OntoDB2 : Support of Multiple Ontology Models within Ontology Based Database

Chimene Fankam
LISI/ENSMA - Poitiers University
86961 Futuroscope Chasseneuil Cedex France
fankamc@ensma.fr

ABSTRACT

The notion of ontology has became more widespread, its usage in different domains has first led to proposition of different ontology models, and secondly, to implementations of different architectures able to manage both ontologies and associated data. We called such database architectures Ontology-Based DataBases (OBDBs). It appears that each OBDB is mainly based on a single ontology-model. During our investigation of the most existing architectures and ontology models, we figure out that most of these models share a common kernel and each one has additional constructs which are orthogonal. In this paper, we propose a new methodology to design an OBDB supporting different ontology-models. This architecture offers specific constructs of existing ontology models.
Advisors : Guy Pierra, Ladjel Bellatreche

Keywords

Ontology, Ontology model, Ontology-based Databases, Architecture

1. MOTIVATIONS

Ontologies are used in a large spectrum of domains by different communities. For example, in database area, to facilitate data exchange and integration [6, 15], in the Semantic Web field, to describe terms, retrieve information and interconnect Web services. With the increasing use of ontologies, a number of ontology models and languages with different formalisms have been proposed: RDF [18], RDFS [9], OWL [5], PLIB [25], KIF [14]. Each model (and language) has its favorite application domain. For instance, PLIB ontology model has been largely used for sharing and integrating heterogeneous data sources [6] in particular in the engineering domain [19], RDF and OWL for annotating documents [17, 1]. Recently, several works proposed to represent both ontologies and associated data in databases. A database offering this functionality is called an Ontology-based Database (OBDB).

Several OBDBs have been proposed such as: RDFSuite [4], SESAME [10], OntoMS [23], OntoDB [12]. The main characteristic of these OBDBs is that they are built around *only one single ontology model*. Several debates have argued that constructs proposed by each ontology model are complementary [20, 11, 24, 13, 26]. Many studies [16, 21, 8] have also being conducted showing (a) the feasibility to combine the semantics underlying Web Semantic ontology models and database semantics, and (b) the added value that would result for data intensive applications.

My Ph.D work concentrates on the development of an OBDB supporting constructs of multiple ontology models. To well conduct my research work, a road map has been defined. (1) To perform a precise study of existing ontology models and of different architectures of OBDB. (2) To define how specific constructs of different ontology models could be implemented in the same architecture. In my work, two standard ontology models are to be used: OWL and PLIB. (3) To design, implement and validate the proposed OBDB architecture, called OntoDB2. This task also includes an ontology editor designed for transparently managing ontologies with different models.

This paper is organized into five sections. Section 2 presents an overview and a classification in three main categories of existing OBDBs, followed by a study of the limitations of theses OBBDs and the need of a new OBDB. Section 3 presents our findings on which is based our strategy for defining an architecture that supports multiple ontology models. Section 4 presents specifications and implementation issues of OntoDB2. Section 5 presents our current progress and our ongoing work.

2. A PROPOSED TAXONOMY OF OBDB

An important number of OBDBs were proposed in the literature. We suggest classifying them into three main categories.

2.1 Type I architecture

In this architecture, ontology and associated data are stored in an *unique schema* (Figure 1). Jena1 [7] is an example of this architecture, where the used storage schema consists of a unique table having the RDF triples structure: *(subject, predicate, object)*. In this representation, there is no separation between ontologies and data. The database structure is frozen. This representation is very simple, since insertion/deletion operations of properties and instances are

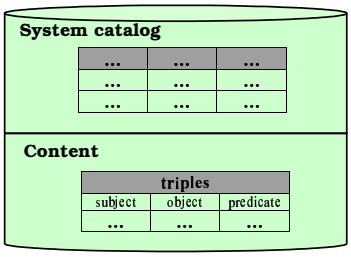

Figure 1: Schema of an OBDB of type I

done easily. But, it suffers from weak data typing and poor performance caused by several auto-join operations over the unique table [12, 23]. To optimize this architecture, clustering techniques need to be used [3]. This may dramatically cause maintenance overhead. Moreover, the ontology model being implicit, it needs to be hard encoded in the query language interpreter.

