Actes des 20 èmes Journées Bases de Données Avancées (BDA 2004), pp263-286

Base de Données à Base Ontologique : le modèle OntoDB

Guy PIERRA and Hondjack DEHAINSALA and Yamine AIT AMEUR and Ladjel BELLATRECHE and Jérôme CHOHON and Mourad EL-HADJ MIMOUNE.

Laboratoire Informatique, Scientifique et Industriel (LISI),
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique (ENSMA),
Téléport 2, 1 avenue Clément Ader, BP 40109, 86960 FUTUROSCOPE Cedex, FRANCE
Tel : +33 5 49 49 80 60 Fax : +33 5 49 49 80 64
Email : {pierra, hondjack, yamine, bellatreche, chochon, mimoune}@ensma.fr

Résumé

Nous proposons dans cet article un nouveau modèle de base de données appelé *base de données à base ontologique* (BDBO) qui présente deux caractéristiques. D'une part, il permet de gérer à la fois des ontologies et des données. D'autres part, il permet d'associer à chaque donnée le concept ontologique qui en définit le sens. Les ontologies considérées sont celles qui peuvent s'exprimer sous forme de modèle, au sens de Bernstein, c'est à dire d'un ensemble d'objets accessibles à partir d'un objet racine, et qui décrivent des univers de classes et de propriétés. Les données considérées sont celles qui représentent des instances des classes de l'ontologie, et dont on exige qu'elles appartiennent à une seule classe de la base (borne inférieure des classes d'appartenance, au sens de la relation de subsomption). On décrit de façon détaillée un modèle d'architecture de BDBO, appelé OntoDB, et l'on présente brièvement son implantation dans un environnement constitué du système PostgreSQL, du modèle d'ontologie PLIB et du langage EXPRESS.

Mots clés : Base de données, ontologie, PLIB, Base de données à base ontologique, BDBOO

1.Introduction

Les travaux menés dans le domaine des bases de données (BD) dans la dernière décennies ont surtout visés à augmenter les capacités de représentation de l'information. Ont ainsi été proposées, après le modèle relationnel, les BDs déductives, actives, objet et relationnel-objet. Ces travaux n'ont, par contre, guère contribué à réduire l'hétérogénéité entre BDs portant sur un même domaine. Ceci rend l'intégration de sources de données hétérogènes toujours aussi difficile.

Parallèlement, les travaux menés en modélisation des connaissances faisaient émerger la notion d'ontologie comme un modèle conceptuel consensuel et partageable d'un domaine [Gruber, 93]. Contrairement au modèle conceptuel qui *prescrit* les informations qui doivent être représentées dans une base de données pour répondre à un cahier des charges applicatif, une ontologie vise à *décrire* de façon *consensuelle l'ensemble des informations* permettant de conceptualiser des domaines d'application assez large. Des langages de définitions ontologies ont été développés. Ils permettent la représentation et l'échange informatique à la fois d'ontologies, et d'objets définis en termes de ces ontologies. De plus, un nombre croissant d'ontologies a pu être développé et faire l'objet de consensus dans des communautés plus ou moins large.

Dans les domaines où il existe une ontologie, il est possible d'exprimer les différents modèles conceptuels correspondant à différents cahiers des charges applicatifs en termes de sous-ensembles ou de spécialisation de cette ontologie. La représentation explicite de cette ontologie au sein de chaque BD permet alors (1) leur intégration plus facile, voire automatique, même si leurs modèles conceptuels sont différents [Bellatreche et al, 03], (2) une génération automatique d'interfaces d'accès au niveau connaissance, c'est à dire au niveau ontologique, pour les données contenues dans chacune des bases. C'est le concept de *Bases de Données à Base Ontologique* (BDBO) que nous proposons et qui possède deux caractéristiques : (1) ontologie et données sont toutes deux représentées dans la BD et peuvent faire l'objet des mêmes traitements (insertion, mise à jour, requêtes, etc.) ; (2) toute donnée est associée à un élément ontologique qui en définit le sens. Le modèle d'architecture que nous proposons pour valider le concept de BDBO est constituée de quatre parties. Aux deux parties traditionnelles des BDs: *méta-base* (qui contient les schémas) et *données*, s'ajoutent, d'une part, *l'ontologie* qui permet de stocker les différentes ontologies des domaines couverts par la BD, et, d'autre part, le *méta-schéma* qui contient un méta-modèle réflexif du modèle d'ontologie utilisé, et qui permet de rendre générique tout traitement sur les ontologies. Cette architecture très flexible, analogue au MOF et appelée *OntoDB*, permet de s'adapter aisément à des évolutions du modèle d'ontologie utilisé. Un prototype de BDBO réalisé sur la base du SGBD PostgreSQL, et utilisant des ontologies PLIB dans un environnement basé sur EXPRESS, est également présenté dans cet article.

Le contenu de cet article est le suivant. Dans la section 2, nous décrivons brièvement le contexte de cette étude. Il s'agit de représenter sous une forme partageable, échangeable et intégrable des instances de classes d'objets techniques constituant des bases de données ou des catalogues de composants industriels. On présente à la fois le modèle d'ontologie utilisé, et le modèle de description d'instances en termes de l'ontologie. La section 3 décrit le modèle d'architecture que nous proposons pour une BDBO, appelé *OntoDB*. La section 4 présente brièvement la mise en œuvre que nous avons réalisée. On discute en conclusion de quelques perspectives ouvertes et les travaux en cours.

2. Contexte de l'étude

Nous présentons ici notre domaine d'étude, ainsi que les modèles développés pour en formaliser les ontologies et en représenter les instances.

2.1. Notre domaine d'application cible

Notre domaine d'application cible est le commerce électronique professionnel et l'échange de données techniques dans le domaine des composants industriels. Il s'agit, d'une part, de pouvoir rechercher de façon entièrement automatique quel fournisseur présent sur le Web est susceptible de fournir un roulement à billes ayant des caractéristiques techniques données (par exemple, supporter une charge radiale de 100 Newton et axiale de 6 Newton avec une durée de vie de 2000 H en tournant à 500 t/s). Et ceci quelque soit la structure particulière du "catalogue" susceptible de le contenir. Il s'agit, d'autre part, de pouvoir intégrer automatiquement les catalogues de différents fournisseurs au sein d'une base de données d'entreprise utilisatrice, en offrant, sans aucune programmation, des possibilités d'accès ergonomiques. Notons que, dans les domaines techniques, les notions d'interchangeabilité et de normalisation sont très développées. Un vocabulaire technique consensuel existe donc déjà, de façon informelle, pour les termes essentiels de chaque domaine. L'approche que nous avons suivi a donc consisté, d'abord, à formaliser ce langage consensuel sous forme d'ontologie, puis à exploiter ces ontologies pour résoudre les deux problèmes posés.

Dans un premier temps, tout au long des années 90, nous avons développés un modèle d'ontologie adapté au domaine [ISO 13584-42, 98], ainsi qu'un modèle d'échange d'instances d'entités décrites en termes de ces ontologies. Ceci constituait le projet PLIB dont les résultats ont en particulier donné lieu à un ensemble de normes ISO dans la série 13584 (Parts Library) dont le dernier document est actuellement sous-presse [ISO 13584-25, 04]. Le modèle étant terminé, nous avons lancé depuis début 2001 le projet OntoDB dont l'objectif est de permettre la gestion, l'échange, l'intégration et l'interrogation de données structurées associées a des ontologies formelles, que celles-ci soient de type PLIB, ou d'autres type similaire

(OWL, DAML+OIL). C'est dans ce contexte qu'a été développée notre notion de BDBO. Notre prototype étant basé sur des ontologies et des modèles d'échanges PLIB, nous décrivons ceux-ci brièvement ci-dessous.

