Extension de ONTODB pour Construire une Architecture Générique de Bases de Données à Base Ontologique

Youness Bazhar
LIAS - ISAE ENSMA
et Université de Poitiers
Futuroscope, France
bazhary@ensma.fr

Stéphane Jean
LIAS - ISAE ENSMA
et Université de Poitiers
Futuroscope, France
jean@ensma.fr

Yamine Aït-Ameur
IRIT - ENSEEIHT
Toulouse, France
yamine@enseeiht.fr

Mickaël Baron
LIAS - ISAE ENSMA
et Université de Poitiers
Futuroscope, France
baron@ensma.fr

ABSTRACT

Ces dernières années ont vu l'émergence des ontologies comme modèles de représentation de connaissances. Elles ont été utilisées dans de nombreux domaines et pour des applications très variées. Cette large utilisation a engendré deux problèmes. D'une part, les données existantes qui sont décrites par des ontologies sont volumineuses, et d'autre part, les ontologies existantes ne sont pas toutes semblables : les ontologies peuvent être définies avec différents langages ou formalismes, menant à différents types d'ontologies (canoniques, non canoniques et ontologies linguistiques). Le premier problème a été résolu en utilisant des bases de données pour représenter et assurer la persistance des données et des ontologies qui en définissent la sémantique. Toutefois, ces bases de données, appelées bases de données à base ontologique (BDBO), ne traitent pas le second problème, à savoir le support de différents modèles d'ontologies. En effet, les BDBO supportent généralement un seul modèle d'ontologies, et ne permettent pas la gestion d'ontologies définies à l'aide de langages ou de modèles d'ontologies différents et donc hétérogènes. Dans cet article, nous proposons une extension de la BDBO OntoDB avec de nouveaux opérateurs qui peuvent être exploités par son langage d'exploitation OntoQL. Ces opérateurs peuvent être implémentés avec un programme externe situé à l'extérieur de la base de données, ou bien peuvent invoquer des services web. Nous montrons que ces opérateurs peuvent aussi être utilisés pour définir de nouveaux constructeurs d'ontologies afin de supporter plusieurs modèles d'ontologies dans OntoDB, définissant ainsi une architecture générique de BDBO. Pour supporter cette affirmation, nous montrons comment les opérateurs proposés peuvent permettre le support des constructeurs de concepts ontologiques non canoniques initialement non supportés par la plate forme OntoDB/OntoQL.

Keywords

Ontologie, base de données à base ontologique, OWL, sémantique structurelle, descriptive et comportementale, concepts canoniques, concepts non canoniques

1. INTRODUCTION

Depuis leur apparition, les ontologies ont été largement utilisées et suscitent encore beaucoup d'intérêt avec le développement récemment du web sémantique. En effet, des ontologies sont définies dans de nombreux domaines comme l'ingénierie, la biologie, la chimie, etc., et sont utilisées dans différentes applications, comme le traitement de la langue naturelle, l'intégration de systèmes, la recherche d'information, etc.

L'utilisation intense des ontologies dans de nombreux domaines conduit à deux problèmes majeurs. (1) La quantité des données décrites par les ontologies peut être importante, en particulier dans des domaines comme l'e-commerce ou le web sémantique. (2) Il existe plusieurs formalismes et langages d'ontologies. Par exemple, de nombreuses ontologies dans le domaine de l'ingénierie sont définies avec PLIB [18, 19], alors que dans le web sémantique, les ontologies sont habituellement définies avec RDF-Schema [3] ou OWL [7]. Le premier problème a été résolu par l'arrivée d'un nouveau type de bases de données qui stocke à la fois les données et les ontologies qui en décrivent la sémantique. De nombreuses bases de données présentant ces caractéristiques, appelées Bases de Données à Base Ontologiques (BDBO), ont été proposées dans la littérature (e.g. Jena [5], Sesame [4], Oracle [6], RStar [13]). Ces systèmes permettent le stockage d'une grande quantité d'ontologies et d'instances. Par contre, ces BDBO sont généralement conçues pour un modèle d'ontologies bien défini. Ainsi, elles ne peuvent pas résoudre le deuxième problème relevé c.-à-d., l'existence de plusieurs modèles d'ontologies. Notre objectif est de résoudre ce problème en utilisant des capacités de méta-modélisation au sein des BDBO.

