les contraintes précises que l'on peut associer à des ensembles d'instances mais qui sont en général masquées par les formalismes d'expression de contraintes utilisées. Ainsi, une cardinalité 0:1 au niveau type, masque en réalité souvent deux classes d'instances très différentes dont la caractérisation précise permettrait de leurs associer des représentations relationnelles différentes. L'approche que nous proposons se base essentiellement sur deux types de telles contraintes. La contrainte *used_once* (qui correspond pour partie,

en UML 2.0, au losange noir) permet de mettre en évidence une relation de composition d'une classe à l'intérieur de plusieurs classes composites. Chaque composant peut alors se représenter à l'intérieur du composite pertinent, ce qui supprime des relations de compositions, et donc des jointures ultérieures. La *redéfinition de type* (qui consiste à redéfinir par restriction, lors d'une spécialisation d'une classe, le type d'un attribut hérité) permet de supprimer des classes abstraites tout en conservant la possibilité de représentation du polymorphisme. Cet élagage de la hiérarchie supprime également des jointures ultérieures. On peut ensuite, et c'est ce que nous faisons ici, combiner ces transformations avec une mise à plat de tout ou partie des hiérarchies subsistantes. Ceci débouche sur un modèle facile à représenter dans un univers relationnel, et d'une efficacité bien supérieure à celle obtenue avec l'utilisation directe de l'une des trois méthodes classiques de représentation des modèles objets dans l'univers relationnel. A notre connaissance, c'est la première fois que la prise en compte systématique de telles contraintes lors de la transposition relationnelle d'un modèle objet est proposée.

Notre approche s'inscrit dans une démarche dirigée par les modèles, en particulier pour *l'échanges entre systèmes hétérogènes*, sans aucune prise en compte de la manière dont cette information sera ensuite représentée au sein de chaque système, archivée ou exploitée. C'est la problèmatique de l'échange de données. Le langage de modélisation que nous utilisons, EXPRESS, a spécifiquement été développé dans ce but. Il contient en particulier un langage d'expression de contraintes très puissant permettant de limiter les interprétations possibles des données échangées. Ce sont précisement ces contraintes, dont nous recommandons l'utilisation dans tout formalisme de modélisation, qui vont nous permettre la mise en œuvre de notre approche.

L'article s'organise comme suit. Dans la section 2, nous présentons les méthodes classiques de transpositions des concepts objets dans l'univers relationnel. Dans la section 3, nous présentons l'approche que nous proposons ainsi que les règles de transformations que nous avons définies. Dans la section 4, nous décrivons la mise en œuvre de cette approche ainsi que les résultats obtenus concernant la transposition d'un modèle de taille significative. La section 5 présente la conclusion et les perspectives.

2. Approches usuelles de représentation relationnelle des modèles objets

Plusieurs approches ont été proposées pour la représentation des concepts objets dans un environnement relationnel. Nous discuterons essentiellement ici les propositions portant sur la représentation des relations d'héritage[RAH 00][RUM 95][KRO 91] [ELM 00], d'associations et d'agrégations [ERI 04].

2.1. *Représentation de la relation d'héritage*

Les propositions effectuées, peuvent être classées en trois catégories :

1) *Représentation "à plat"* : une table par arborescence.
Dans cette stratégie, tous les attributs de toutes les classes d'une hiérarchie sont stockés dans une seule et même table. Le nom de la racine de la hiérarchie est le plus souvent utilisé pour nommer cette table. Deux autres attributs sont ajoutés. Le premier représente l'identifiant, ou clé primaire, de la table. Le second est généralement un code spécifiant le type effectif de l'objet instancié. Une variante de cette approche consiste à remplacer ce second attribut par plusieurs attributs de type booléen; chacun d'eux correspondants à un des types possibles. Cette approche est très simple et supporte le polymorphisme car tous les objets polymorphes appartiennent à la même table. L'accès aux instances d'un niveau quelconque de la hiérarchie, demande seulement la projection d'une sélection car toutes les instances sont rangées dans une seule table. C'est la seule méthode qui sera économe en jointure. L'inconvénient, par contre, se situe au niveau de l'espace de stockage car les tables peuvent être très grandes et contenir beaucoup (et parfois énormément) de valeurs nulles. De plus, la mise à plat impose parfois de regrouper un très grand nombre de classes disparates, qui n'ont pas d'autres points communs que d'utiliser le même mécanisme ou le même patron de conception (exemple : Modèle-Vue-Contrôleur (MVC)) représentés par héritage de classes abstraites. Les transformations que nous proposons vont précisement permettre de supprimer beaucoup de classes abstraites, rendant, dans le cas qui nous intéresse, cette méthode très performante.

