*L*ABORATOIRE D'*I*NFORMATIQUE *S*CIENTIFIQUE ET *I*NDUSTRIELLE

Rapport de recherche

N° 03 -2006

Managing Instance Data in Ontology-based Databases

Hondjack DEHAINSALA, Guy PIERRA, Ladjel BELLATRECHE,

ÉCOLE NATIONALE SUPÉRIEURE DE MÉCANIQUE ET D'AÉROTECHNIQUE
Site du Futuroscope - B.P. 109 - 86960 FUTUROSCOPE Cedex - FRANCE
Tél. : +33 (0) 5-49-49-80-63 - FAX : +33 (0) 5-49-49-80-64
Web : http://www.lisi.ensma.fr

Managing Instance Data in Ontology-based Databases

Hondjack Dehainsala[1], Guy Pierra[1], and Ladjel Bellatreche[1]

LISI/ENSMA, Téléport 2, 1, ave. Clément Ader 86960 Futuroscope - France
(dehainsala, pierra, bellatreche)@ensma.fr

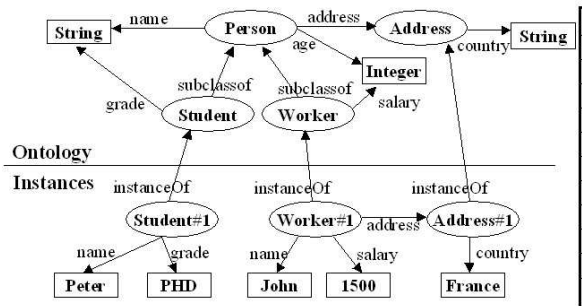
Abstract. Recently, several approaches and systems were proposed to store in the same repository data and the ontologies describing their meanings. We call these databases, ontology-based databases (OBDBs) and ontology-based data represents those data that represent ontology individuals (i.e., instance of ontology classes). To ensure a high performance of queries on top of these OBDBs, representation of ontology-class instance, called, ontology-based data becomes a new challenge. Two main representation schemes have been proposed for ontology-based data: vertical and binary representations with a variant called hybrid. All these schemes store in separate tables, instances and their properties. In this paper, we propose a new representation of ontology-based data, called *table per class*. It consists in associating a table to each ontology class, where all property values of a class instance are represented in the same row. Columns of this table represents those rigid properties of the ontology class that are associated with a value for at least one instance of this class. In the following, we present the context of our project, the architecture we have developed for ontology-based databases, and a comparison of the effectiveness of our representation of ontology-based data with the classical representations used in Semantic Web applications. Our benchmark involves three categories of queries: (1) targeted class queries, where users know the classes they are querying, (2) no targeted class queries, where users do not know the class(es) they are querying, and (3) update queries.

1 Introduction

Nowadays, ontologies are largely used in several research and application domains such as Semantic Web [19], information integration [18], e-commerce [14], data warehousing [7], etc. This is due to their nice characteristics: consensual, formal, shared, etc. Actually, several tools for managing (building, inferring, querying, etc.) ontology data and ontology-based data (also called ontology individuals or ontology class instances) are available [8]. Usually, ontology-based data manipulated by these tools are stored in the main memory. Thus, for applications that manipulate a large amount of ontology-based data, it is difficult to ensure acceptable performance. Over the last five years, several approaches have been proposed for storing ontology-based data in database schemas to get

benefit of the functionalities offered by DBMSs (query performance, data storage, transaction management, etc.) [2, 10, 3, 15, 13, 9] (see [19] for an extensive comparison) We call this kind of databases Ontology-based Databases (OBDBs).

Two main OBDB structures for storing ontology and ontology-based data were proposed [3, 4, 15]: the *single table approach* and the *dual schemes approach*. In the single table approach, the description of classes properties and their instances are stored in a single table called *vertical table* [1]. The schema of this table has three columns: (*subject*, *predicate*, *object*), representing instance identifier, property of an instance and value of an instance, respectively. This approach is simple to implement and the structure may be used both for the ontology and for instance data. Therefore, tools (inference engine, APIs, etc.) developed for storing ontologies can also be used for processing instances data. To ensure a high performance of queries, each column shall be indexed, and the predicate column shall be clustered [1]. Therefore, this approach requires extra storage cost and it may be not very efficient to process queries having a large number of join operations [2].