Dans ce papier, nous considérons la BDBO OntoDB [8]. A l'origine, cette BDBO a été définie pour le modèle d'ontologies PLIB. Ainsi, elle supporte les constructeurs de base

d'une ontologie, à savoir les constructeurs de classe et de propriété. Par contre, OntoDB ne permet pas de définir des *concepts non canoniques* (aussi appelés concepts définis ou dérivés) c.-à-d. des concepts définis par une relation exprimée en termes d'autres concepts [9]. Ce type de concepts est particulièrement important dans le modèle d'ontologies OWL. Par exemple, ce modèle permet de définir une classe comme étant l'union d'autres classes. Il permet aussi de définir une classe comme étant l'ensemble des instances qui ont une certaines valeur de propriété (restriction de classe). Aussi, notre proposition consiste à étendre OntoDB avec de nouveaux opérateurs qui peuvent être exploités pour supporter les concepts non canoniques. Ces opérateurs sont flexibles car ils peuvent être implémentés par des programmes externes définis en dehors de la base de données ou peuvent invoquer des services web. Pour montrer l'intérêt de ces opérateurs, nous montrons comment ces opérateurs peuvent être utilisés pour implémentation des concepts non canoniques dans le système OntoDB/OntoQL.
Le reste du papier est organisé comme suit. La section 2 présente quelques travaux connexes au problème traité. La section 3 décrit la BDBO OntoDB et son langage d'exploitation OntoQL sur lesquels notre approche est basée. La section 4 présente l'extension de la BDBO OntoDB pour supporter les concepts non canoniques. La section 5 explique l'implémentation de quelques concepts non canoniques dans OntoDB, et finalement, la section 6 conclut cet article.

2. TRAVAUX CONNEXES

Durant ces dernières années, plusieurs architectures de BDBO ont été proposées pour stocker les données et les ontologies qui en décrivent la sémantique. Nous présentons les différents types d'architectures en fonction des niveaux d'abstraction supportés. Nous décrivons également les possibilités d'extension permises par ces architectures.

- **Les BDBO de type1** sont utilisées pour stocker les triplets RDF qui peuvent contenir la définition d'ontologies et d'instances. Les principales BDBO dédiées au stockage de triplets RDF sont 3Store [10], Jena [5], Oracle [6] ou Sward [17]. Ces BDBO utilisent un seul schéma pour stocker les ontologies et leurs instances. Ce schéma est composé d'une seule table de triplets composée de trois colonnes (`sujet`, `predicat`, `objet`) représentant respectivement le sujet, le prédicat et l'objet. Par exemple, au niveau ontologie, le triplet (`Name`, `domain`, `Person`) indique que le domaine de la propriété `Name` est la classe `Person`. Au niveau instance, le triplet (`i`, `name`, `Henry`) indique que `Henry` désigne le nom de l'instance `i`.
  Puisque les données RDF peuvent comporter des descriptions d'ontologies RDFS ou OWL, la plupart de ces BDBO supportent la sémantique d'OWL ou de RDFS. Cette sémantique est habituellement exprimée par des règles déductives [6] ou en faisant appel à des raisonneurs externes [10].
- **Les BDBO de type2** stockent les ontologies et leurs instances dans deux parties séparées. Les exemples principaux des BDBO de type2 sont RDF Suite [1], Sesame [4], RStar [13], DLDB [15] ou OntoMS [16]. Dans ce type de BDBO, le schéma de stockage des ontologies dépend du modèle d'ontologies supporté. Ainsi, ce schéma est composé de tables qui permettent de stocker les concepts des ontologies comme des classes, des propriétés et des relations d'héritage entre classes. Pour le stockage des instances d'ontologies, plusieurs schémas ont été proposés présentant chacun des capacités de passage à l'échelle différentes.
  Les BDBO de type2 supportent généralement les relations d'héritage telles qu'elles sont spécifiées dans la sémantique de RDFS [11] (c.-à-d. les relations `subClassOf` et `instanceOf`). Pour cela, elles utilisent les mécanismes offerts par les bases de données, comme les vues [16], les schémas étiquetés [1, 4] ou les relations d'héritage entre tables issues des bases de données objet-relationnel [14, 20, 2, 15]. Certaines BDBO traitent également des raisonnement complexes en utilisant des raisonneurs de bases de données déductives basés sur la logiques (par exemple, un raisonneur Datalog), ou en utilisant des raisonneurs OWL [14, 20, 2, 15].
- **Les BDBO de type3**. OntoDB [8] propose d'ajouter un autre schéma aux BDBO de type2. Ce nouveau schéma, appelé *méta-schéma*, permet de stocker les modèles d'ontologie. Pour les ontologies, le méta-schéma joue le même rôle que celui joué par la méta-base pour la partie données dans les bases de données traditionnelles. Si cette partie supporte l'évolution des modèles d'ontologies supportés, nous montrons dans la section suivante que la sémantique comportementale des concepts ontologiques ne peut pas être exprimée.