2) *Représentation horizontale* : Une table par classe concrète.
Dans cette approche, une table est créée pour chaque classe concrète. Tous les attributs de la classe concrète et ceux hérités des superclasses de cette dernière constituent les colonnes de la table. A ces colonnes, s'ajoute une clé primaire. Lorsqu'il n'y a qu'un seul niveau de classes concrètes, l'avantage de cette approche est qu'il est facile d'obtenir et de stocker les informations sur les objets existants car toutes les informations sur un objet se retrouvent dans une seule table. Les inconvénients principaux sont (1) qu'une requête générique sur une classe abstraite nécessite des unions de projections des tables des sous-classes, et (2) que le polymorphisme n'est pas supporté de façon aisée car un lien vers une classe abstraite se matérialise par une référence vers des tables différentes selon les instances. De complexes mécanismes d'"aiguillage" (cf. Section 2.2) sont alors nécessaires.

3) *Représentation verticale* : une table par classe.
Dans cette approche, on crée une table pour chaque classe. Chaque table a pour colonnes les attributs définis au niveau de la classe qu'elle représente. Un même identifiant est utilisé comme clé primaire pour toutes les tables. Au niveau des sous-classes, il représente à la fois une clé primaire et une clé étrangère. L'avantage de cette approche est qu'elle représente de façon très adéquate les concepts orientés objet. Le polymorphisme peut se représenter assez aisément par une clé étrangère unique (avec jointure automatique avec les superclasses) parce que nous avons un enregistrement dans la table correspondant à chaque classe auquel un objet appartient. Le principal inconvénient est que pour récupérer ces informations, il faut faire des jointures avec toutes les tables des superclasses, ce qui peut devenir très coûteux en termes de temps de traitement des requêtes en particulier si le modèle comporte des hierarchies assez

profondes ce qui est notre cas.

2.2. *Représentation des relations d'associations, agrégations et compositions*

Pour ce qui concerne les associations, agrégations et compositions, on trouve également dans la littérature, plusieurs propositions [ERI 04][SOU 01] de représentation. Ces propositions sont définies en fonction des cardinalités des relations (1:1, 1:n, n:m) et des propriétés des relations d'agrégations (dépendance, partage, exclusivité, prédominance) [MAG 97]. [SOU 01] énumère toutes les solutions envisageables pour la représentation des relations en relationnel. Ces différentes solutions se résument soit à la création de tables intermédiaires entre les tables des classes participantes à la relation, soit à la déclaration de clés étrangères sur des colonnes de tables. Ces différentes propositions ne s'appliquent que lorsque le choix de représentation de l'héritage est vertical ou à plat. Lorsque seules les classes concrètes sont représentées, il est nécessaire de représenter tout lien par un aiguillage comportant en particulier une colonne qui indique le type effectif de l'instance référencée.

2.3. *Choix d'une méthode de représentation*

Choisir de façon systématique les mêmes règles de transposition pour toutes les classes d'un même modèle présente un avantage important. La correspondance entre les deux modèles pouvant se représenter aisément de façon formelle, cette correspondance peut être utilisée pour générer automatiquement tous les programmes de conversions en utilisant les techniques classiques d'ingénierie de modèles [BER 03].

Malheureusement, l'efficacité relative de chacune des représentations possibles des différents mécanismes objets, et de l'héritage en particulier, dépend étroitement de la configuration locale du modèle. Ainsi, une partie du modèle dans laquelle les classes concrètes n'existent qu'à un niveau, et où il n'existe ni lien ni besoin de requêtes au niveau d'une classe abstraite, sera avantageusement représentée de façon horizontale. Une hiérarchie assez plate, avec peu d'attributs différents dans les feuilles, sera avantageusement mise à plat. Aucune représentation ne donne des temps de traitement acceptables dans tous les cas. Une méthode fréquemment utilisée consiste à définir les correspondances au cas par cas, ce que la plupart des environnements permettent effectivement de faire [HEU 03]. Si elle aboutit souvent à des implémentations efficaces du point de vue traitement, cette méthode présente néanmoins deux inconvénients majeurs.

1) Elle devient très difficile à mettre en pratique lorsque l'on traite des modèles de grande taille. C'est le problème qui s'est posé à nous pour traiter de façon efficace le modèle d'ontologie PLIB [PIE 03] qui comporte 179 classes et 80 types définis.

2) Si la correspondance est définie au cas par cas, les programmes de traitements (et, en particulier, les programmes d'accès au niveau objet, ou de migration d'instances

objet relationnel) doivent être développés de façon ad hoc, ce qui représente un coût significatif.

La démarche proposée dans ce travail vise à réunir les avantages des deux approches. Elle consiste dans une première phase part à utiliser des règles systématiques de transposition, de façon à générer à la fois un modèle objet plus adapté à la représentation relationnelle et la correspondance entre modèles (le "mapping" [BER 03]); puis dans un deuxième temps, à utiliser des règles classiques de transposition objet/relationnel.