TRIPLES			
SUBJECT	PREDICAT	OBJECT	
Person	rdf:type	rdfs:class	Ontology
Student	rdf:type	rdfs:class	
Worker	rdf:type	rdfs:class	
Student	rdfs:subClassOf	Person	
Worker	rdfs:subClassOf	Person	
name	rdf:type	rdfs:property	
name	rdf:range	xsd:String	
...	...	...	
Student#1	instanceOf	Student	Instances
Worker#1	instanceOf	Worker	
Student#1	name	Peter	
Worker#1	name	John	
...	...	...	

Fig. 1. vertical table approach.

To overcome the drawback of the first approach, a dual scheme approach has been proposed. It consists in storing separately ontologies and instance data in two different structures, called ontology and data, respectively [2, 4, 15]. The ontology structure depends upon the ontology model (e.g., RDF Schema, DAML+OIL, OWL). For instance, figure 2 shows an example of ontology structure for RDF schema.

In the dual scheme approach, instances and their properties values are also stored separately. Three different schemes have been proposed to record class belonging [19]. (1) a single table with two columns ID and Class, representing the identifier of an instance, and its ontology class. (2) one table per class with only one column storing all IDs of class instances [19] (see *NOISA* on figure 3).

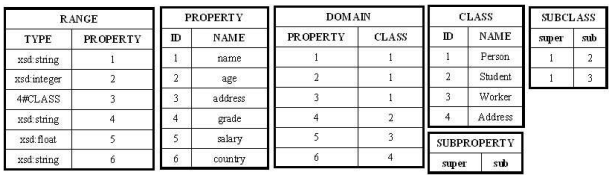


Fig. 2. Ontology Schema in the dual scheme approach.

(3) one table per class with table inheritance using SQL99 capabilities [19, 2] (see *ISA* on figure 3).

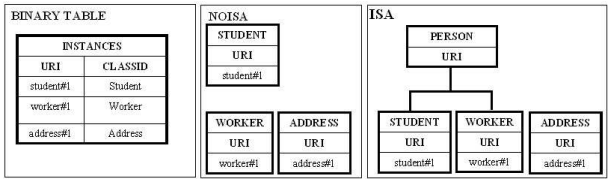


Fig. 3. Scheme instances representation approaches.

Property values are represented in a binary table or in vertical table of triples with or without strong typing.

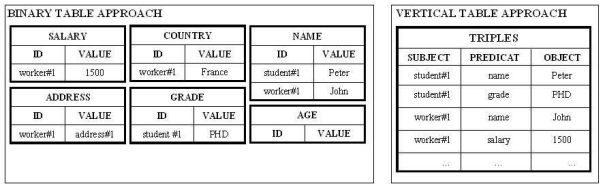


Fig. 4. Property values in the dual scheme approach

The *dual scheme approach*h has been proved to be more efficient than the single approach in various configurations [19]. Nevertheless, when the number of properties associated with each instance grows, browsing or querying instances becomes more and more difficult as it requires a large number of joins. The goal of this paper is (1) to propose a third approach for recording ontologies and ontology-based data and (2) to present performance results that show in which context this approach outperforms the classical approaches.

The paper is organized as follows: Section 2 presents the motivation of our ontology-based database architecture and PLIB ontology model. Section 3 presents the architecture of OntoDB model and our proposed representation,

called, table per class, for storing ontology-based data. Section 4 presents our experimental results. Section 5 concludes the paper and presents some perspectives.

2 Motivation of our Ontology-based Database Architecture

In this section, we outline the context of our work on ontology-based databases. In the 90s, to allow the exchange of electronic catalogues of industrial components, an ontology model for technical domain was developed [12] and then published as an international standard known as PLIB (ISO 13584-42: 98). Then, a model to exchange objects described in terms of such ontologies was developed [17] and also standardized (ISO 13584-25:2003). In the beginning of 2001, the PLIB model being finished, a new project called OntoDB was launched. It aimed to store, exchange, integrate and process industrial catalogues modelled as instance data associated with a formal ontology. PLIB-based ontologies were first targeted. Then, the decision to also to support other ontology models such that OWL or DAML+OIL has been taken.