Synthèse. Comme nous avons vu dans cette section, les travaux autour des BDBO se sont essentiellement concentrés sur le passage à l'échelle de ce nouveau type de bases de données. Concernant le support de différents modèles de d'ontologies, chaque BDBO supporte un modèle spécifique en utilisant des mécanismes spécifiques. Le but de nos travaux est de proposer une approche plus flexible, c.-à-d. qui permet le support de différents modèles d'ontologies de différentes manières (par exemple, par des mécanismes internes à la base de données ou par l'appel de services web). Comme nous l'avons vu, OntoDB propose la partie méta-schéma qui introduit des capacités de méta-modélisation au sein des BDBO. Notre approche repose sur ce schéma pour stocker l'extension proposée. Ainsi, nous présentons cette BDBO plus en détail dans la prochaine section.

3. LE SYSTÈME ONTODB/ONTOQL

Dans cette section, nous présentons l'architecture OntoDB [8], et son langage d'exploitation OntoQL [12]. Ensuite, nous montrons à travers un exemple les limites de ce système concernant le support des concepts ontologiques non canoniques du modèle OWL.

3.1 L'architecture de BDBO OntoDB

OntoDB est une BDBO composée de quatre parties (Figure 1). Les parties *Méta-base* et *Instances* représentent les parties classiques des bases de données traditionnelles. Ainsi, la partie *Méta-base* contient les tables système utilisées pour gérer toutes les données contenues dans la base de données. La partie *Instances* contient les données qui représentent les instances d'ontologies. La partie *méta-schéma* contient le modèle d'ontologie supporté (le méta-modèle de PLIB). Enfin, la partie *Ontologies* contient des ontologies conformes aux modèles pris en charge par OntoDB.
Dans OntoDB, l'ensemble des données des différents niveaux d'abstraction (méta-schéma, ontologies et instances) sont stockées dans des tables relationnelles. Pour illustrer cette

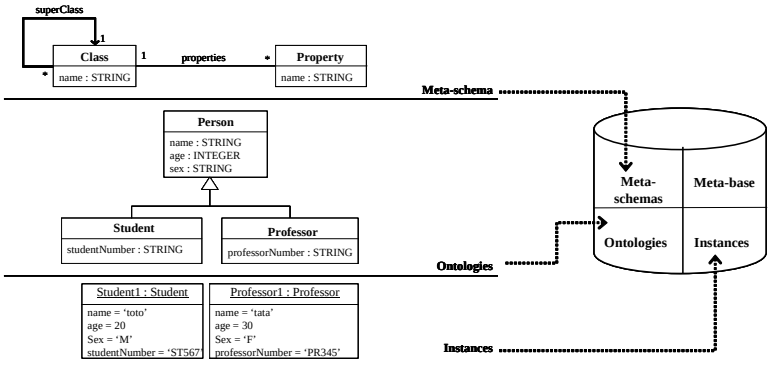
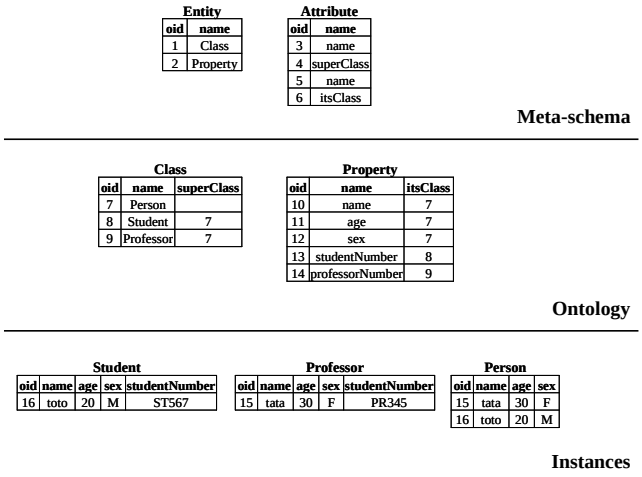


Figure 1: Stockage des modèles d'ontologies, des ontologies et des instances dans OntoDB.