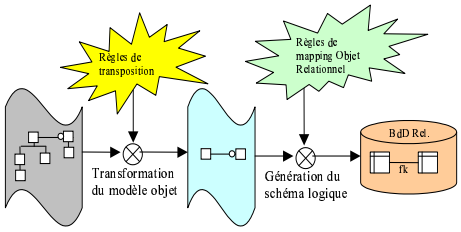

Figure 1. *Notre approche*

3. Approche proposée

3.1. *Principe et objectifs*

L'objectif de notre approche est de proposer un ensemble de règles permettant de transformer un modèle objet en un modèle objet plus simple à représenter de façon efficace dans un univers relationnel. Une règle (r_i) est constituée d'un prédicat (p_i) contrôlant le déclenchement d'une transformation (t_i) du modèle de données. Une première implantation de notre modèle [PIE 05] avait, au paravant, été effectuée avec une classique représentation horizontale. Les temps de réponses resultant des opérations de jointures tant d'héritage que d'associations, etaient presque inacceptables. Notre objectif a donc été de developper une approche évitant systématiquement les jointures, tout en réduisant autant que possible les valeurs nulles. Cette approche se base donc sur une stratégie de représentation d'héritage par "mise à plat". Les règles que nous proposons sont non seulement basées sur la structure des classes (relations et classes) des modèles objets, mais aussi sur les relations et contraintes existant au niveau des sous-ensembles d'instances, telles qu'elles peuvent cependant être exprimées au niveau du modèle initial si l'on s'impose d'exprimer toutes les contraintes implicites de ce modèle. Deux informations de ce type sont exploitées : les redéfinitions du co-domaine[1] des attributs lors d'un héritage[2] (ce qui permet de supprimer

1. un attribut est vue comme une fonction dont le domaine est une classe et le co-domaine l'espace des valeurs
2. la rédéfinition du type d'attribut est implicite en UML. L'attribut est simplement redéclaré avec un nouveau type.

beaucoup de classes abstraites, et donc d'atténuer les défauts de la mise à plat), et le caractère compositionnel d'une relation d'agrégation. Lorsque ces règles ne peuvent plus être appliquées, les fragments des hiérarchies sont mis à plat. Il faut noter que ces transformations se font toujours de façon à conserver le pouvoir d'expression du modèle.

Prédicats et transformations devant être évalués sur le modèle, celui-ci est donc représenté comme instance d'un méta-modèle. Notre implantation est développée dans un environnement EXPRESS [ISO 94], à la fois parce que notre modèle [ISO 04] est exprimé en EXPRESS, et parce que EXPRESS possède un formalisme de contraintes puissant permettant de représenter, en particulier, la plupart des contraintes que nous exploiterons dans nos transformations (les autres seront reprèsentées par des conventions particulières au sein du modèle). Avant de décrire les règles que nous avons définies, nous présentons dans un premier temps le sous-ensemble graphique du langage de modélisation EXPRESS que nous allons utiliser pour illustrer nos règles de transformations.

3.2. *Langage EXPRESS et notations*

Le langage EXPRESS [ISO 94] est le langage de modélisation objet conçu dans le cadre du projet STEP. EXPRESS supporte l'héritage multiple et répété et les attributs de type collection (SET, LIST, ARRAY, BAG). Les attributs peuvent avoir une entité pour co-domaine, de sorte que toutes les associations sont représentées par des attributs. Lorsque nous parlerons désormais d'attributs, cela recouvrira à la fois les attributs et associations des formalismes tels qu'Entité-Association [CHE 76]. Dans ce langage, l'accent a été mis sur la précision du modèle, et tout particulièrement sur les contraintes que doivent respecter les données pour être acceptées comme conformes au modèle. Au langage EXPRESS est associé un langage procedural proche du Pascal qui permet d'exprimer des contraintes et d'assurer la fiabilité de l'information représentée. Le langage offre deux formalismes : un textuel et l'autre sous forme graphique (EXPRESS-G). La figure 2, présente brièvement les quelques éléments graphiques d'EXPRESS-G. Une caractéristique intéressante d'EXPRESS est que, lors d'une spécialisation de classe, le co-domaine d'un attribut peut être restreint. Ceci est appelé redéfinition de type (RT).

Dans le modèle EXPRESS-G de la figure 3a, nous avons quatre entités. Les entités A_1 et B_1 sont sous-classes respectives des entités A et B. L'entité A référence l'entité B par l'attribut $a2b$ de cardinalité 1:1 puisque l'attribut est en train plein. L'entité A_1 redéfinit l'attribut $a2b$ (hérité de la superclasse A) en raffinant son domaine pour qu'il soit désormais du type B_1. L'entité B, définit un attribut inverse ($b2a$) de l'attribut $a2b$. La cardinalité de la relation est [1:n]. Une instance de B peut donc être référencée par 1 ou plusieurs instances de A par l'attribut $a2b$.