2.2. Le modèle d'ontologie PLIB

A la différence des ontologies basées sur des logiques de descriptions qui utilisent intensivement les expressions de classes pour représenter des équivalences terminologiques, le modèle d'ontologie PLIB vise à décrire l'ensemble des entités, supposées consensuelles, existant dans un domaine, par l'intermédiaire de propriétés permettant de caractériser toutes les entités du domaine. Certaines propriétés n'ayant de sens que pour un sous-ensemble des objets du domaine, des classes sont introduites pour autant qu'elles soient nécessaires pour définir le domaine de certaines propriétés. Chaque propriété est alors définie dans le domaine d'une classe d'entités, et n'a de sens que pour cette classe et ses éventuelles sous-classes. Cette approche, dite "orienté-propriété", à pour objectif de s'affranchir autant que faire se peut du caractère contextuel des classifications. Peu de classes existent donc par comparaison aux ontologies basées sur les logiques terminologiques. Inversement, une propriété peut être définie dans le contexte d'une classe même si elle ne s'applique pas à toutes ses instances ou sous-classes. La seule condition est qu'elle soit définissable de façon non ambiguë dans le contexte de cette classe.

Les hiérarchies des ontologies partagées PLIB sont toujours extrêmement "plates". Elles ne définissent pas tous les termes possibles existant dans un domaine, mais au contraire elles visent à définir un vocabulaire canonique minimal. Ce vocabulaire doit seulement permettre de décrire d'une façon unique, par une classe d'appartenance et par un ensemble de valeurs de propriétés, toutes les entités qui font l'objet d'une compréhension commune par les experts d'un domaine. Toute entité existant dans un domaine peut donc se décrire, soit directement en termes de l'ontologie partagée, soit en lui rajoutant des classes et/ou propriétés supplémentaires. Le modèle d'ontologie PLIB offre pour cela une relation d'extension particulière, appelée *case-of* (est-un-cas-de). Cette relation permet à un utilisateur de définir sa propre ontologie à partir d'une ontologie partagée et de décrire explicitement la correspondance ("mapping" [Bernstein et al., 00]) existant entre ces deux ontologies. Une classe *case-of* d'une autre classe est subsumée par celle-ci. Elle peut importer de l'ontologie partagée un sous-ensemble quelconque des propriétés qui y sont définies. Ceci permet de définir, à partir d'une même ontologie partagée, des ontologies utilisateurs de structures très différentes.

Formellement (voir [Pierra, 03] pour un modèle plus complet), une ontologie PLIB peut être définie comme un quadruplet :

O :< C, P, Sub, Applic>, avec:

– C : l'ensemble des classes utilisées pour décrire les concepts d'un domaine donné (comme les service de voyages, les pannes des équipements, les composants électroniques, etc.). Chaque classe est associée à un identifiant universel globalement unique (GUI).
– P : l'ensemble des propriétés utilisées pour décrire les instances de l'ensemble des classes C. Nous supposons que P définit toutes les propriétés susceptibles d'être présentes dans une base de données. Chaque propriété est associée à un identifiant universel globalement unique (GUI).
– Sub : $C \rightarrow 2^C$ est la relation de subsomption[1] (*is-a* et *is-case-of*) qui, à chaque classe c_i de l'ontologie, associe ses classes subsumées directes[2]. Sub définit un ordre partiel sur C .
– Applic : $C \rightarrow 2^P$, associe à chaque classe de l'ontologie les propriétés qui sont applicables pour chaque instance de cette classe. Les propriétés qui sont applicables sont héritées à travers la relation *is-a* et peuvent être importées de façon explicite à travers la relation de *case-of*.

Soulignons que les définitions ontologiques sont intentionnelles. Le fait qu'une propriété soit applicable pour une classe signifie qu'elle est rigide [Guarino, 02][Pierra, 04] c'est-à-dire essentielle pour chaque instance de la classe. Cela ne signifie pas qu'une valeur sera explicitement représentée pour chaque instance dans la base de données. Dans notre approche, le choix des propriétés effectivement représentées est fait au niveau du schéma parmi les propriétés applicables.

Le modèle d'ontologie PLIB est lui-même défini sous forme d'un schéma décrit dans le langage de spécification de données EXPRESS [Schenck et al, 94]. Une ontologie particulière s'échange donc sous forme d'un fichier d'instances. La figure 8 ci-après montre un exemple d'ontologie PLIB. Celle-ci définit l'ensemble des catégories de composants d'un certain domaine et les propriétés techniques les décrivant (leur nombre est, en général, très important). Tout composant particulier peut être défini (éventuellement par spécialisation) à partir de cette ontologie. Notons que cette représentation est générée automatiquement à partir des données décrivant l'ontologie, ce qui montre les différents éléments d'information représentés dans une ontologie PLIB.

2.3. Représentation d'instances décrites en termes d'une ontologie PLIB

De façon analogue à une instance OEM dans le projet TSIMMIS [Chawathe et al, 95], mais à la différence des individus dans les systèmes de classification ou dans les logiques terminologiques, toute instance conforme à une quelconque ontologie PLIB possède une *classe de base*. Cette classe est la classe la plus spécialisée à laquelle l'instance appartient et dont le modèle PLIB impose l'unicité. Les instances

[1] 2^C désigne l'ensemble des parties de C
[2] C_1 subsume C_2 si et seulement si $\forall x \in C_2$, $x \in C_1$.

peuvent donc être représentées de façon générique sous forme d'un autre fichier d'instances. Une instance PLIB consiste en :

- Un identifiant d'objet ;
- une référence à la classe de base de l'objet (par l'intermédiaire de l'identifiant universel de la classe) ;
- une liste de couples :
  - Référence à une propriété applicable à la classe (par son identifiant universel).
  - Valeur de la propriété (type simple, identifiant d'objet ou collection).

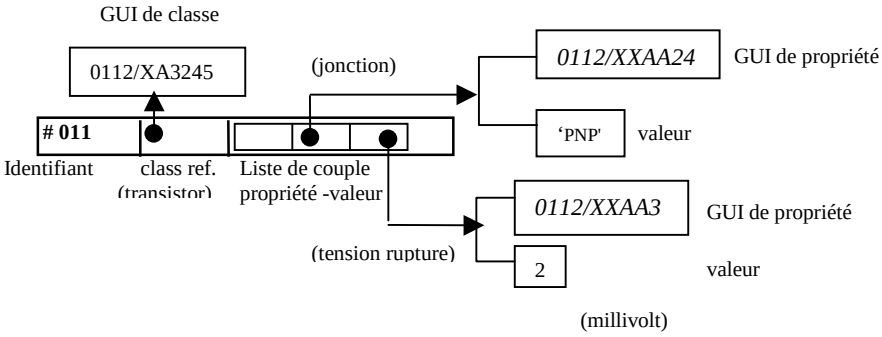


Figure 1. *Exemple d'une instance décrite en termes d'une ontologie PLIB*

2.4 Objectifs

Les objectifs que nous fixons sont au nombre de trois :

1. nous souhaitons pouvoir intégrer automatiquement, et gérer de façon homogène, des populations d'instances dont les données, le schéma et l'ontologie sont chargés dynamiquement ;
2. nous souhaitons que notre système puisse s'adapter aisément soit à des évolutions du modèle d'ontologie utilisé, soit au changement de modèle d'ontologie ;
3. nous souhaitons offrir des accès génériques tant aux ontologies qu'aux instances, en permettant à l'utilisateur de faire abstraction de l'implémentation particulière sur un SGBD particulier (relationnel, relationnel objet ou objet par exemple).