représentation, considérons l'exemple de la Figure 1. Dans cet exemple, trois niveaux de modélisation sont représentés. Le niveau méta-schéma définit un modèle d'ontologie contenant deux entités : `Class` et `Property`. `Class` est la description abstraite d'un ou plusieurs objets similaires. Les classes sont organisées dans une hiérarchie exprimée par la relation d'héritage (`superClass`). `Property` décrit les propriétés d'une classe. Le niveau ontologies définit les classes `Person`, `Student` et `Professor` caractérisées par plusieurs propriétés. Enfin, le niveau instances définit des instances des classes `Student` et `Professor` définies au niveau ontologies.

La Figure 2 illustre la représentation utilisée par OntoDB



Entity

oid	name
1	Class
2	Property

Attribute

oid	name
3	name
4	superClass
5	name
6	itsClass

Meta-schema

Class

oid	name	superClass
7	Person	
8	Student	7
9	Professor	7

Property

oid	name	itsClass
10	name	7
11	age	7
12	sex	7
13	studentNumber	8
14	professorNumber	9

Ontology

Student

oid	name	age	sex	studentNumber
16	toto	20	M	ST567

Professor

oid	name	age	sex	studentNumber
15	tata	30	F	PR345

Person

oid	name	age	sex
15	tata	30	F
16	toto	20	M

Instances

Figure 2: Représentation des données dans OntoDB.

des données de l'exemple précédent. Dans cet exemple, les données de la partie méta-schéma sont stockées dans les tables `Entity` et `Attribute`. La première table contient ainsi les deux entités `Class` et `Property`, et la deuxième table contient les attributs des entités définies. Pour chaque entité, une table correspondante existe au niveau ontologies. Ces tables contiennent des instances des différentes entités définies dans la partie méta-schéma. Dans notre exemple, elles contiennent des classes et des propriétés. Enfin, pour chaque classe, une table correspondante est définie au niveau instances pour stocker les instances.

3.2 Le langage d'exploitation OntoQL

Puisque toutes les données sont stockées dans des tables relationnelles au sein d'OntoDB, on peut penser que langage SQL est suffisant pour les manipuler. Cependant, pour utiliser SQL dans OntoDB, les utilisateurs devraient avoir une connaissance profonde de la représentation interne des tables utilisées par OntoDB. Pour surmonter ce problème, OntoDB a été équipé d'un langage d'exploitation nommé OntoQL qui cache les représentations internes et manipule directement les modèles d'ontologie et les concepts d'ontologies. Ce langage peut être utilisé pour créer de nouveaux modèles d'ontologie et les instancier pour créer des ontologies. Les instructions suivantes représentent des exemples de requêtes OntoQL agissant à différents niveaux :

```
CREATE ENTITY #Class (#name STRING, #superClass REF(#Class));
CREATE ENTITY #Property (#name STRING, #itsClass
REF(#Class));
CREATE #Class Person (name STRING, age INT, sex STRING);
CREATE #Class Student UNDER Person (studentNumber STRING);
CREATE #Class Professor UNDER Person (ProfessorNumber
STRING);
INSERT INTO Professor VALUES ('tata', 30, 'F', 'PR345');
INSERT INTO Student VALUES ('toto', 20, 'M', 'ST567');
```

Les deux premières instructions définissent un modèle d'ontologie. L'entité `Class` est définie avec un attribut `name` et une super-classe `superClass` (une référence à une autre classe). L'entité `Propriété` possède aussi un attribut `name` et elle est liée à la classe qu'elle décrit (`itsClass`). Dans ces exemples de requêtes OntoQL, les noms d'entités et d'attributs sont préfixés par le caractère `#`. En effet, la partie méta-schéma de OntoDB peut être étendue et modifiée, et par conséquent, ce niveau d'abstraction ne peut pas être codé par des mots clés du langage OntoQL. Les trois instructions suivantes définissent les différentes classes de notre exemple avec leurs propriétés en utilisant la clause `CREATE #Class`. Une classe est définie comme une sous-classe d'une autre classe en utilisant le mot-clé `UNDER`. Enfin, les deux dernières instructions permettent de définir des instances de nos classes en utilisant une commande `INSERT INTO` similaire à celle du langage SQL.