2.2 Type II architecture

In type II architecture, ontology and associated data are stored into *two different schemas* (Figure 2), one for storing the local ontology and another for instance data. This architecture outperforms the above one, but it still has some drawbacks: (1) the ontology schema is based on the underlying ontology model and thus is static, and, (2) introduction of concepts originated from other ontology models is not allowed. In Sesame, for example, structure of the ontology

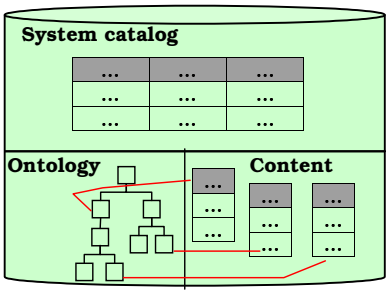

Figure 2: Schema of an OBDB of type II

part is based on RDFS (tables include: *class, property, domain, range, etc.*), whereas different representations can be used for the data part: (1) An unique table of triples (like in type I architecture), which contains extensions of all concepts (classes and properties) of the local ontology. (2) A unary distinct table for each class of the ontology and a binary table for each property of the ontology. In this approach, the management of ontology part and data part is different. This architecture is more efficient [4]. The second data representation scales quite well, especially, when queries refer to a small number of properties [2]. Contrariwise, when each instance is described by a large number of properties, it does not scale [12].

2.3 Type III architecture

This architecture is proposed in OntoDB [6, 25], with PLIB as the underlying ontology model. An additional part, called, the *meta schema part* is introduced (Figure 3). Thus the database structure is defined by *three schemas*. The presence of the meta schema part offers flexibility of the ontology part, since it is represented as an instance of the meta schema. This approach offers a generic access to both ontology and data. Whereas different representations could be used for the data part, in OntoDB an horizontal approach was used where a single table is associated to each class of the ontology with one column per each used property. This architecture scales well compared to type I architecture. It is also more efficient than type II architecture, when numerous properties per instances are used [12].

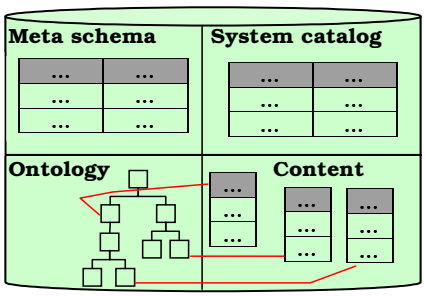

Figure 3: Schema of an OBDB of type III

2.4 Limits of existing solutions

A number of studies have pointed out the need for ontology-based applications (in various domains) to integrate features from several ontology models or from others modeling formalisms [21, 13]. It is the case, for example of e-business related applications, where there is a need for a large expressivity for describing domain information (like in OWL) and a capability for defining integrity constraints to ensure data quality (like in PLIB). Note that these applications are strongly related to Web Semantic domain, but they cannot totally be modeled and interpreted using only RDFS/OWL-oriented OBDBs (which do not support constraint definition), or a PLIB-oriented OBDB (which does not offer support for inverse property or class expression). Thus, if we consider the case of a PLIB-oriented OBDB, constraint capability will be satisfied while loosing description expressivity that description logics-based ontology models offer.

This example shows that those applications need to deal with several kinds of features which are unfortunately not available in a single ontology model. Consequently, there is a real need to integrate a set of ontology models.

As previously observed, existing OBDBs only deal with a single ontology model (RDFS, OWL, PLIB,...), or, with possible semantics-compatible ontology models (RDFS/OWL). Therefore, none of them is a good candidate for fields, where applications require constructs from ontology-models with different semantics. Our proposition is to design an OBDB architecture aiming at providing a more adequate environment, i.e, based on an ontology model integrating constructs from several ontology models like RDFS, OWL and PLIB. The architecture must also manage very large size ontology-based data and provides an efficient environment to the common operations or queries on these ontologies and associated data.