Pour simplifier la représentation des règles de transformations que nous allons définir, nous ajoutons deux conventions pour les représentations graphiques :

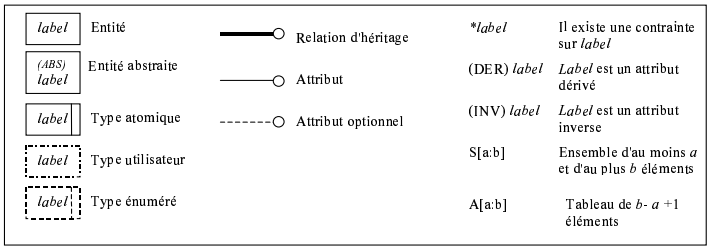

Figure 2. *Les différents éléments graphiques de EXPRESS-G*

Figure 3. *a) Exemple de modèle EXPRESS-G; b) Notation graphique de l'absorption de l'entité A par l'entité B*

– nous omettons de représenter les attributs (ou certains domaines des attributs) d'une entité dans la notation EXPRESS-G si ces derniers n'ont pas d'intérêt dans le contexte où l'on se trouve;

– lorsqu'une entité A est fondue dans (ou "absorbée par") une entité B, on l'illustrera de la façon définie dans la figure 3b;

3.3. *Définition règles de transformations*

Les transformations que nous souhaitons faire sur les modèles de classes visent à éliminer totalement les relations d'héritages et à réduire, autant que possible, les relations d'agrégation entre les classes du modèle. Notre analyse se porte donc sur les points suivants :

– Quelle simplification peut-on effectuer sur les hiérarchies d'héritage d'un modèle objet tout en conservant la possibilité de polymorphisme pour les instances du modèle?

– Comment simplifier les relations d'agrégation /composition entre les différentes classes du modèle tout en conservant sa sémantique?

3.3.1. *Relation d'héritage*

L'héritage permet la généralisation et la spécialisation des classes. Dans une hiérarchie de classes, on peut distinguer des classes abstraites et des classes concrètes. Les classes abstraites ne sont jamais instanciées. Elles servent à factoriser des propriétés ou des comportements. Il existe cependant un cas, relativement fréquent lorsque le modèle initial est bien contraint (i.e., toutes les contraintes existantes ont été repré-

sentées), où une entité abstraite peut être supprimée sans modifier la sémantique du modèle. C'est le cas correspondant à notre règle 1 dite de "*suppression d'abstrait*".

1) **Règle 1** (***Suppression d'abstrait***).

Une classe abstraite sera supprimée :
(1) si elle n'est co-domaine d'aucun attribut pouvant avoir pour domaine (même par polymorphisme) une classe concrète, et
(2) si toutes les classes abstraites qui la référencent peuvent être supprimées.
Dans ce cas, tous ses attributs sont descendus au niveau de ses sous-classes directes.

Notons que cette règle n'est réellement féconde que si, pour chaque attribut hérité dont le co-domaine est une classe, le concepteur a vérifié si le co-domaine de l'attribut pouvait être redéfini, et, si tel était le cas, alors il l'a fait apparaître dans le modèle (cf. figure 4).

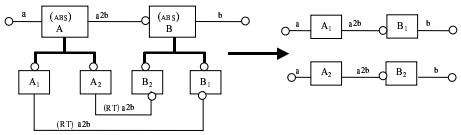

Figure 4. *Exemple d'application de la règle "suppression d'abstrait"*

2) **Règle 2** (***Concrétisation d'abstrait***). La règle 2 complémentaire de la règle 1, vise à concrètiser les classes abstraites que la règle 1 ne permet pas de supprimer en les fusionnant avec toutes ses sous-classes (cf. figure 5):

Une classe abstraite ne doit pas être supprimée si :
(1) elle est co-domaine d'un attribut pouvant avoir pour domaine (éventuellement par polymorphisme) une classe concrète, ou
(2) elle est référencée par une classe abstraite qui ne peut être supprimée.
Dans ce cas, toutes ses sous-classes doivent être remontées et fusionnées à son niveau.

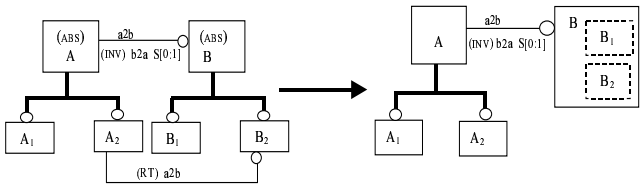

Figure 5. *Exemple d'application de la règle "concrétisation d'abstrait"*

3.3.2. *Relation d'agrégation*

La relation d'agrégation est une relation dans laquelle une entité référence une autre à travers un attribut qui représente une relation tout/partie. A cette relation peuvent être associées une cardinalité minimale et une cardinalité maximale (cardinalité directe). Une relation inverse peut être définit comme pour toute relation. Cette relation inverse peut également être associée à une cardinalité (cardinalité inverse).