We outline below both the PLIB ontology model and the model for PLIB-based instance data.

2.1 PLIB Ontology Model

The PLIB ontology model is technical domain-oriented as it supports three main capabilities broadly used in engineering: (1) a property value may depend upon its evaluation context (e.g., the length of an axis depends upon its temperature), (2) the property value may be associated with a measure unit (e.g., a temperature may be expressed in degree Celsius), and (3) an object may be characterized by one single class, and it may be represented by any number of discipline-specific ontology class, the point of view itself being represented by an ontology class [16].

PLIB ontologies are domain ontologies: they describe by means of properties all the consensual entities of the target domain. Each property is defined in the context of a class, that constitutes its domain, and it has a meaning only for this class and its possible subclass(es). To avoid the contextual character of a classification, in a PLIB ontology, a class is created only if it is necessary define domain of a property that would not be understood in the context of its superclass. Inversely, a property can be defined in the context of a class even if it does not apply to all its instances or subclasses. The single condition is that it is defined in an unambiguous way.

Thus, class hierarchies of PLIB ontology are extremely "flat". They do not define all the possible classes existing in a given domain, but they define only a canonical minimal vocabulary that consists only of primitive concepts. This vocabulary shall only make it possible to describe, in a single way by a class belonging and a set of properties value pairs, all instances which are subject of

a common understanding by domain experts. Any entity existing in a domain can thus be described, either directly in terms of the shared ontology or by adding additional classes that refine shared concepts and/or that add properties to the shared ontology. To do so, the PLIB ontology model offers an extension relationship, called *case of*. This relation offers to an user the possibility to define her own ontology by specializing the relevant subset of a shared ontology. From a semantic point of view, a class *case-of* of another class is subsumed by the later. From a syntactic point of view, the case of class can import from its subsuming class any subset of its applicable properties. This relationship makes it possible to define, from the same shared ontology, various user ontologies with quite different structures, while preserving the capability to integrate automatically ontology-based data (the case-of link may be recorded in each instance).

2.2 PLIB Instance Data Model

Contrary to individuals of description logic-based ontologies that may belong to any number of non connected ontology classes, the PLIB instance model is strongly typed. This means that (1) each instance belongs to exactly one minimal class (called its basis class which is the minimum for subsumption order of all the classes to which the instance belongs, (2) each property is defined in the context of a class that defines its domain of application, and is associated with a range and (3) only properties that are applicable in the context a class may be used for describing its instance. This assumption, rather similar to the OEM model in the TSIMMIS project [5] ensures that there exists an envelop modem that fits with any instance of a class: namely the set of all its applicable properties. But, unlike strongly typed conceptual models, an instance is not required to be associated with values for all the applicable properties of its basis class. This simple principle gives more schematic autonomy to the various data sources that may refer to the same ontology while preserving automatic integration capabilities. It also makes consensus more easier during ontology design.

3 OntoDB model architecture

We describe below the OBDD architecture we have proposed for storing ontologies and PLIB-instance data. Three objectives were assigned to our architecture model:

1. it shall support automatic integration and management of heterogeneous populations whose data, schemas and ontologies are loaded dynamically;
2. it shall support evolutions of the used ontologies (adding new classes, new properties, etc.) and their population schemas, and
3. it shall offer data access, at the ontology level, whatever is the type of the used DBMS (relational, object-relational or object oriented).
   Taking into account these objectives, our architecture is composed of four parts. Parts 1 and 2 are traditional parts available in all DBMSs, and parts 3 and 4 are specific to OntoDB (Figure 5).