2.5 Summary

In order to provide an effective OBDB for fields, where applications need to integrate constructs from different ontology models, we propose to design an OBDB based on an ontology model flexible enough to allow expression and processing of complementary constructs from several ontology models. Based on the above classification of OBDBs, it appears that type III architecture is more flexible. Indeed, in type I and type II architectures, the ontology part is fixed, whereas in type III, the ontology part is defined according a meta schema. This flexibility criteria is a key success factor in our goal to support multiple ontologies. Also, in type III, the management of the ontology part and the data part is done in a homogeneous manner and can be expressed through their upper level. This allows a better processing of operations/queries and more efficiency in handling large amount of data. Based on this analysis, type III architecture seems a good candidate for our proposed approach. Table 1 summarizes advantages and disadvantages of the studied architectures.

	Type I	Type II	Type III
Strong typing	no	no/yes	yes
Separation ontology/data	no	yes	yes
Model evolution	no	no	yes
Scalability	-	++	++

Table 1: A comparison of OBDB architectures

3. FINDINGS AND STRATEGIES

In [13], we studied and analyzed different ontology models including the standard for the Semantic Web (OWL), the standard for engineering (PLIB) and other models and languages (KIF, F-logic) used in various applications. This study allowed us to identify three fundamental characteristics shared by these models.

1. In all ontology models, domain ontologies are in particular described in terms of primitive (canonical) classes and properties associated with datatypes.
2. Most of the specificities of each ontology model consist in defining conceptual equivalences (non-canonical concepts) over primitive concepts. Moreover conceptual equivalences operators (e.g. class expression for OWL and property rules in F-logic) are orthogonal and can co-exist without any conflict in the same architecture.
3. Instances of non-canonical concepts (defined concepts) may also be represented as instances of canonical concepts (primitives concepts).

Based on our findings, we defined a five step strategy to design an architecture able to support multiple ontology models :

1. Determine a common kernel (representing the intersection of ontology models) and its enrichment to suitably integrate the specification of the various ontology models.
2. Define how to integrate conceptual equivalences around the common kernel.
3. Propose a policy for representing and managing instances of data in the database. This policy shall take into account the fact that non-canonical concepts can be computed (derived) from their representation as canonical concepts and their materialization in the database is not necessary.
4. Support the extension of the ontology model.
5. Ensure the completeness of reasoning operations.

In our work, we currently deal with the following ontology models PLIB, RDFS OWL Lite and OWL DL. OWL Full is not decidable and is not concern by the scope of our study. In the remaining paper, we will refer to OWL DL and / or OWL Lite using the term OWL.

3.1 Specification of the common kernel

The common kernel is composed of concepts which are shared by all the studied ontology models. In particular, RDFS, OWL and PLIB ontology models describe domains in terms of classes and properties which characterize classes whose range is defined by a data type (that may be a class). Properties may have a collection has values. A number of meta data are used to describe concepts (class and property) of the common kernel (e.g., definition, comment, etc.).

To ensure an efficient implementation of ontology model and instance data, it was decided to introduce some constraints like in OntoDB [12] and RDFSuite [4]. (1) A property must have a unique domain and a unique range. This enables strong typing of property values. (2) Multi instantiation is not supported in general. The set of classes to which an instance belongs has a minimum for the subsumption relationship (least subsumer). This class is called the instance base class. (3) Only properties that are applicable to an ontology class (i.e., whose domain includes the class) may be used for describing its instances. It is worth to notice that if only properties applicable to a class may be used for describing its instances, there is no constraint that all applicable properties should be used. Thus, the logical schema of instances of a class may be a subset of all the properties applicable to this class.

3.2 Expression of conceptual equivalences

Most ontology models add specific constructs to the common kernel that allow defining conceptual equivalences between concepts. These extensions allow defining new constructors on the basis of pre-existing canonical concepts using various operators (set operators, logical rules, algebraic expressions, and algebraic characteristics). We call theses concepts "non-canonical" concepts. Since these concepts are defined from canonical concepts, their instances may be represented as instances of canonical concepts.

3.3 Instances representation

As we have seen in section 2, the best representation of instances data depends on the application domain (e.g., the number of properties values associated with each instance). To leave open the choice of instance implementation, we

choose to associate to each concept (class) of ontology a view which will correspond to its set of instances. For a canonical concept, this view will be in fact the table or set of tables that contains its instances, and, for a non-canonical concept, this view will be an expression corresponding to the definition of the concept.