Lorsque plusieurs relations d'agrégation sont définies vers la même entité avec des cardinalités directes et inverses [0:1], il est essentiel de savoir au niveau de chaque instance :

– est-ce qu'une entité composante peut, ou non, appartenir à plusieurs composés ?

– est-ce qu'une entité composante peut exister sans appartenir à aucun composé ?

Si la réponse à la première question est négative, une entité composante peut être représentée (sous forme optionnelle, si la cardinalité est 0:1) dans chaque entité composante. Si, de plus, la réponse à la deuxième question est négative, l'entité composée n'a plus besoin d'être représenté en tant que telle dans le modèle de données. Notons que cette contrainte, que nous appelons *used_once* ou *ou exclusif* correspond, en UML 2.0, au cas où chacune des relations d'agrégation est une relation de composition (le losange noir) mais où, de plus, un composant ne peut exister qu'à l'intérieur d'un de ses composés. Ceci définit la règle d'absorption multiple :

1) **Règle 3** (***Absorption multiple***).

(1) Si toutes les références à une classe sont de cardinalités directes et inverses maximum 1, et
(2) si toute instance de cette classe est effectivement liée à une seule instance référençante (contrainte "ou exclusif" ou " used_once "), et
(3) si cette entité n'a pas de sous-type,
alors, cette entité sera absorbée par chacune des entités la référençant.

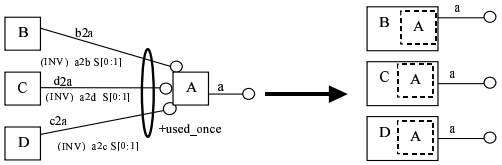

Figure 6. *Exemple d'application de la règle "Absorption multiple"*

2) **Règle 4** (***Absorption simple***). En absence de *ou exclusif*, lorsque des entités sont reliées par une relation de cardinalité directe et inverse 0:1, ces entités peuvent être regroupées, les attributs de l'entité absorbée deviennent optionnels si la cardinalité est 0:1. Les attributs de l'entité absorbante deviennent optionnels si la cardinalité inverse est 0:1.

si une entité A est référencée par une entité B par un attribut de cardinalité directe et inverse maximale 1,
alors cette entité A sera absorbée par B.

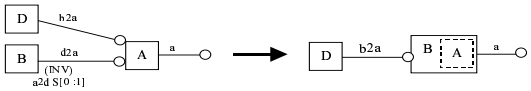

Figure 7. *Exemple d'application de la règle "Absorption simple"*

3.3.3. *règle de mise à plat sans condition*

Règle 5 (***Mise à plat sans condition***). Enfin, si toutes ces règles ont été appliquées répétitivement, qu'aucune ne s'applique plus et qu'il reste des relations d'héritage, l'objectif de suppression de l'héritage revient à la solution de mise à plat classique.

Si aucune des règles précédemment énoncées ne peut s'appliquer à une entité d'une hiérarchie,
alors on met cette dernière à plat : toutes les sous-classes sont remontées et fondues dans la racine.

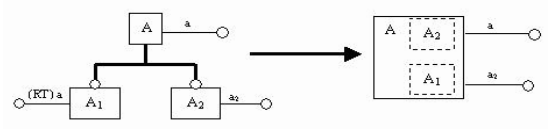

Figure 8. *Exemple d'application de la règle "Mise à plat sans condition"*

3.3.4. *Ordre d'application des règles*

Il convient de noter qu'aucune des règles ne restreint la capacité de représentation de l'information du modèle. Il est donc possible de les appliquer dans n'importe quel ordre. Néanmoins, il est clair que le résultat dépend de l'ordre d'application des règles. Ceci nous amène à définir les priorités suivantes.

1) La règle d'absorption multiple (Règle 3) doit être plus prioritaire que la règle d'absorption simple (Règle 4). En effet, si un modèle vérifie la règle 3, alors il vérifie aussi la règle 4. Si la règle 4 est appliquée avant la règle 3 alors l'entité référencée dans la relation sera absorbée par une seule entité et par application successive de la règle 4, toutes les entités référençantes seront fusionnées en une seule (cf. figure 9). Ceci est clairement moins précis que l'application prioritaire de la règle 3.