– **Meta-base** ("*system catalog*"). It is a traditional part of databases. It is made of the whole set of system tables used by a DBMS to manage data and to ensure its processing. Therefore, all tables of all other parts of OntoDB are recorded in this meta-base. By means of naming convention, this part also represents the logical model of the content, and its link with the ontology, thus representing implicitly the conceptual model of data in database relations.
– **Content** ("*data*"). It is the data part in traditional databases. It allows to store instances of ontology's classes in DB relations.
– **Ontology**. It allows to represent ontologies in the database. When the ontology model is object-oriented and the target DBMS is relational, its logical schema is defined using an object/relational mapping. If shared ontologies have been specialized as local ontologies, both shared ontologies and specialization may be recorded in this part.
– **Meta-schema.** It records the ontology model into a reflexive meta model. For the ontology part, the meta schema part plays the same role as the one played by the meta-base in traditional DBs. Indeed, this part may allow: (1) generic access to the ontology part, (2) support of evolution of the used ontology model, and (3) storage of different ontology models (OWL [6], DAML-OIL, PLIB [12], etc.).

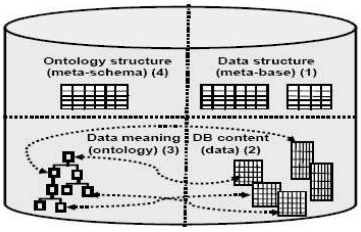


Fig. 5. The OntoDB Architecture

Therefore our OBDB model, called OntoDB represents explicitly: (1) ontologies, (2) data structures, (3) data, (4) the links between the data and their schema and (5) the link between the data and the ontology.

We just detail below the representation of ontologies and of instance data.

3.1 Representation of Ontologies

The ontologies supported by our architecture are all those that can be represented as models following Bernstein's terminology, i.e., a set of objects accessible through a root object using links between objects. This definition corresponds to most of the ontology models recently developed such as OWL [6], and in particular PLIB. Such ontologies are represented as sets of instances of an object

schema (often called meta-model) expressed in a particular modelling formalism (XML-schema for OIL et OWL, EXPRESS for PLIB). As a result, this representation provides an exchange format for these ontologies (an XML document for OWL, a physical file of EXPRESS instances for PLIB). In OntoDB, the ontology schema is generated automatically from EXPRESS model (for OWL for instance, the OWL model only needs to be expressed in EXPRESS).

3.2 Instances Representation

The main goal of ontologies is the representation of the semantic of objects of a given domain. This goal is reached by assigning objects to ontological classes and by describing them using ontological properties. According to the used ontology model, various constraints govern such descriptions. For instance, in RDF Schema, DAML+OIL and OWL, an object may belong to any number of classes and can be described by any set of properties. Therefore each domain object has its own structure. In the opposite, a database schema aims to describe "similar" objects by an identical logical structure in order to optimize queries using indexing techniques. Without any particular assumption, the only possible common structure consists in associating each object: (1) with any subset of set of classes; and (2) with any subset of the set of properties.

The drawbacks of such structure become evident when, like in engineering, each instance is characterized by a significant number of properties (e.g., 50), but this number is quite smaller than the total number of properties of the various classes of objects (e.g., 1000). Either it would require an important storage cost (by using a systematic representation of all properties for every instance). This approach has been evaluated by Agrawal et al. [1] which argued that it is not efficient. Or it would also require a significant response time due to the need of performing a large number join operations, if one of the approaches presented in the introduction are used.

STUDENT		
OID	**name**	**grade**
student#1	Peter	PHD
...	...	...

WORKER			
OID	**name**	**salary**	**address**
worker#1	John	1500	address#1
...	...	...	...

ADDRESS	
OID	**country**
address#1	France

Fig. 6. table per class representation approach.

Thanks to the strong typing assumptions presented in section 2.21, it is possible for us to define a relevant schema for any ontology class. This schema consists of all the class applicable properties that are used at least by one instance of the class. Of course, this schema might contain in some case a number of null values. But experience shows that in most context, and in particular both in industrial component catalogues and in Web portal meta data catalogues (see e.g., [2]) basically the same properties are used for the various instances of the same class. As a rule, the null values are much less than 50%. Thus our approach

supports efficient query processing. Note that this schema definition implies a major difference between object-oriented databases (OODB) and our OBDB. In an OODB, subsumption means inheritance of properties/attributes. All the properties defined in some class do exist in all its subclass(es). The only mechanism for property sharing between two subclasses of a class is to factorize this property at the level of the mother class. But then the property shall appear in all sibling classes. In OBDBs, inheritance is intentional: it concerns only the ontology level. Represented properties may be any subset of applicable properties. Thus two classes (C_1, C_2) may share a property P_1 when none of their sibling class, (e.g., (C_3, C_4, C_5) use this property. It only means that P_1 is applicable (from an ontological point of view) for all C_1 to C_5, but that only C_1 and C_2 provide a value of this property for (some of) their instances. This makes the OBDB model much more flexible, in particular for integrating heterogeneous databases.