3.4 Extension of the ontology-model

The used common kernel must support extension. Therefore, the architecture of our OBDB must support mechanisms for extending or specializing the common kernel. The type III architecture satisfies this requirement. Indeed, it is a Model-Driven Architecture similar to that of the Meta-Object Facility (MOF). By the means of reflexive meta schema, it offers a generic access to the ontology part enabling to support its evolution.

3.5 Reasoning operations

In order to ensure the coherence and the completeness of the database while maintaining good performances, much of the reasoning operations will be done within the architecture using SQL and processing offers by DataBase Management System (DBMS). For the others, for which the inference cannot be efficiently processed in the database, a pre-processing must be done prior to the insertion of data whether at the ontology or the data part, so that the result of the pre-processing allows to easy the exploitation of the inserted data. In OBDB, reasoning operations can be organized into two groups: those concerning the ontology part and those concerning the data part.

1. For the ontology part, OBDBs mostly deal with classification in order to compute the subsumption hierarchy and instantiation to determine the type of a given individual.

   - **Subsumption.** All OBDBs allow a hierarchical visualization of classes of ontologies. As canonical classes, defined or non-canonical classes must also be computed and positioned in the subsumption hierarchy in order to provide a render of the full ontology content. The classification operation is used to check the relation of generalization/specialization between concepts, to organize and to classify concepts in a hierarchy.
   - **Instantiation.** This operation enables to find to which class(es) an instance may belong, and is necessary while migrating instances from an OWL ontology to the OBDB, particularly for derived classes which are essentially views; their instances will be stored as instances of their least primitive(s) subsumer(s).

   Such mechanisms will be pre-processed using external inference engines. The results are then stored in the database which becomes ready for supporting queries.

2. For the data part, inferences are usually related with non-canonical operators: defined classes and logical characteristics of properties (symmetric, inverse, transitive). The next section discusses how reasoning on the data part can be achieved using SQL and DBMS processing.

4. OUR CURRENT SPECIFICATION

4.1 The common kernel

All common characteristics of PLIB, RDFS and OWL represent the kernel of our ontology model. In addition to these concepts, we add characteristics which are usually important for applications like algebraic characteristics of properties (inverse, symmetric, transitive). We also introduce features for the architecture optimization (class labelling) and to easier the integration of future extension.
Under the assumptions of section 3.1, the common kernel can then be defined as follows: *an ontology is made of a set of classes organized in a subsumption hierarchy. Classes are characterized by a set of properties associated with data types. A property may have a simple range or be a collection. Classes and properties are associated with logical characteristics which are expressed in term of constraints, axioms or rules. They are associated with meta data. Individuals or ontology-based data are defined in terms of concepts (classes or properties) defined in ontologies. Lastly, the common kernel must contain only canonical concepts.*

4.2 Extensions and mapping operators

`Observation :` Specific constructs offer by the different ontology models over the common kernel are made using conceptual equivalence operators on canonical and/or non-canonical concepts. These operators include: (1) Boolean expressions, (2) algebraic characteristics, (3) restrictions on property range, etc. Therefore, their instances can be derived from those of canonical concepts, since DBMS offers similar mechanisms and operators.
So, as canonical concepts, non-canonical concepts will be represented in the database, the major difference resides on the representation and the manipulation of associated data.

4.2.1 Non-Canonical Classes

1. Non-canonical classes defined using Boolean expressions
   `Example :`
   Airdale Terrier $\equiv$ *Old-English-Terrier* $\cap$ *Otterhoung*
   This example defines a non-canonical class "Airdale Terrier" whose individuals must both belongs to "Old-English-Terrier" and "Otterhoung". From this definition, the following database view exactly computes the extension of this non-canonic class.
   ```
   SELECT * FROM Old-English-Terrier
   INTERSECT
   SELECT * FROM Otterhoung
   ```
   This example shows that the extension of non-canonical classes are not stored, but computed starting from the evaluation of their definition by creating views over extensions of canonical classes from which they are built. Thus, non-canonical classes can be compared to external schema on canonical classes, offering then reasoning capabilities and local mapping enriching the expressiveness of the ontology model.