3. Modèle d'architecture proposé : OntoDB

Notre modèle d'architecture, appelé OntoDB vise à atteindre les trois objectifs fixés. Il définit séparément l'implémentation de l'ontologie (Onto) et celle des données (DB). Nous présentons dans les sections suivantes le modèle que nous proposons pour gérer ces deux classes d'informations, puis l'architecture globale qui en résulte.

3.1 Représentation des ontologies

Les ontologies auxquelles nous nous intéressons sont celles susceptibles d'être représentées sous forme d'un modèle au sens de Bernstein [Bernstein et al, 00], c'est à dire d'un ensemble d'objets accessibles à partir d'un objet racine par des relations de connexité particulière, dite de composition. Cette définition correspond à la plupart des modèles d'ontologies récents tels que OIL [Fensel et al, 00], OWL [Broekstra et al, 02], PLIB [ISO 13584-42, 98] [ISO 13584-25, 04] [Pierra, 03]. Une telle ontologie est donc représentée comme instance d'un schéma objet (souvent appelé méta-modèle) dans un formalisme de modélisation particulier (XML-Schema pour OIL et OWL, EXPRESS pour PLIB). Cette représentation, à son tour, fournit un format d'échange pour les ontologies visées (document XML pour OWL, fichier physique d'instances EXPRESS pour PLIB).

3.1.1 Identification des besoins

Les fonctions dont nous souhaitons doter les systèmes incluent les suivantes :

- **F1** : capacité de stockage interne des ontologies au sein d'un schéma logique adapté au SGBD cible;

  Exemple 3.1 : la figure 2 (b) donne un exemple simplifié de modèle d'ontologie. La figure 2 (c) représenté sous forme d'instances une ontologie particulière représentée en UML dans la figure 2(a).

- **F2** : capacité de lire les ontologies représentées dans leur format d'échange et de les stocker dans la BDBO, et, inversement, d'exporter des ontologies ;
- **F3** : interface générique (i.e. indépendante du schéma d'ontologie) d'accès par programme aux entités définissant une ontologie, et ce indépendamment du schéma logique de la base de données (une telle interface générique, c'est à dire pouvant être utilisée quelque soit le schéma d'ontologie, est usuellement appelée "API à liaison différée").

  Exemple 3.2 : En notant :

– *oid* : un type chaîne (dont le contenu est un identifiant d'objet pour la base de donnée;

– *nom_entité* : un type chaîne (dont le contenu est un nom de type d'entité);

– *nom_attribut* : un type chaîne (dont le contenu est un nom d'attribut);

– *nom_association* : un type chaîne (dont le contenu est le nom d'une association);

les fonctions d'une API générique pourraient, par exemple, se définir comme suit :

– `get_association : oid x nom_entite x nom_association -> oid` // association 1-1

– `get_associations : oid x nom_entite x nom_association -> oid`^{n} // association 1-N

– `iterateur : oid`^{n} `x integer -> oid`

– `get_final_type : oid x nom_entite -> nom_entite` // type_dynamique

– `get_integer_attribute :oid x nom_entite x nom_attribut -> integer` // attribut simple

– `get_integer_attributes : oid x nom_entite x nom_attribut -> integer`^{n} // attribut collection

– `get_string_attribute :oid x nom_entite x nom_attribut -> string`

– `get_string_attributes : oid x nom_entite x nom_attribut -> string`^{n}

– …

En prenant le schéma d'ontologie défini à la figure 2, l'écriture du type de la première propriété de l'instance "*id_person*" représentant la classe personne pourrait s'écrire (*set_oid* etant de type *oid*^{n}):

```
   set_oid :=      get_associations      ("id_person",
"class","properties")
IF    get_final_type    (iterateur    (set_oid,1),"property")    =
"data_property"
THEN write (get_string_attribut(iterateur(set_oid,1)
                    , "data_property" ,"value_type"))) ;
END IF ;
```

```
IF…
```

– **F4** : interface d'accès par programme aux entités de l'ontologie spécifique à la fois du modèle d'ontologie et du langage de programmation (une telle interface, qui permet donc de contrôler la correction syntaxique des appels, est usuellement appelée "API à liaison préalable").

Exemple 3.3 : En notant :

– *oid*: un type chaîne (dont le contenu est un identifiant d'objet pour la base de donnée);

– *enumeration_nom_entité* : un type énuméré décrivant les noms d'entités;

les fonctions de l'API spécifique, dépendant du schéma d'ontologie et permettant donc de vérifier la correction des appels, pourraient par exemple se définir en Java en utilisant que des méthodes statiques :

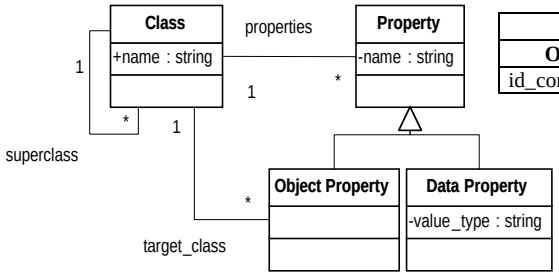
– `Class.properties : oid -> oid[]`
– `data_properties.value_type : oid -> string`
– `get_final_type : oid x enumeration_nom_entité-> enumeration_nom_entité`
– `…`

Avec cette API, l'écriture du type de la première propriété de l'instance "*id_person*" représentant la classe personne pourrait s'écrire (*set_oid* etant de type *oid[]*) :

```
set_oid = class.properties (id_person) ;
IF (get_final _type (set_oid[1],property) == data_property)
THEN                                    System.out.println
(data_properties.value_type(set_oid[1]) ;
END IF ;
IF…
```

Notons que nos deux exemples d'API permettent bien de faire abstraction du schéma logique particulier choisi pour représenter les ontologies dans un SGBD support.