3.3 Représentation des concepts canoniques dans OntoDB

Pour montrer les limites de la plate forme OntoDB/OntoQL pour le support de différents modèles d'ontologies, considérons le méta-modèle simplifié du langage OWL représenté dans la Figure 3. Ce méta-modèle contient deux constructeurs de concepts canoniques : `OWLClass` et `OWLProperty` permettant la création de classes et de propriétés primitives OWL. Tous les autres constructeurs de ce méta-modèle permettent de définir des concepts non canoniques. Par exemple, `UnionClass` est une classe OWL construite à partir de l'union d'un ensemble de `OWLClass`. Ainsi, `UnionClass` est un const ructeur de concept non canonique. En utilisant OntoQL, nous pouvons représenter les constructeurs de ce méta-modèle au sein d'OntoDB. Par exemple, les deux instructions suivantes permettent d'ajouter les constructeurs de `OWLClass` et `OWLProperty` au sein d'OntoDB :

```
CREATE ENTITY #OWLClass (#uri STRING, #superClasses REF
(#OWLClass) ARRAY, #subClasses REF (#OWLClass) ARRAY);
CREATE ENTITY #OWLProperty (#uri STRING, #domain REF (#OWL-
Class) ARRAY);
```

3.4 Limites du systèmes OntoDB/OntoQL

Comme nous l'avons vu dans la section précédente, le langage OntoQL peut être utilisé pour étendre le méta-schéma d'OntoDB avec de nouvelles entités. Ainsi, le constructeur `UnionClass` peut être ajouté avec la commande suivante :

```
CREATE ENTITY #UnionClass UNDER #OWLClass (#unionOf
REF(#Class) ARRAY);
```

Cependant, cette requête OntoQL ne définit que la structure du constructeur `UnionClass` (c.-à-d. que `UnionClass` possède les mêmes attributs que `OWLClass` et qu'elle est définie par un ensemble de classes). Cependant, nous ne sommes pas capables d'exprimer, ni à l'aide de cette requête, ni avec aucune autre requête OntoQL, que les instances d'une union de classes peuvent être calculées comme l'union des instances des classes utilisées dans sa définition *(instances(C) = instances(C1) U instances(C2) U ... U instances (Cn))*, et que la classe résultante de l'union devient une superclasse des classes qui la définissent (*subClasses(C) = {C1, C2, ..., Cn}*). Par conséquent, OntoQL ne supporte pas la définition de la sémantique comportementale des concepts non canoniques d'OWL (`UnionClass`, `IntersectionClass`, etc.). Cette sémantique pourrait être définie à l'aide d'opérations telles que des fonctions ou des procédures implémentées à *l'intérieur* (procédures stockées), ou à *l'extérieur* (programmes externes) de la base de données, ou avec des invocations de services web. Ainsi, nous devons être autorisés à définir la sémantique de l'opérateur *unionOf* à travers une requête ressemblerait à :

```
CREATE #UnionClass C1 AS unionOf(C2, C3);
```

Cette requête devrait permettre la modification de la structure des classes C2 et C3. En effet, C1 devient une superclasse de C2 et C3. De même, C2 et C3 deviennent des sous-classes de C1. Par ailleurs, l'union de classes définit les individus de C1 comme l'union des individus de C2 et C3. Par conséquent, un opérateur `unionOfInstances` est requis pour permettre le calcul des individus. Cet opérateur pourrait être utilisé comme suit :

```
CREATE EXTENT OF C1 AS unionOfInstances(C2, C3);
```

Les opérateurs *unionOf* et *unionOfInstances* devraient pouvoir être implémentés par des procédures internes ou externes ou peuvent invoquer des services web. Ainsi, l'extension du langage OntoQL par les concepts de procédure et de fonction permettrait d'exprimer la sémantique comportementale des concepts non canoniques d'OWL. Offrir ces capacités au sein d'OntoDB est notre objectif. Sa réalisation est détaillée ci-dessous.

4. EXTENSION DE LA PLATE FORME ONTODB AVEC LA SÉMANTIQUE COMPORTEMENTALE

Notre objectif est d'étendre l'architecture de OntoDB et de son langage d'exploitation (OntoQL) avec le support des opérations. Ces opérations peuvent être implémentées par des procédures stockées dans le SGBD, avec des programmes externes définis en dehors de la base de données ou avec des invocations de services distants.
Cette extension est réalisée en deux étapes. La première étape consiste à étendre l'architecture d'OntoDB pour autoriser le stockage des informations sur les opérations définies. La deuxième étape consiste à étendre le langage OntoQL pour supporter la définition et l'appel d'opérations.