2. Non-canonical classes defined using restrictions
   `Example :`
   French dogs $\equiv \exists$ *comes-from. 'France'*
   This example defines a non canonical class "French dogs" as the class of individuals whose the value for the property "comes-from" has the value 'France'. Based on this definition and according to our assumptions on

the uniqueness and the presence of the domain of each property, the extension of the latter class is computed by the following view:
```
SELECT * FROM dogs WHERE comes-from = 'France'
```
Thus, the representation of classes defined using restriction is also carried out by preserving the non-redundancy of the data in the OBDB.

In OWL, three types of restriction are available: value restrictions, range restrictions and cardinality restrictions. These restrictions are respectively related to: the value of a property, the type of the values allowed for a property and, the maximum and/or the minimal cardinality of the values of a property for an occurrence of a given class. As for the value restriction of the previous example, a range restriction can be compared with a selection in other to filter individuals for whose the value(s) of the property involved in the restriction is of the type of the specified type, while, a cardinality restriction can be compared with a selection in other to filter individuals for whose the value(s) of the property involved in the restriction respects the specified cardinality.

4.2.2 Non-Canonical Properties

1. Inverse properties
   Due to the importance of inverse characteristic, we have integrated it in the common kernel. It is usually used for bi-directional associations between classes. The value of a property defined as being the inverse of another property can be calculated using a view. The expression of this view must take into account the cardinalities of the property and that of its inverse. Some systems, like Hibernate, currently implement this characteristic on traditional databases.

2. Properties defined by an algebraic expression
   Example : *diameter = radius * 2*
   This example defines the value of the property diameter of a given occurrence as the double of its radius. From this definition, the following database view computes the value of diameter for any instance in the database :
   ```
   SELECT (radius *2) AS diameter FROM ...
   ```
   In most traditional DBMSs, data manipulation language offers support for arithmetic operators as well as operators on strings. Thus, properties defined by expressions using, for example, arithmetic operators and whose operands included other properties can currently be expressed using views.

3. Algebraic characteristics of properties
   Some properties carry characteristics which make it possible to completely derived their whole extent starting from a subset of the latter. It is the case of OWL symmetric and transitive properties.

   - *Symmetric.*
     Observation: given a property P, the symmetric characteristic allows to deduce for an occurrence A related to another occurrence B by the property P, that the occurrence B is related to the occurrence A by this same property.
     It is thus possible for this characteristic to automatically fill the database by exploiting *triggers* offers by DBMS. This mechanism is simple to implement.
   - *Transitive.*
     Observation : Given a property P, the transitive characteristic allows to deduce given an occurrence A related to another occurrence B by the property P, that at the establishment of a new association between the occurrence B and another occurrence C by the property P, a new relation between A and C by property P must be add to the database. Moreover, the same applies to any occurrence related to B by P.
     The value of a transitive property will also be filled using a trigger. However, this mechanism is more complex to implement: (1) the computation of the transitive closure has an elevated cost since it is recursive, (2) if one considers the example of the observation above, if the relation between B and C is removed, it would also be necessary to remove the relation between A and C. But without any information on the origin of the A to C, it is not possible to know whether this relation should be removed.
     In order to efficiently manage transitive relationship, we considered that the database content is saturated for the property P before the insertion of a new relation between B and C, the transitive closure of P is then calculated after insertion in a non recursive way. Indeed, we just have to relate any individual related to B, with C. The corresponding trigger is expressed as follow:
     ```
     FOR EACH A IN
     SELECT oid FROM ...  WHERE P = NEW.oid
     LOOP
     INSERT INTO ...(oid, P)VALUES(A.oid, NEW.P);
     END LOOP;
     ```

4.3 Discussion

This clause presents a discussion about how our suggested solution is different, new and better compared to existing approaches. The discussion surrounds three major characteristics of our approach.

- **Supporting multiple ontology models.** All the existing OBDBs [4, 10, 23, 22, 12] support only one particular ontology model. The finding that all ontology models: (a) share a common kernel, and (b) include extensions that may be compatible with each other, allowed us to propose an architecture that may both support various ontology models and integrate the capabilities of several models. This is done by (1) representing, in the architecture, the meta model level, (2) representing directly a kernel ontology model, that represents the common semantics of the various ontology models, and (3) defining mechanisms allowing to map specific ontology constructs on database mechanisms such that triggers and views.

- **Managing canonical data.** In database, information shall be represented in a single (canonic) way.