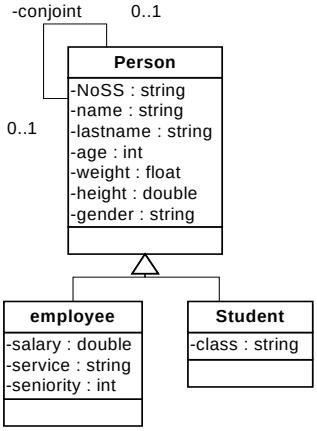
(b)



Object_property				
OID	**name**	**InvProperties**	**target_class**	**…**
id_conjoint	conjoint	id_person	id_person	

class			
OID	**name**	**superclass**	**…**
id_person	person		
id_student	student	id_person	
id_employee	Employee	id_person	

(c)



data_property				
OID	**name**	**InvProperties**	**value_type**	**…**
id_noSS	noSS	id_person	String	
id_name	name	id_person	String	
id_lastname	lastname	id_person	String	
id_age	age	id_person	Int	
id_weight	weight	id_person	Float	
id_height	height	id_person	Float	
id_gender	gender	id_person	char	
id_service	service	id_empoyee	String	
id_salary	salary	id_empoyee	Float	
id_seniority	seniority	id_empoyee	Int	
id_class	class	id_student	String	

Figure 2 exemple d'ontologie (a), son modèle (b) et représentation comme instance de son modèle (c)

3.1.2 Analyse des solutions existantes

Les modèles d'ontologies considérés étant des modèles objets, le choix d'une représentation correspond au problème classique de la représentation de modèles objets au sein d'une BD non forcement de type objet. La solution en est plus ou moins naturelle selon que le SGBD cible est de type objet, relationnel ou relationnel objet [Stonebraker, 90]. Dans la littérature, on trouve les nombreux travaux sur la représentation des modèles objets au sein de SGBDs de type différents. Toutes les approches consistent d'abord à définir des règles de correspondance entre les concepts du modèle objet considéré et les concepts du SGBD cible [Chung et al, 95][Raharu et al, 00][Stonebraker, 90][Blaha et al, 94]. Selon la mise en œuvre, on peut alors distinguer deux approches :

1- la première approche consiste à appliquer cette correspondance générique (manuellement ou automatiquement) au modèle objet particulier afin de créer le modèle logique du SGBD cible pour stocker les instances du modèle objet. Les applications sont alors écrites en fonctions de ce modèle.

2- la deuxième approche [Chung et al, 95][Sutherland, 93][Stoimenov et al, 99] consiste, en plus, à représenter le modèle objet lui-même (classes, propriétés,

relations, héritages, etc.) dans la BD, sous forme d'instances d'un méta-modèle objet (méta-schéma).

L'intérêt principal de la deuxième approche pour notre problème réside dans la possibilité de rendre générique, par rapport au modèle d'ontologie particulier, les applications développées pour réaliser un système de gestion de BDBO.

En effet, rendre générique une application par rapport à un ensemble de modèles exige deux conditions. Si (1) on peut définir un méta-modèle tel que tout modèle d'ontologie envisageable puisse être représenté comme instance de ce méta-modèle, et (2) certains traitements à effectuer peuvent s'exprimer par un programme générique réalisant le traitement pour toute instance de ce méta-modèle, alors tous les traitements définis au niveau méta-modèle pourront être réutilisés à l'identique lors de chaque modification du modèle d'ontologie. Or, dans notre cas, les deux conditions s'appliquent : (1) les modèles d'ontologies qui nous intéressent s'expriment tous dans un formalisme qui dispose de son propre méta-modèle; l'utilisation ce méta-modèle permettra donc toutes les modifications possible du modèle d'ontologie considéré, à la condition que celui-ci continue à s'exprimer dans le même formalisme (par exemple XML-Schema pour OIL et OWL; EXPRESS pour PLIB); (2) les quatre fonctions identifiées comme nécessaires peuvent s'exprimer de façon générique comme montré dans le tableau ci-dessous.

Fonction	**Sans représentation explicite du méta-modèle**	**Avec représentation explicite du méta-modèle**
F1 - Définition du modèle logique de représentation des ontologies.	Définition manuelle du schéma (ou génération par programme externe)	– Définition de règles d'implantation (R1) pour tout instance du meta-modèle (exemple: tout schéma XML pour OWL, ou tout modèle EXPRESS pour PLIB). – Programme de génération qui interprète les instances du méta-modèle correspondant au modèle courant et met en œuvre (R1).
F2 - Lecture / écriture de fichier	Ecriture d'un programme spécifique	– Des règles de génériques (R2) de représentation externes existent déjà (par exemple: document XML pour OWL, fichier d'instances EXPRESS pour PLIB) – Programme générique qui interprète les instances du méta-modèle correspondant au modèle courant en exploitant (R1) et (R2)
F3 - API à liaison différée	Tout le modèle doit être codé dans l'API (cf. exemple 3.4)	L'API devient un interprète des instances du méta-modèle (cf. exemple 3.4) en exploitant (R1)

F4 - API à liaison préalable	Toute l'API doit être codée à la main	– Définition de règles génériques (R3) de représentation d'un modèle EXPRESS ou d'un schéma XML dans le langage cible (Par exemple java). – Programme de génération interprétant le méta-modèle en exploitant (R1) et (R3)

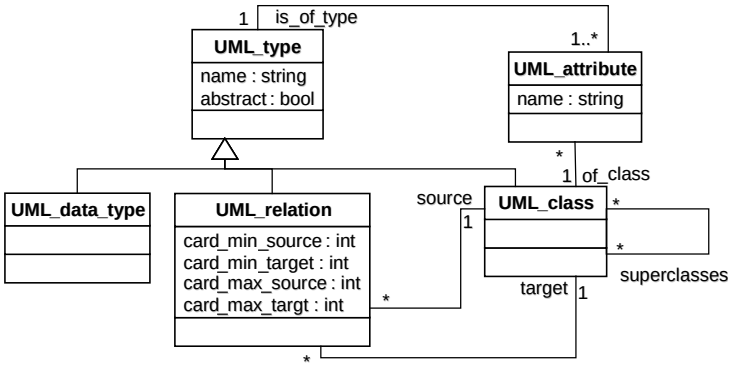
Table 1: implantation des fonctions nécessaires, avec et sans méta-schéma.

Si le méta-modèle de l'ontologie est représenté, comme dans la figure 3, on voit aisément que chacune des fonctions ci-dessus peut être programmée de façon générique en tenant compte, dans tous les cas, de la correspondance (R1) choisie.

Exemple 3.4. Si le méta-modèle d'ontologie n'est pas représenté, il n'existe aucune manière d'interpréter la fonction `get_associations(id_person,"class","properties")` autre que de représenter, à l'intérieur de la programmation de l'API, son résultat[3] :

```
SELECT ID FROM DATA_PROPERTY
     WHERE invProperties = id_person
UNION
SELECT ID FROM OBJECT_PROPERTY
WHERE invProperties = id_person
```

(1)



(a)

[3] On suppose que les règles d'implantation (R1) retenues consistent à ne pas représenter les classes abstraites. D'autres règles (R1) aboutiraient à la même conclusion.

UML_relation							
OID	**name**	**source**	**target**	**s_min**	**s_max**	**t_min**	**t_max**
8	properties	1	2	1	1	0	null
9	superclass	2	1	0	1	0	null
21	target_class	4	1	1	1	0	null

UML_data_type	
OID	**name**
10	String
11	String
12	String

UML_class			
OID	**name**	**superclasses**	**abstract**
1	class		false
2	property		true
3	data_property	2	false
4	object_property	2	false

UML_attribute				
OID	**name**	**of_class**	**is_of_type**	**…**
5	name	1	10	
6	name	2	11	
7	value_type	3	12	

(b)

Figure 3 : *un exemple de méta-modèle d'ontologie (a) et son utilisation pour représenter le modèle de la figure 2b (b).*

Exemple 3.5 : Si le méta-schéma est représenté comme dans la figure 3, l'interprétation de l'appel `get_associations(id_person,"class","properties")` amène à rechercher l'entité *UML_class* de nom "*class*" puis la *UML_relation* de nom "*properties*" admettant cette classe comme source. La cardinalité étant 1-N, cet lien est à chercher sous forme de clé étrangère dans la classe cible, soit "*property*" et avec le nom *inv<nom>* (soit *invproperties*). La classe "*property*" étant elle-même abstraite, ce lien doit être cherché dans toutes ses sous classes non abstraites, soit les tables DATA_PROPERTY, OBJECT_PROPERTY. Ceci permet alors de générer (1).