2) Les règles concernant les classes abstraites (1,2) sont de même priorité. Mais leur priorité doit être plus faible que celles des règles d'absorption (simple et multiple). En effet, l'application des règles 1 et 2 peut engendrer la perte des liens absorbants ou

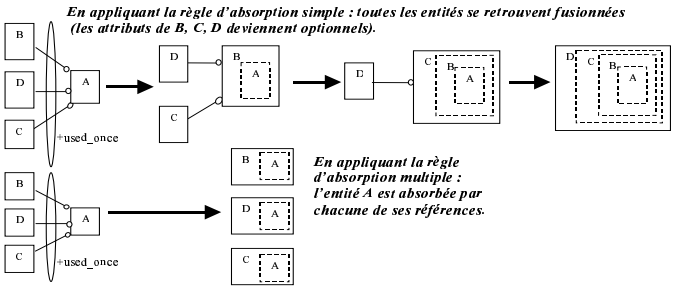

Figure 9. *Priorité entre l'absorption simple et l'absorption multiple*

l'affaiblissement de leurs caractéristiques (surjectivité et optionalité). Ceci augmenterait le pouvoir d'expression du modèle, le rendant par là moins contraint.

3) La règle de mise à plat a la plus faible priorité et ne doit être appliquée que lorsque aucunes des autres règles ne sont applicables.

Soulignons, que la prise en compte des priorités devra se faire de façon répétitive, en activant, à chaque étape, la règle la plus prioritaire activable.

4. Mise en œuvre et résultat

Nous présentons d'abord les résultats obtenus du point de vue de la représentation des données dans un système relationnel (SQLServer 2000), puis la méthode mise en œuvre pour traiter de façon générique tous les problèmes de conversion.

4.1. *Transposition du modèle objet en relationnel : un exemple le modèle PLib*

L'approche proposée dans cet article a été mise en œuvre dans l'environnement ECCO d'EXPRESS [STA 97] analogue à l'architecture MDA d'UML [OMG]. Elle a été appliquée au modèle PLIB [ISO 04]. Il s'agit d'un modèle assez complexe constitué de 179 entités et de 80 types définis. Nous avions déjà essayé auparavant une approche classique sur ce modèle [PIE 05]. Le schéma relationnel résultant était constitué de 519 tables : 179 tables d'entités et 340 tables annexes (types énumérés, aiguillage, associations, etc.). Avec l'approche proposée ici, le modèle transformé contient 49 entités. La base de données générée est alors constituée de 85 tables dont 49 correspondant aux tables d'entités et le reste des tables des types énumérés et des tables d'associations. Nous ne pouvons pas encore présenter de résultat sur l'accélération du traitement des requêtes car l'interpréteur de requêtes est en cours de développement. Le ratio d'amélioration devrait néanmoins d'après nos évaluations être au moins du même ordre de grandeur que celui portant sur le nombre de tables.

4.2. *Génération automatique des traitements de conversion*

En utilisant les techniques classiques d'ingénierie des modèles où une correspondance entre modèles est elle-même représentée comme un modèle [BER 03], tout traitement de conversion peut s'écrire sous forme d'un programme générique ayant pour paramètres le modèle initial, le modèle transformé et le modèle de correspondance. Nous avons donc défini un modèle de transposition permettant de représenter les correspondances entre modèle initial et modèle transformé.

L'application des règles proposées aboutit aux traitements d'une suite de modèles : m_0, m_1, m_2, ..., m_p, dont le premier modèle (m_0) est le modèle objet initial, et le dernier (m_p) est le modèle objet complètement transformé. Tous ces modèles sont représentés en EXPRESS comme instances du même méta-modèle (celui d'EXPRESS en EXPRESS). Le modèle de correspondance est lui-même représenté comme un modèle EXPRESS. Il est donc également géré par le même méta-modèle. Ce modèle de correspondance contient à la fois les entités du modèle initial (*source_entity*) et les entités du modèle résultant de l'application des règles (*computed_entity*). Le modèle final est constitué des entités calculées ainsi que des entités du modèle initial qui n'ont pas disparu lors de l'application d'une règle. Le modèle de correspondance représente donc à la fois les morphismes (m_i, m_{i+1}), et leur composition globale définissant le morphisme (m_0, m_p).

La figure 10 présente, en EXPRESS-G, le modèle de correspondance défini. On notera que les attributs ne sont pas représentés puisque leurs évolutions sont parfaitement définies par chaque triplet <*classe_absorbante*, *classe_absorbée*, *nom_de_règle*>. L'entité *entities* est une entité qui représente toutes les entités du

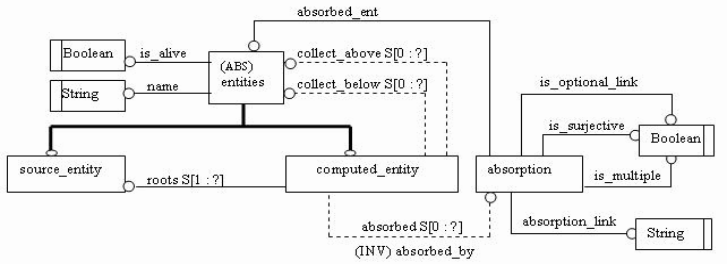

Figure 10. *Modèle de correspondance entre modèles initial et cible.*

modèle et leur état à un moment donné. L'attribut *is_alive* spécifie si l'entité existe encore dans le schéma à un moment donné. Toute transformation élémentaire comporte: une entité "absorbante", dans laquelle une autre entité est fusionnée, donnant lieu à une nouvelle entité. La nouvelle entité est représentée par l'entité *computed_entity*. L'entité absorbante est représentée soit par une entité *source_entity* lors de la première transformation de cette *source_entity*, soit par l'entité *computed_entity* elle-même dans les transformations suivantes. L'entité fusionnée est représentée par les autres attributs de *computed_entity*:

– *collected_above* dans le cas d'une suppression d'abstrait;

– *collected_below* dans le cas d'une concrétisation d'abstrait, ou d'une mise à plat;

– *absorbed* dans le cas d'une absorption.

Tous les attributs ci-dessus étant de type collection, on notera que certaines compositions de morphismes sont directement réalisées au niveau du modèle.

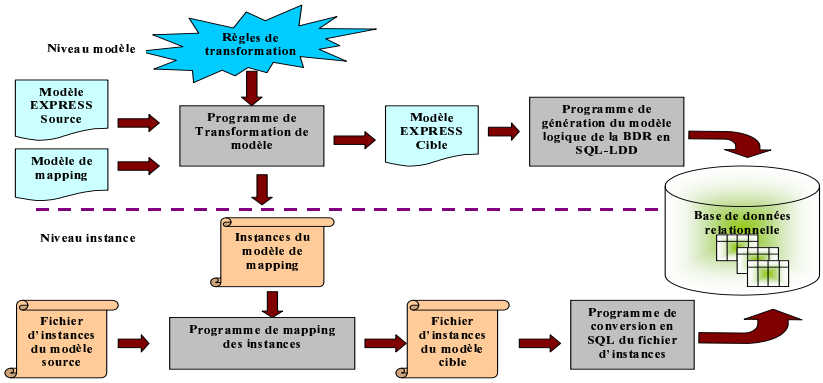

Figure 11. *Transposition de modèles et migration des instances.*

4.3. *Utilisation du modèle de correspondance*

La première utilisation de ce modèle de correspondance est la réalisation du chargement de la base de données relationnelle à partir d'instances du modèle objet initial. Le programme, générique, prend en entrée le modèle initial, le modèle transformé et le modèle de correspondance et accède à deux API d'accès aux instances sources et cibles. Il traite successivement toutes les entités (sources ou calculées) encore actives dans le modèle cible, c'est à dire les instances des entités ayant la valeur *vrai* pour l'attribut *is_alive*. Les *computed_entity* actives sont utilisées pour traiter les instances de leurs entités racines (*roots*), et pour chacune de celles ci, les instances qu'elles ont absorbées (récursivement) par l'une des règles représentées par les attributs de l'entité *computed_entity*. Les entités sources encore actives ne font l'objet d'aucune transformation. Elles sont ensuite transférées à l'identique.

Le modèle cible, bien qu'ayant une capacité de représentation de l'information au moins égale à celle du modèle initial, est par contre beaucoup moins contraint que celui-ci. Pour pallier à ce problème, on peut, en utilisant le modèle de correspondance, coder dans les programmes d'implementation des APIs, générés automatiquement, les contraintes non représentées du modèle pour assurer l'intégrité des données de la base.

Le modèle de correspondance peut être exploité également pour générer des interfaces (API) au niveau du modèle initial pour accéder au modèle cible. Ceci permet de faire fonctionner les applications pré-existantes sur le modèle initial.

Le schéma de la figure 11 représente l'architecture globale de la mise en œuvre effectuée.

5. Conclusion

Nous avons présenté dans cet article, une approche de transposition de modèles qui vise à améliorer l'efficacité de la représentation relationnelle d'un modèle objet. L'approche consiste à définir un ensemble de règles de transformations qui, dès lors qu'elles sont vérifiées, provoquent l'exécution d'une opération de transposition du modèle objet jusqu'à arriver à un modèle très facile à implémenter dans un univers relationnel. Ces règles sont basées sur l'analyse fine des structures d'instances d'un modèle objet. En effet, une étude détaillée de ces structures nous a permis de constater qu'il était possible de les modéliser par des contraintes, puis de les exploiter pour simplifier certains éléments d'un modèle objet. Les contraintes d'instances dont nous avons mis en évidence l'intérêt (*used_once* et *redéfinition d'attributs*) ne sont malheureusement pas toujours exprimables dans les formalismes de modélisation conceptuelle usuels. Leur utilisation dans la modélisation objet pourrait pourtant être judicieusement exploitée pour définir des modèles objets plus finement contraints. Cet article plaide pour leur introduction dans les formalismes où elles n'existent pas. En tout état de cause, une convention est toujours possible pour représenter leur existence.