Contrariwise, ontologies allow to describe the same piece of information by different data [13, 8, 21]. For instance, a woman may be represented either as a *woman* instance, or as a *person* instance whose *gender* is *female*. Such data do not fulfil the closed world assumption and thus, any query may require inferencing capabilities. By proposing to represent any instance of non canonical class (defined concept) as an instance of the canonical class (primitive concept) to which it belongs, the represented data fulfil the closed world assumption, and any canonical class may be queried by usual query language without any inferencing capabilities.

- **Using database mechanisms for reasoning over ontological data.** An important inference capability of description logic-based ontology, called the instantiation test, is to compute whether some instance belongs to some class. Representing all information as canonical information, and representing all non-canonical classes as views over canonical classes allows to compute both the instances of canonical class and the instances of non canonical class by means of the database query language.

5. ONGOING IMPLEMENTATION

5.1 Work to date

A prototype of OntoDB2 is under implementation. The framework of our approach is depicted as Figure 4. The overall design objectives of our system include :

1. *Scalability and flexibility.* Our implementation is based on the OntoDB operational prototype of OBDB [27], OntoDB2 thus benefits from its scalability [12]. Also, the ontological part allows the specialization of the used kernel ontology model and thus is appropriate for the implementation of all the specifics extensions of the various existing ontology models.

2. *Portability.* In order to ensure the portability of our system with respect to the underlying DBMS, the various accesses to database are made using Hibernate, through mappings files we have defined between the object representation of our ontology model and its database schema representation. Actually, we are using PostgreSQL relational-object database system for our implementation. The migration of our implementation from PostgreSQL to another DBMS requires at most to replace mapping files and, at best no change.

3. *Accessibility.* All OBDBs offer Application Programming Interface (API) for accessing and manipulating ontologies and associated data contained in the database. This allows the extraction of the underlying data schema for applications accessing to data. In our case, the architecture provides API to access data in a transparent way. The manipulation of ontologies and associated data may be done in their source model, regardless their representations in the "common kernel" ontology model of the OBDB. Importation/Exportation mechanisms from and to their source model are to be provided. Our APIs are encoded in the JAVA language. Model specific packages for OWL and PLIB are under development, these packages encode the mapping rules between PLIB and OWL ontologies [13] and their representation in the common kernel. We also provide a Web editor called *OntoWeb* which allows visualisation and manipulation of ontologies and associated data. At this stage, this interface is only available for the common kernel and for PLib specific extensions.

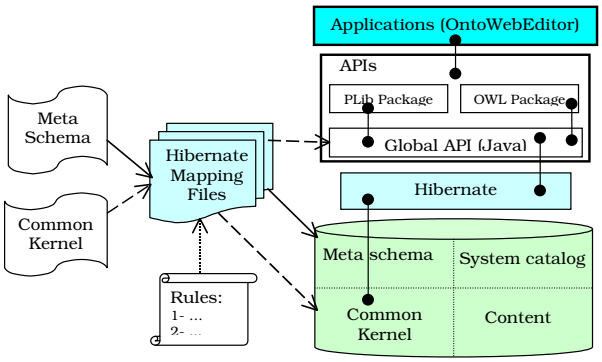

Figure 4: Global architecture of OntoDB2

5.2 Future work

We plan to further work on the following aspects:

- **Automatic generation of non-canonical classes views.** OntoDB2 must be associated with an external module in charge of generating views from non-canonical classes descriptions.

- **Preprocessing of ontology and ontology-based data.** To allow both view generation and canonic representation of instances for non canonical classes, complete subsumption hierarchies and complete instance representations need to be computed by inference engines. Thus, we have to define how information will be shared and transferred between the OBDB and external engines.

- **Algebraic expression module.** In order to enable the definition of properties using algebraic expressions, it is necessary to : (1) enrich the ontology editor with an expression builder and compiler and (2) study the possibility of invoking external functions (in charge of the evaluation of these expressions) in views. This will allow us to support a larger set of operators in addition to the mathematical operations and functions offered by SQL and existing commercial database systems.

- **Constraint checking.** We plan to support a ontological constraint language in order to maintain the consistency of the database by expressing and checking ontological constraints against instance data.