3.1.3 Choix d'une solution

Le tableau ci-dessus montre donc le caractère indispensable, si l'on souhaite que le système de gestion de la BDBO puisse s'adapter automatiquement à des évolutions de modèle d'ontologie, de représenter explicitement son méta-modèle. Cette représentation constitue alors la deuxième partie de notre modèle d'architecture appelée *méta-schéma.*

3.2 Représentation du méta-modèle d'ontologie : la partie méta-schéma

Ainsi que nous l'avons vu dans la section précédente, le principal objectif de la partie *méta-schéma* est d'offrir une interface de programmation permettant l'accès au modèle d'ontologie courant par parcours des instances représentées dans le *méta-schéma*. Ceci permet de rendre générique, par rapport aux modèles d'ontologies, un certain nombre de fonctions devant être réalisées par le système.

Lorsque le SGBD utilisé : (1) est un SGBD Objet, et (2) permet de représenter tous les mécanismes existant dans le formalisme de définition du modèle d'ontologie, alors cette partie préexiste en général dans le SGBD et s'identifie avec la méta-base du système. Dans tous les autres cas, il s'agit d'une partie nouvelle, nécessaire dans une BDBO.

Comme la partie *ontologie*, la partie *méta-schéma* doit offrir les fonctions de :

- **F1** génération du schéma logique de gestion des modèles d'ontologie;
- **F3** parcours des modèles d'ontologie par une API à liaison différée;
- **F4** parcours des modèles d'ontologie par une API à liaison préalable,

et, éventuellement, s'il existe un format de représentation externe des modèles d'ontologie sous forme d'instances d'un méta-modèle;

- ***F2*** *lecture / écriture de fichiers d'instances représentant des modèles d'ontologie*.

La discussion effectuée à propos de la partie ontologie s'applique à l'identique. Le méta-schéma doit lui même être représenté comme instance d'un méta-modèle.

Exemple 3.6 : En reprenant le méta-modèle de la figure 3, pour savoir si un *UML_attribute* dont l'oid est connu (oid_{att}) s'applique à une *UML_class* ou une *UML_association* (ce qui a des conséquences, par exemple, pour la représentation de l'association en Java), les deux fonctions suivantes devront être utilisées sur l'API à liaison différée.

```
oid1 :=              get_association              (oid_att,
"UML_attribute","is_of_type")
IF get_final_type (oid1, "UML_type") = "UML_class"
    THEN …
    ELSE …
    END IF ;
```

L'interprétation de la fonction *get_final_type* à son tour nécessite (1) de parcourir la structure du méta-schéma pour identifier les sous-types de

UML_type, puis (2) de chercher *oid1* dans la table *UML_class* et dans la table *UML_association*. Sauf, ici encore, à coder tout le modèle du méta-schéma dans les fonctions de l'API, ceci nécessite de représenter le méta-schéma comme instance d'un nouveau méta-modèle.

Les exigences existant jusqu'alors sur le méta-schéma, etaient d'être capable de représenter l'ensemble des variantes des modèles d'ontologie envisagés . Si nous imposons alors au méta-schéma d'être, de plus, réflexif, c'est-à-dire de pouvoir se représenter lui-même, on pourra représenter le méta-schéma comme instance de sa propre structure (cf. fig. 4). Ceci permettra à nouveau d'écrire, de façon générique, l'ensemble des quatre fonctions envisagées.

Notons de plus que si le formalisme utilisé pour définir le modèle d'ontologie et le modèle du méta-schéma est le même (par exemple : UML, ou alors EXPRESS) *alors le même code générique pourra être réutilisé* pour les fonctions **F1**, **F2**, **F3** et **F4**, *à la fois pour la partie ontologie, et pour la partie méta-schéma*.

UML_relation

OID	name	source	target	s_min	s_max	t_min	t_max
8	properties	1	2	1	1	0	null
9	superclass	2	1	0	1	0	null
21	target_class	4	1	1	1	0	null
18	is_of_type	16	13	1	1	0	null
19	of_UML_class	16	14	1	1	0	null

UML_data_type

OID	name
10	String
11	String
12	String
20	String

UML_class

OID	name	superclasses	abstract
1	class		false
2	property		true
3	data_property	2	false
4	object_property	2	false
13	UML_type		true
14	UML_class	13	false
15	UML_data_type	13	false
16	UML_attribute		false
21	UML_relation	13	false

UML_attribute

OID	name	of_class	is_of_type	…
5	name	1	10	
6	name	2	11	
7	value_type	3	12	
17	name	13	20	

Figure 4: *Auto-représentation du méta-schéma réflexif.*

3.3 Représentation des instances: la partie données

Une ontologie vise à représenter la sémantique des objets d'un domaine en les associant à des classes, et en les décrivant par les valeurs de propriétés. Selon les modèles d'ontologies utilisés, plus ou moins de contraintes existent sur ces

descriptions. Ainsi par exemple, en OWL, un objet peut appartenir à un nombre quelconque de classes et, sauf restriction particulière, être décrit par n'importe quelles propriétés. Ceci donne à chaque objet du domaine *une structure qui peut lui être spécifique*. A contrario, un schéma de base de données vise à décrire des ensembles d'objets "*similaires*" par une structure logique identique de façon à pouvoir optimiser les recherches sur tels ou tels ensemble par des techniques d'indexation. En absence de toute hypothèse particulière sur la représentation des objets du domaine, la seule structure commune possible consiste à pouvoir associer chaque objet :

– à un sous ensemble quelconque de l'ensemble des classes ;

– à un nombre quelconque de propriétés

Cette structure entraînerait soit une perte de place très considérable (avec représentation systématique des appartenances ou propriétés non pertinentes pour une instance), soit des temps de traitement important résultant de l'absence d'indexation ou de besoins de jointures nombreuses (si seules les appartenances et les propriétés pertinentes sont représentées).

Dans le cadre de notre modèle d'architecture OntoDB, nous imposons deux restrictions :

– **R1**- tout objet du domaine est décrit par son appartenance à une et une seule classe, dite *de base* qui est la borne inferieure unique de l'ensemble des classes auquel il appartient (il appartient bien sûr également à ses super-classes).

– **R2**- tout objet du domaine ne peut être décrit que par les propriétés applicables à sa classe de base (ceci signifie, si *c* est la classe de base, en PLIB : "appartenant à applic(c)", en OWL "propriété associée à un domaine et telle que ce domaine subsume c").

Il est alors possible de définir un schéma, significatif pour la classe, qui est constitué de toutes les propriétés applicables et effectivement utilisées (soit, selon le choix de l'administrateur de la BD, par toutes les instances représentées, soit par au moins une instance appartenant à cette classe comme classe de base).