L'intérêt de contraindre très finement un modèle objet est que cela rend possible la définition de règles à la fois systématiques et efficaces de transpositions dans les univers relationnels ou relationnels objets. Le caractère systématique des règles permet de générer automatiquement à la fois le modèle de cible et le modèle de correspondance. Les techniques d'ingénierie des modèles permettent alors d'écrire des programmes génériques pour toutes les opérations de traduction d'un modèle à l'autre. La mise en œuvre de notre approche sur un exemple nous a permis de diviser par un facteur de six, le nombre de tables du schéma relationnel correspondant à un modèle objet de complexité significative.

Du point de vue perspectives, d'autres contraintes de niveau instances restent, sans aucun doute, à identifier. Cela permettrait d'améliorer encore les règles de transposition. D'autre part, la même approche peut être utilisée pour d'autres transpositions de modèles. Nous sommes en train de travailler sur la définition de règles de transposition de nos schémas objets vers des structures XML. Ceci faciliterait la description des documents XML lorsque le modèle de classes d'origine possède des hiérarchies profondes.

6. Bibliographie

[BER 03] BERNSTEIN P., « Applying model management to classical meta data problems », *In Proceedings of the Conf. on Innovative Database Research (CIDR'03)*, 2003.

[CHE 76] CHEN P., « The Entity Relationship Model - Toward a Unified View of Data », *ACM Transactions on Database Systems*, vol. 1(1), March 1976.

[ELM 00] ELMASRI R., NAVATHE S., *Fundamentals of Database Systems (3rd ed. ed.)*, Addison Wesley, 2000.

[ERI 04] ERIC P., RAHAYU J., TANIAR D., « Mapping Methods and Query for Aggregation and Association in Object-Relational Database using Collection », *In Proc. of the International Conference on Information Technology(ITCC'04)*, IEEE Computer Society, 2004.

[HEU 03] HEUDECKER N., « Introduction to Hibernate », *http://www.systemmobile.com/articles/IntroductionToHibernate.html*, , 2003.

[ISO 94] ISO10303-11, « Industrial automation systems and integration - Product data representation and exchange - Part 11 : Description methods : The EXPRESS language reference manual », rapport, 1994, International Standardization Organization, Genève.

[ISO 99] ISO9075, « Information Technology Database Language SQL », rapport, 1999, International Standardization Organization, Genève.

[ISO 04] ISO13584-25, « Industrial automation systems and integration - Parts library - Part 25 : Logical resource: Logical model of supplier library with aggregate values and explicit content », rapport, 2004, International Standardization Organization, Genève.

[KLA 04] KLAENE M., « Using iBatis SQL Maps for Java Data Access », *Resources for Java server-side developers*, vol. 1, 2004.

[KRO 91] KROENKE D., *Object Oriented Modelling and Design*, Prentice-Hall, 1991.

[LIU 91] LIU C., LAYLAND J., « Third Generation Database System Manifesto », *In Computer Standards and Interfaces*, , 1991, p. 41–54.

[MAG 97] MAGNAN M., OUSSALAH C., *Ingenierie Objets : Chapitre II : Objets et composition*, InterEditions, mai 1997.

[OMG] OMG, « Model Driven Architecture », *http://www.omg.org/mda/*.

[PIE 03] PIERRA G., « Context-Explication in Conceptual Ontologies: The PLIB Approach », JARDIM-GONÇALVES R., CHA J., STEIGER-GARÇAO A., Eds., *In Proc. of CE'2003, Proc. of Concurrent Engineering (CE'2003)*, July 2003, p. 243-254.

[PIE 05] PIERRA G., DEHAINSALA H., AIT-AMEUR Y., BELLATRECHE L., « Base de Données à Base Ontologique : principe et mise en oeuvre », *To Appear in Ingénierie des Systèmes d'Information (ISI)*, 2005.

[RAH 00] RAHARU J., CHANG E., DILLON T., TANIAR D., « A methology for transforming inheritance relationships in an object-oriented conceptual model tables », *Information and Software Technology*, vol. 42, 2000, p. 571–592.

[RAH 01] RAHAYU J., CHANG E., DILLON T., TANIAR D., « Performance evaluation of the object-relational transformation methodology », *Data Knowl. Eng.*, vol. 38, n° 3, 2001, p. 265–300, Elsevier Science Publishers B. V.

[RUM 95] RUMBAUGH J., BLAHA M., PREMERLANI W., EDDY F., LORENSEN W., *Database Processing Fundamentals, Design, and Implementation*, Prentice-Hall International Editions, 1995.

[SOU 01] SOUTOU C., « Modeling relationships in object-relational databases », *Data and Knowledge Engineering*, vol. 36, n° 1, 2001, p. 79–107.

[STA 97] STAUB G., MAIER M., « ECCO Tool Kit - Programmer's Guides », *User Manual*, , 1997.