6. ACKNOWLEDGMENTS

This work has been partially supported by the French ANR under grant ANR05RNTL02706 (e-Wok-Hub).

7. REFERENCES

[1] RDFa Primer : Embedding Structured Data in Web Pages. *W3C, http://www.w3.org/TR/xhtml-rdfa-primer/*, 2007.

[2] D. J. Abadi, A. Marcus, S. Madden, and K. J. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422, 2007.

[3] R. Agrawal, A. Somani, and Y. Xu. Storage and querying of e-commerce data. In *VLDB*, pages 149–158, 2001.

[4] 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*, 2001.

[5] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. Owl web ontology language reference. *W3C, http://www.w3.org/TR/owl-ref/*, 2004.

[6] L. Bellatreche, G. Pierra, D. Nguyen Xuan, H. Dehainsala, and Y. Ait Ameur. An a priori approach for automatic integration of heterogeneous and autonomous databases. *International Conference on Database and Expert Systems Applications (DEXA'04)*, (475-485), September 2004.

[7] B.McBride. Jena: Implementing the rdf model and syntax specification. *Proceedings of the 2nd International Workshop on the Semantic Web*, 2001.

[8] A. Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[9] D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. *W3C, http://www.w3.org/TR/rdf-schema/*, 2002.

[10] 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.

[11] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. Owl flight. *WSML Delivrable D20.3 v0.1, http://www.wsmo.org/TR/d20/d20.3/*, 2004.

[12] H. Dehainsala, G. Pierra, and L. Bellatreche. Ontodb: An ontology-based database for data intensive applications. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, Lecture Notes in Computer Science, pages 497–508. Springer, 2007.

[13] C. Fankam, Y. Ait-Ameur, and G. Pierra. Exploitation of ontology languages for both reasoning and persistency purposes: mapping plib, owl and flight ontology models. In J. Filipe, J. Cordeiro, B. Encarnação, and V. Pedrosa, editors, *Third International Conference on Web Information Systems and Technologies(WEBIST'07)*, pages 254–262. INSTICC Press, March 2007.

[14] M. Genesereth. Knowledge Interchange Format. In *Proceedings of the SecondInternational Conference on the Principles of Knowledge Representation and Reasoning*, pages 238–249. Morgan Kaufman Publishers, 1991.

[15] C. Goh, S. Bressan, E. Madnick, and M. D. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–293, 1999.

[16] S. Grimm and B. Motik. Closed-world reasoning in the semantic web through epistemic operators. In *In CEUR Proceedings of the OWL Experiences and Directions Workshop*, 2005.

[17] J. Kahan and M. Koivunen. Annotea: an open rdf infrastructure for shared web annotations. In *In proceedings of the 10th international conference on World Wide Web*, pages 623–632, 2001.

[18] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. *W3C, http://www.w3.org/TR/rdf-concepts/*, February 2004.

[19] L.Bellatreche, D. N. Xuan, G. Pierra, and H. Dehainsala. Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases. *Computers in Industry*, pages 711–724, 2006.

[20] B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and logic programming live together happily ever after? In *in proceedings of the 2006 International Semantic Web Conference (ISWC'06)*, volume 4273 of *Lecture Notes in Computer Science*, pages 501–514. Springer, 2006.

[21] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between owl and relational databases. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 807–816. ACM, 2007.

[22] 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.

[23] 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*. Springer, 2007.

[24] P. F. Patel-Schneider and I. Horrocks. A comparison of two modelling paradigms in the semantic web. In *Proceedings of the Fifteenth International World Wide Web Conference (WWW'06)*, pages 3–12. ACM, 2006.

[25] G. Pierra. Context representation in domain ontologies and its use for semantic integration of data. *Journal Of Data Semantics (JODS)*, pages 173–210, 2008.

[26] G. Pierra. Context representation in domain ontologies and its use for semantic integration of data. *Journal Of Data Semantics (JODS)*, pages 173–210, pp. 2008.

[27] G. Pierra, H. Dehainsala, Y. Ait-Ameur, L. Bellatreche, J. Chochon, and M. E.-H. Mimoune. Bases de données a base ontologique. Principe et mise en oeuvre. *Ingénierie des Systèmes d'Information (ISI'05)*, 10(2):91–115, 2005.