Notons, et c'est une grande différence avec les BDOOs, qu'il est parfaitement possible qu'une propriété définie comme applicable au niveau d'une classe A soit effectivement utilisée, et donc effectivement représentée dans le schéma de deux sous classes (A1 et A2) de A, et non dans celui d'une troisième sous-classe (A3). Dans une BDBO, la relation de subsumption est *intentionnelle* et ne peut donc pas être simplement représentée par héritage au niveau du modèle logique de la partie *données* (cf. fig. 5b la propriété *gender*). Ainsi donc, malgré le caractère objet des données à représenter, et malgré la hiérarchie de classes existante au niveau ontologique, la représentation verticale usuelle (où chaque propriété est représentée dans la table correspondant au niveau où la propriété est définie) s'avère peu

adaptée. Dans notre implémentation, nous utilisons d'ailleurs la représentation dite horizontale où toutes les propriétés sont représentées au niveau des seules classes de bases (cf. fig. 5).

Nous présentons dans les deux parties suivantes, d'abord la représentation du schéma des données, puis la représentation des données elles-mêmes.

3.4 Représentation du schéma des instances : la partie méta-base

La définition des classes à représenter en tant que classes de base (donc associées à des instances), et des propriétés devant être représentées pour chaque classe de base est effectuée soit par l'administrateur, soit automatiquement (selon le scénario d'utilisation défini pour la BDBO) lors de la lecture des fichiers d'instances [Bellatreche et al, 03]. Ceci définit la relation devant être associées à chaque classe de base. De plus :

- la clé de cette relation (par défaut un identifiant d'objet généré par le système) peut être définie par l'administrateur;
- le schéma de cette relation (par défaut une simple table) peut être également défini par l'administrateur (nous étudions actuellement une alternative consistant en une génération en 3FN à partir des dépendances fonctionnelles qui seraient alors représentées dans l'ontologie).

Les informations, de niveau schéma, sont alors représentées dans la partie *méta-base* (ou *catalogue*) qui existe dans tout SGBD.

Correspondances entre schéma et ontologie

Schémas et Ontologies étant gérés séparément (partie *ontologie* et *méta-base*), il est nécessaire d'établir un mécanisme de liaison bilatére entre ces deux parties. Ce mécanisme de liaison est constitué de deux fonctions partielles :

- *Nomination* : {*classe* ∪ *propriété*} → {*table* ∪ *attribut*}
- *Abstraction* :{*table* ∪ *attribut*} → {*classe* ∪ *propriété*}

Ces deux fonctions sont partielles car :

- certaines classes et / ou propriétés peuvent ne pas être représentées;
- certaines tables et / ou attributs, de nom prédéfinis, correspondent à des informations de type système.

Suivant les caractéristiques du modèle d'ontologie utilisé, diverses solutions sont envisageables pour matérialiser ce lien. On peut par exemple utiliser les identifiants universels éventuellement définis par le modèle d'ontologie. Une autre approche,

utilisée dans notre prototype, consiste à utiliser les OIDs de la représentation des concepts dans l'ontologie pour nommer les tables et colonnes dans la partie *données*. L'exemple de la figure 5 illustre la façon dont les tables et colonnes sont nommées dans notre architecture de BDBO.

(a)

Objet1:student
noSS:string=02489 name:string=Woopy lastname:string=Golder age:int=35 gender:char=F

Objet2:employee
noSS:string=13457 name:string=Jean lastname:string=Claude age:int=40 salary:double=1500 service:string=CRCFAO

(b)

id_student						
OID	**id_NoSS**	**id_name**	**id_lastname**	**id_age**	**id_gender**	**id_conjoint**
100	02489	Woopy	Golber	35	F	103

id_employee						
OID	**id_NoSS**	**id_name**	**id_lastname**	**id_age**	**id_salary**	**id_service**
103	13457	Jean	Claude	40	1500	CRCFAO

Figure 5: *exemple de population (a) et sa représentation dans un schéma extrait de l'ontologie de la figure 2 (b).*

3.5 Représentation des instances : la partie données

La représentation des données d'instances est alors effectuée dans la partie donnée du SGBD support conformément au schéma défini dans la méta-base (cf. fig. 5).

3.6 Le modèle OntoDB

La figure 6 représente l'architecture logique que nous proposons donc pour la réalisation d'une BDBO. Ce modèle d'architecture, appelé *OntoDB*, permet de représenter simultanément des ontologies de domaines, décrites en termes de classes et de propriétés, et des objets du domaine, définis en termes de ces ontologies et respectant les deux hypothèses **R1** et **R2** de la section 3.2.

Le modèle comporte quatre parties :

1. La partie *ontologie* permet de représenter complètement les ontologies de domaines définissant la sémantique des différents domaines couvert par la BD, ainsi, éventuellement, que l'articulation (représentée sous forme de "*mapping*"

[Bernstein et al, 00] [Pierra, 03]) de ces ontologies avec des ontologie externes, par exemple normalisées.

2. La partie *méta-schéma* permet de représenter, au sein d'un modèle réflexif, à la fois le modèle d'ontologie utilisé et le méta-schéma lui-même.

3. La partie *méta-base* permet de représenter le schéma de représentation des objets du domaine couvert par les ontologies.

4. La partie *données* représente les objets du domaine; ceux ci sont décrits en termes d'une classe d'appartenance et d'un ensemble de valeurs de propriétés applicables à cette classe.

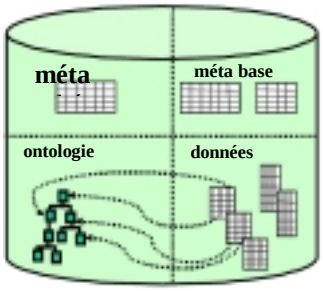


Figure. 6 *Architecture de Base de Données à Base Ontologique (BDBO) selon le modèle OntoDB.*

Notons que ce modèle d'architecture n'est pas le seul modèle possible pour une BDBO.

– Partie ontologie et partie données peuvent éventuellement être représentées dans deux systèmes différents, coordonnés par une interface d'accès commune (nous avons également réalisé un prototype selon cette architecture physique).

– Si le niveau méta-schéma est nécessaire en génération, cette génération n'est réellement indispensable que lors d'un changement de modèle d'ontologie. On peut donc éventuellement effectuer les générations correspondant aux fonctions **F1**, **F2**, **F3** et **F4** à l'aide d'un système externe dans lequel la partie méta-schéma serait représenté. Notons cependant que, dans ce cas, il faudrait régénérer également à chaque évolution du modèle d'ontologie de l'API à liaison différée, alors que ceci n'est pas nécessaire si elle peut accéder au méta-schéma.

Pour conclure cette section, nous pouvons constater que notre architecture de BDBO a de fortes similitudes avec l'architecture *metadata* du MOF (Meta Object Facility) [Kobryn, 99]. Notre architecture est constituée de quatre couches superposées. La couche modèle M1 de l'architecture MOF correspond à notre modèle conceptuel, sous ensemble de l'ontologie. La couche méta-modèle M2 correspond au modèle d'ontologie (fig. 2), la couche méta-méta-modèle M3 (MOF model) correspond au méta-modèle du langage de définition du modèle d'ontologie

(cf. fig. 3), lui-même réflexif (cf. fig. 4). Cette architecture, nous permet d'intégrer automatiquement [Bellatreche et al, 2003][Bellatreche et al, 2004], de faire migrer et d'échanger des instances non nécessairement définies dans le même modèle d'ontologie. Elle nous permet également d' utiliser les résultats des travaux effectués dans le cadre du MOF.