4.1 Extension de l'architecture OntoDB

Nous avons étendu le méta-schéma d'OntoDB avec une nouvelle entité `EXTERNAL_PROGRAM` pour permettre le stockage des informations concernant les implémentations externes. En effet, ces informations peuvent être par exemple le nom de le fonction à invoquer, le nom de la classe où elle est définie, le nom du paquetage, son emplacement, etc. La requête OntoQL qui permet d'étendre la partie méta-schéma `EXTERNAL_PROGRAM` est :

```
CREATE ENTITY #EXTERNAL_PROGRAM (#FCT_NAME STRING, #PA-
CKAGE_NAME STRING, #CLASS_NAME STRING, #LOCATION STRING);
```

Nous avons, de plus, mis en place une API qui permet d'invoquer, à partir de la plate-forme OntoDB, des programmes externes. Ainsi, nous pouvons, par exemple, définir un programme externe qui implémente l'opérateur `unionOf` :

```
CREATE #EXTERNAL_PROGRAM ('unionOf', 'fr.ensma.lisi.owlnccon-
cepts', 'NCOperators', 'D://owlncoperators.jar');
```

Cette instruction indique que `unionOf` est un programme externe défini dans la classe `NCOperators` du paquetage `fr.ensma.lisi.owlncconcepts` contenu dans une archive Java dont le chemin d'accès est `D://owlncoperators.jar`.

4.2 Extension du langage OntoQL

Après l'extension de l'architecture d'OntoDB, nous avons étendu le langage OntoQL pour prendre en compte la définition et l'invocation de programmes externes. Ainsi, nous sommes capables de faire des appels de fonctions externes dans les requêtes OntoQL comme le montre la requête sui-

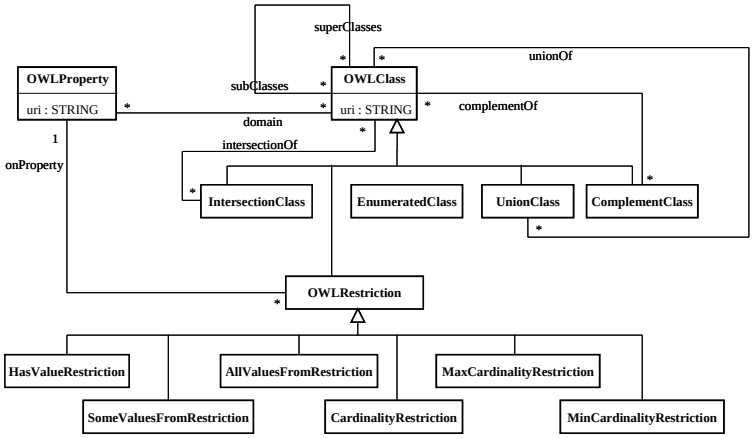


Figure 3: Méta-modèle simplifié de OWL

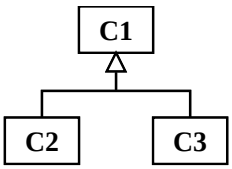


Figure 4: Structure de l'union de classes

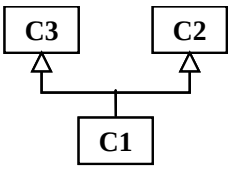


Figure 5: Structure de l'intersection de classes

vante [1] :

```
CREATE #UnionClass SchoolMember ('schoolmember') AS unionOf
(Professor, Student);
CREATE EXTENT OF SchoolMember AS unionOfInstances (Professor,
Student);
```

5. SUPPORT DES CONCEPTS NON CANONIQUES DANS ONTODB

Avec l'extension proposée précédemment, OntoDB peut supporter l'ajout de nouveau opérateurs pour définir la sémantique comportementale des concepts ontologiques non canoniques. Dans cette section, nous présentons la mise en oeuvre de trois concepts non canoniques : `UnionClass`, `IntersectionClass` et `HasValueRestriction`.