4. Mise en œuvre

Afin de valider notre architecture OntoDB, nous avons implémenté un prototype de BDBO sur le SGBD PostgreSQL qui est de type relationnel-objet. Les ontologies auxquelles nous nous intéressons sont celles définies par le modèle d'ontologie PLIB spécifié en EXPRESS. Elles sont donc échangeables sous forme de fichier d'instances EXPRESS ("fichier physique"). Pour la mise en œuvre de notre architecture, nous avons utilisé un environnement de développement EXPRESS (*ECCO Toolkit* de *PDTec*), qui a la particularité de lire et écrire des fichiers d'instances et de permettre d'accéder à la description d'un schéma sous forme d'instances d'un méta-schéma.

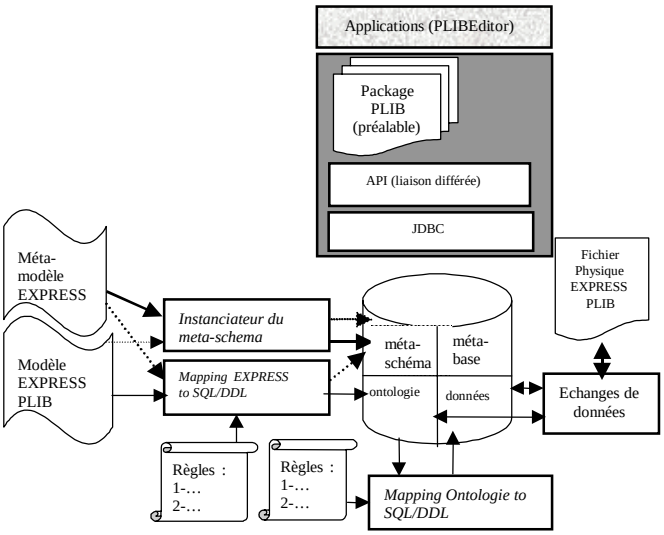


Figure 7. BDBO basée sur des ontologies PLIB et l'environnement PostgreSQL.

Pour la représentation de la partie *ontologie* (implémentation de la fonction **F1,** cf. table 1), nous avons d'abord défini des règles de correspondance entre les différents concepts EXPRESS et ceux du SGBD cible. Les règles (analogues à celles de [Blaha et al, 1994] pour l'essentiel) ont été codées pour constituer un générateur qui reçoit en entrée un modèle EXPRESS quelconque et produit le modèle logique

PostgreSQL correspondant. Pour générer respectivement le modèle logique de la partie *ontologie* et de la partie *méta-schéma*, il suffit de transmettre au générateur, d'un part, le modèle EXPRESS de PLIB, et, d'autre part, le modèle EXPRESS d'un méta-modèle EXPRESS que nous avons défini. La partie *méta-schéma* ainsi générée est alors est peuplée d'abord avec le modèle d'ontologie PLIB, puis avec le méta-modèle EXPRESS par un autre programme, dit *instanciateur* (cf. fig. 7), qui prend en entrée un modèle EXPRESS et l'insère dans les tables du *méta-schéma* comme données. La mise en œuvre de la partie *données* est semblable, dans le principe, à celle de la partie *ontologie*. Un autre générateur, qui prend en entrée l'ontologie et la description des propriétés devant figurer dans le schéma, génère le modèle logique PostgreSQL des données. Ce générateur, quant à lui, est basé sur des règles de correspondances entre les concepts objets de PLIB, et ceux du SQL/DDL de PostgreSQL. Nous avons par ailleurs implémenté un module d'échanges des données contenu dans la BDBO (ontologies et instances) avec d'autres sources de données par lecture / écriture de fichiers physiques EXPRESS.

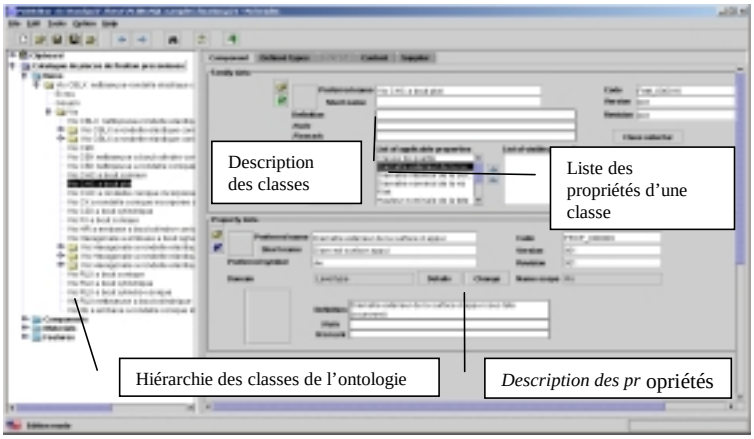


Figure 8. *Edition des classes et des propriétés de l'ontologie.*

Pour accéder par programme au contenu de la BDBO, nous avons d'abord défini une API à liaison différée (dans le langage *plpgsql* natif du SGBD PostgreSQL). Pour la partie ontologie, nous avons ensuite généré une interface à liaison préalable spécifique du modèle PLIB (permettant donc le contrôle syntaxique des appels en java). Pour finir, nous avons développé en java un éditeur (PLIBEditor) qui permet aussi bien de créer et de visualiser des classes et des propriétés dans l'ontologie (cf. fig. 8) et que de créer et visualiser des instances de ces classes (cf. fig. 9).

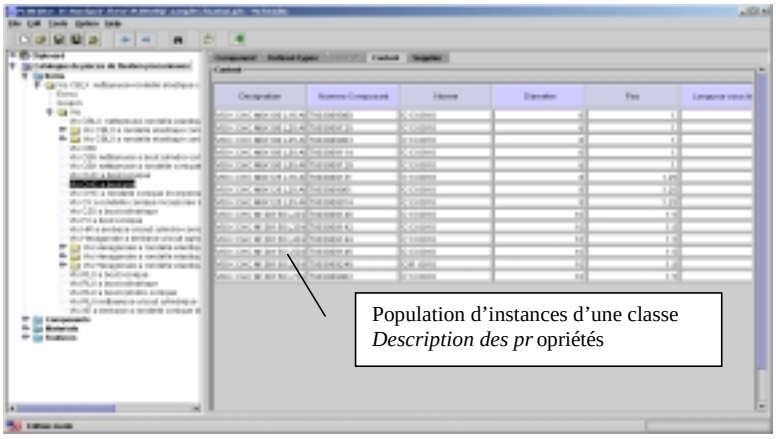


Figure 9. Edition d'instances d'une classe.

Conclusion

Dans ce travail, nous avons proposé une nouveau modèle de base de données qui vise à représenter, en plus des données, *l'ontologie* qui en représente le sens, le *modèle conceptuel* qui en définit la structure, et, éventuellement les *correspondances* existant entre l'ontologie locale et des ontologies partagées. Les bases de données qui possèdent ces caractéristiques sont appelées *bases de données à base ontologique* (BDBO). Nous avons également proposé un modèle d'architecture complet pour les BDBOs. Ce modèle, appelé *OntoDB*, définit quatre parties. Outre les deux parties classiques des bases de données : *données* et *méta-base* qui stockent, d'une part, les données d'instances, et, d'autre part, le schéma, elle comprend une partie qui représente l'*ontologie*, et une autre partie, appelée *méta-schéma*, qui représente sous forme d'instances, au sein d'un modèle réflexif, le modèle de données de l'ontologie.

Contrairement aux approches objet et relationnel-objet qui visent à rapprocher le modèle conceptuel et le modèle logique d'une base de données, notre approche consiste à représenter dans la base de données les deux modèles, ainsi que leurs relations. Elle représente également l'ontologie qui décrit explicitement le sens des données.

Cette approche présente les avantages suivants :

– Les performances de stockage et de requêtes des SGBDR et SGBDRO sont conservées. Cela est simplement dû au fait que notre modèle d'architecture de BDBO est greffée sur ces systèmes.

– L'accès aux données est offert au niveau sémantique, et ce sans aucune programmation spécifique. La présence de l'ontologie dans la base de données permet en effet de réaliser des requêtes sur les données en termes d'ontologies,

indépendamment du modèle logique. Un langage de requête spécifique est d'ailleurs en cours de développement en collaboration avec TOSHIBA [Mizoguchi et al, 02].

- L'ontologie est elle-même interrogeable simultanément avec les données. C'est une des grandes originalités de notre approche. En effet, on peut effectuer des requêtes du type :" je veux toutes les instances de classes dont le nom contient la chaîne "*VIS"* et dont la propriété "*masse"* est inférieure à 50 *grammes"*.
- La représentation explicite de la sémantique des données dans la base prépare celle-ci pour une future intégration avec d'autres sources de données. Ceci débouche sur une nouvelle approche, qualifiée de *a priori* pour l'intégration de sources de données hétérogènes [Bellatreche et al, 03] [Bellatreche et al, 04].
- Dans une BDBO, le modèle logique ne pouvant être créé qu'a partir de l'ontologie, le modèle conceptuel ne peut qu'évoluer simultanément avec le modèle logique de données, ce qui n'est pas souvent le cas dans les approches classiques.
- Enfin, lorsque des ontologies PLIB sont utilisées, la structure proposée apparaît particulièrement bien adaptée à la fois pour des serveurs de commerce électroniques professionnels et pour les bases de données de composants industriels.

Le type de connaissance représentable dans une BDBO dépend des caractéristiques du modèle d'ontologie utilisé, et de la diversité des liens pouvant être établis, d'une part, entre le niveau conceptuel et le niveau logique et, d'autre part, entre ontologie locale et ontologie partagée. Différents travaux sont encours actuellement dans le laboratoire, à la fois pour augmenter le pouvoir d'expression du modèle d'ontologie utilisé, en y ajoutant en particulier des fonctions de dérivations et certain constructeurs des logiques de description, et, en modélisant explicitement les dépendances fonctionnelles au niveau ontologique de façon à normaliser automatiquement les schémas physiques.

références

[Blaha et al, 1994] M.Blaha, W.Premerlani, H.Shen, *Converting OO Models into RDMS Schema*, Software, Volume 11, Issue 3 (May 1994), pp 28-39

[Broekstra et al, 02] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. Van Harmelen, I. Horrocks, *Enabling knowledge representation on theWeb by extending RDF Schema*, Computer networks, 39 (2002) pp. 609-634.

[Bellatreche et al, 03] L. Bellatreche, G. Pierra, D. Nguyen Xuan and H. Dehainsala, *An Automated Information Integration Technique using an Ontology-based Database Approach*. in Proc. of 10th ISPE International Conf. on Concurrent Engineering: Research and Applications (CE'03) : Special Track on Data Integration in Engineering, eds. R. Jardim- Gonsalves, J. Cha and A. Steimer-Garo (Balkena, Lisse, 2003)

[Bellatreche et al, 04], L.Bellatreche, G.Pierra, D.Nguyen Xuan and H. Dehainsala., *Integration de sources de données autonomes par articulations à priori d'ontologies* (To appear INFORSID 2004, Biarritz)

[Bernstein et al, 00] P. A. Bernstein, A. Y. Havely and R. A. Pottinger, *A vision of management of complex models, SIGMOD Record* 29(4) (2000) pp. 55-63.

[Chung et al, 95] J.Y. Chung, Y-J. Lin, *Objects and Relationnal Databases*, OOPSLA Worshop, 1995

[Chawathe et al, 95] S.Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J Widom. *The tsimmis project: Integration of heterogeneous information sources*. In IPSJ Conference, pages 7-18, Tokyo, Japan, October 1994.

[Fensel et al, 00] D. Fensel et al.: *OIL in a nutshell* In: Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000), R. Dieng et al. (eds.), Lecture Notes in Artificial Intelligence, LNAI, Springer-Verlag, October 2000.

[Guarino et al, 02] N. Guarino, C. Welty, *Evaluating ontological decision with Ontoclean*, *Comm. ACM*, 45(2) (2002) pp. 61-65.

[Gruber, 93] Gruber T, *A translation approach to portable ontology specification.* Knowledge Acquisition, 7, 1993

[ISO 13584-42, 98] G. Pierra, Eds. *ISO 13584-42 Insdustrial Automation Systems and Integration- Part Library-Methodology for Structuring -Parts Famillies ISO*, Geneva, 1998.

[ISO 13584-25, 04] ISO/IS 13584-25 Y. Ait-Ameur, G. Pierra, Eds.,*"Industrial Automation Systems and Integration, Parts Library, Logical model of supplier library with aggregate values and explicit content*, ISO 13584-25", Genève, ISO , 2004

[Kobryn, 99] C. Kobryn, *UML 2001 : A Standardization Odyssey*, Communications of the ACM, vol. 42, no. 10, Oct. 1999.

[Mizoguchi et al, 02] Mizoguchi, Y., Murayama, H & Minamino, N. 2002, *Class Query Language and its application to ISO13584 Parts Library Standard*, ECEC2002

[Pierra, 03] G. Pierra, *Context-Explication in Conceptual Ontologies: The PLIB Approach*, Proc. of 10th ISPE International Conf. on Concurrent Engineering: Research and Applications (CE'03) : Special Track on Data Integration in Engineering, eds. R. Jardim-Gonsalves, J. Cha and A. Steimer-Garo (Balkena, Lisse, 2003)

[Raharu et al, 00] J.W.Raharu, E.Chang, T.S.Dillon, D.Taniar, *A methology for transforming inheritance relationships in an object-oriented conceptual model tables*, Information and Software Technology 42 (2000) 571-592.

[Schenck et al, 94] Schenck D., Wilson P. - *Information Modelling The EXPRESS Way*, Oxford University Press, 1994.

[Stonebraker, 1996] Stonebraker, M. *Object-Relational Database Systems: The Next Great Wave*, Morgan Kaufmarm Publishers, San Francisco, CA, 1996.

[Stoimenov et al, 99] L. Stoimenov, A.Mitrovic, S.Djordijevic-Kajan, D. Mitrovic, *Bridging objects and relation : mediator for an OO front-end to RDBMSs*, Information and Software Technology 41, pp 57-66, 1999

[Sutherland, 93] Sutherland, JV, Rugg, K, Pope, M *The Hybrid Object-Relational Architecture (HORA): An Integration of Object-Oriented and Relational Technology*. In Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing. : ACM Press, pp 326-333.