5.1 Support de l'union de classes

`UnionClass` représente une classe obtenue à partir de l'union d'un ensemble de classes OWL. Par exemple, si on considère que la classe C1 est l'union des classes C2 et C3 *(C1 = C2 U C3)*, les individus de C1 sont l'union des individus de C2 et C3 *(Instances(C1) = instances(C2) U instances(C3))*. De plus, C1 devient une super-classe de C2 et C3 : *sous-classes(C1)* = {*C2, C3*} (Figure 4). La requête OntoQL suivante permet d'étendre le méta-schéma d'OntoDB avec le concept `UnionClass` :

```
CREATE ENTITY #UnionClass UNDER #OWLClass(#unionOf REF (#OWL-
Class) ARRAY);
```

Afin de montrer la chaine complète d'une union de classes, nous créons deux classes OWL (Professor et Student) :

```
CREATE #OWLClass Professor('professor');
CREATE #OWLClass Student('student'));
```

La requête OntoQL permettant de faire l'union de classes est :

```
CREATE #UnionClass SchoolMember('schoolmember') AS unio-
nOf(Professor, Student);
```

Cette requête OntoQL modifie la structure des classes `SchoolMember`, `Professor` et `Student`. En effet, `SchoolMember` devient une super-classe de `Professor` et de `Student`. La requête permettant de calculer les individus de `SchoolMember` est :

```
CREATE #EXTENT OF SchoolMember AS unionOfInstances(Professor,
Student);
```

5.2 Support de l'intersection de classes

`IntersectionClass` représente une classe obtenue à partir de l'intersection d'un ensemble de classes OWL. La classe qui en résulte devient une sous-classe de toutes les classes qui participent à l'intersection. La Figure 5 illustre le concept `IntersectionClass` avec un exemple d'une classe C1 qui est une intersection de C2 et C3. Nous remarquons que C1 devient une sous-classe de C2 et C3 (*superClasses(C1)* =

1. Pour simplifier, nous utilisons les noms de classes au lieu des uri dans toutes les requêtes qui suivent.

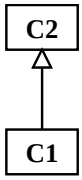


Figure 6: Structure d'une restriction de classe

{*C2, C3* }). Par conséquent, C1 hérite des propriétés de C2 et C3. La requête OntoQL permettant d'étendre le méta-schéma d'OntoDB avec le concept `IntersectionClass` est :

```
CREATE ENTITY #IntersectionClass UNDER #OWL-
Class(#intersectionOf REF (#OWLClass) ARRAY);
```

Nous considérons que la classe `StudentEmployee` est l'intersection des classes `Student` et `Employee`. la requête OntoQL autorisant l'éxecution de l'intersection est :

```
CREATE #IntersectionClass StudentEmployee('studentemployee')
AS intersectionOf (Student, Employee);
```

5.3 Support de restriction de classe

`HasValueRestriction` correspond à une classe obtenue en restreignant une classe OWL à une valeur d'une de ses propriétés. La classe qui en résulte devient une sous-classe de la classe sur laquelle nous appliquons la restriction. Par exemple, si C1 est une restriction de la C2 sur une propriété p, C1 devient une sous-classe de C2 : *superClasse (C1) = C2* (Figure 6), et les instances de C1 sont celles de C2 avec une valeur fixe pour la propriété à laquelle la restriction est appliquée (p). La requête OntoQL permettant d'étendre le méta-schéma d'OntoDB avec le concept `HasValueRestriction` est :

```
CREATE ENTITY #HasValueRestriction UNDER #OWLClass(#onPro-
perty REF (#OWLProperty));
```

Nous considérons que la classe `MaleSM` est définie comme une restriction de la classe `SchoolMember` sur la propriété sexe dont la valeur est égale à 'M' :

```
CREATE #HasValueRestriction MaleSM('malesm', SchoolMember,
sex);
```

Les individus appartenant à `MaleSM` sont obtenus avec la requête suivante :

```
CREATE EXTENT OF MaleSM AS hasValueRestriction(1080) WHERE
sex = 'M';
```

L'implémentation des autres constructeurs de concepts non canoniques est similaire à celle que nous venons d'exposer.

6. CONCLUSION

Dans cet article, nous avons présenté notre travail sur l'extension des bases de données à base ontologique et de leurs langages d'exploitation afin de supporter l'expression de la sémantique comportementale des concepts ontologiques non canoniques. Notre objectif est d'exprimer la sémantique comportementale des concepts non canoniques du modèle OWL à l'aide d'opérateurs définis dynamiquement à l'intérieur ou à l'extérieur de la BDBO. Pour répondre à ce besoin, nous avons d'abord soulevé le problème du support de la sémantique comportementale dans OntoDB. Ensuite, nous avons proposé une extension de ce système afin d'autoriser la définition de nouveaux opérateurs. Ces opérateurs peuvent être implémentés par des programmes externes (définis à l'extérieur de la base de données) ou par des invocations de services web. Nous avons aussi étendu le langage d'exploitation associé à OntoDB, OntoQL avec la possibilité de définition et d'appels d'opérations. Pour illustrer l'intérêt de ces opérateurs, nous avons présenté l'extension de la partie méta-schéma de la BDBO OntoDB avec trois constructeurs de concepts non canoniques du modèle OWL. Comme perspectives de notre travail, nous comptons exploiter les capacités de méta-modélisation offertes par le système OntoDB/OntoQL pour faire des transformations de modèles en base de données ou vérifier la cohérence des données de la BDBO.

7. REFERENCES

[1] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite : Managing Voluminous RDF Description Bases. In *Proceedings of the 2nd International Workshop on the Semantic Web*, pages 1–13, 2001.

[2] A. Borgida and R. J. Brachman. Loading data into description reasoners. *SIGMOD Record*, 22(2) :217–226, 1993.

[3] D. Brickley and R. V. Guha. *RDF Vocabulary Description Language 1.0 : RDF Schema*. World Wide Web Consortium, 2004. http ://www.w3.org/TR/rdf-schema.

[4] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame : A Generic Architecture for Storing and Querying RDF and RDF Schema. In I. Horrocks and J. Hendler, editors, *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, number 2342 in Lecture Notes in Computer Science, pages 54–68. Springer Verlag, July 2002.

[5] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena : Implementing the Semantic Web Recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters (WWW'04)*, pages 74–83, New York, NY, USA, 2004. ACM Press.

[6] E. I. Chong, S. Das, G. Eadon, and J. Srinivasan. An Efficient SQL-based RDF Querying Scheme. In *Proceedings of the 31st international conference on Very Large Data Bases (VLDB'05)*, pages 1216–1227, 2005.

[7] M. Dean and G. Schreiber. *OWL Web Ontology Language Reference*. World Wide Web Consortium, 2004. http ://www.w3.org/TR/owl-ref.

[8] H. Dehainsala, G. Pierra, and L. Bellatreche. Ontodb : An ontology-based database for data intensive applications. In *Proc. of the 12th Int. Conf. on Database Systems for Advanced Applications (DASFAA'07). LNCS*. Springer, 2007.

[9] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies (IJHCS)*,

43(5-6) :907–928, 1995.

[10] S. Harris and N. Gibbins. 3store : Efficient bulk RDF Storage. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PPP'03)*, pages 1–15, 2003.

[11] P. Hayes. *RDF Semantics*. World Wide Web Consortium, 2004. http ://www.w3.org/TR/rdf-mt/.

[12] S. Jean, Y. Aït-Ameur, and G. Pierra. Querying Ontology Based Database Using OntoQL (an Ontology Query Language). In *Proceedings of On the Move to Meaningful Internet Systems 2006 : CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences (ODBASE'06)*, volume 4275 of *Lecture Notes in Computer Science*, pages 704–721. Springer, 2006.

[13] J. Lu, L. Ma, L. Zhang, J.-S. Brunner, C. Wang, Y. Pan, and Y. Yu. Sor : a practical system for ontology storage, reasoning and search. pages 1402–1405, 2007.

[14] J. Mei, L. Ma, and Y. Pan. Ontology query answering on databases. In *Proceedings of the 5th International Semantic Web Conference (ISWC'06)*, pages 445–458, 2006.

[15] Z. Pan and J. Heflin. DLDB : Extending Relational Databases to Support Semantic Web Queries. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pages 109–113, 2003.

[16] M. J. Park, J. H. Lee, C. H. Lee, J. Lin, O. Serres, and C. W. Chung. An Efficient and Scalable Management of Ontology. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, volume 4443 of *Lecture Notes in Computer Science*, pages 975–980. Springer, 2007.

[17] J. Petrini and T. Risch. SWARD : Semantic Web Abridged Relational Databases. In *Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA'07)*, pages 455–459, 2007.

[18] G. Pierra. Context Representation in Domain Ontologies and its Use for Semantic Integration of Data. *Journal Of Data Semantics (JODS)*, X :34–43, 2007.

[19] G. Pierra and E. Sardet. *ISO 13584-32 âĂŞ Industrial automation systems and integration âĂŞ Parts library âĂŞ Part 32 : Implementation resources : OntoML : Product ontology markup language*. ISO, 2010.

[20] R. Volz, S. Staab, and B. Motik. Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases. *Journal of Data Semantics II*, 3360 :1–34, 2005.