To summarize our ontology-based data representation, we create a table for each class in the database. Its columns consists of a subset of applicable properties those that are used by some of its instances.

Figure 6 shows an example of our representation structure with ontology data of figure 1a. In the following section, we will call by *table per class* our approach of representation of ontology-based data.

4 Evaluating Instance Representation Schemes

In order to study the effectiveness of our representation of ontology-based instance data, we carried out a series of performance experiments to compare two representation schemes: *binary* and *table per class*. Our initial intent was to compare also with the *vertical* representation. Finally, we restricted to binary representation for four reasons:

1. Performance results of binary and vertical representation are very similar if the vertical table is clustered [1].
2. Most popular OBDB management systems (e.g., Sesame [4], RDFSuite [2] and DLDB [15]) use binary representation to store ontologies instances.
3. During our experiments, we got some problem to cluster *vertical* tables for the very large databases [1] we have created and we know that clustering operation is necessary to guarantee efficiency of the vertical representation [1].
4. The clustering operation is time consuming. A test was done on a small database [2] and we got about 3 min and 30 seconds. This time is very significant because a clustering operation has to be performed each time modification queries (insertion, update, deletion) are executed.

As an experimental platform, we use the ORDBMS PostgreSQL-7.4 (emulated on cygwin) installed on a Pentium 3.7 GHz CPU, 6 GO of RAM, 200 GO of Hard Disk. In all our experiments, a cache memory of 50 MO has been used.

[1] Our databases are created on PostgreSQL

[2] DB_10P_1K : database which has 10 valued properties and 1K instances per class

4.1 Databases

To perform our experiments, we use an ontology that is representative of our application domain. It describes the various kinds of electronic components together with their characteristic properties. Published as an International Standard in 1998, IEC 61360 [11] is composed of 190 classes: 134 leaf classes and 56 no leaf classes. These classes have a total of 1026 properties. The average deep of the IEC ontology hierarchy is 5. To facilitate the computation of the sizes of the test databases, all ranges of properties were changed to have a string (255) as their range. A generator of population of each class has been developed. Various contents of database were generated by varying the number of instances and the number of valued properties used for each class. We denote by *TP* and *TC*, *the binary table per property*) and *table per class* approach, respectively. Let DB_aP_iK be a database with a properties and iK instances per class. For example, *BD_50P_2K* is a database with 2K instances and 50 valued properties for each class in the database.

To conduct our experiments, we create six databases. The description of these databases is shown in Figure 7. These databases have been classified into two series: databases in the first serie (Serie1) have the same number of instances per class and a different number of properties: *BD_10P_1K*, *BD_25P_1K* and *BD_50P_1K*. Thus, they allow to study the effect of database size. Databases in the second category (Serie2) have the same size (*InstancesNumber* × *PropertiesNumber*), but different number of instances and of properties per class: *BD_10P_10K*, *BD_25P_4K* and *BD_50P_2K*. This classification allows us to study the effect on query performance of the number of properties and of instances per class.

	Serie 1			Serie 2		
	DB 10P 1K	DB 25P 1K	DB 50P 1K	DB 10P 10K	DB 25P 4K	DB 50P 2K
Number of valued properties / class	10	25	50	10	25	50
Number of instances / class	1K	1K	1K	10K	4K	2K
Number of initialised classes	134	134	134	134	134	134
Total number of instances in DB	134K	268K	134K	1340K	536K	268K
Data size in DB	0,341 GO	0,84GO	1,68GO	3,41GO	3,41GO	3,41GO

Fig. 7. Databases created.

4.2 Query Taxonomy

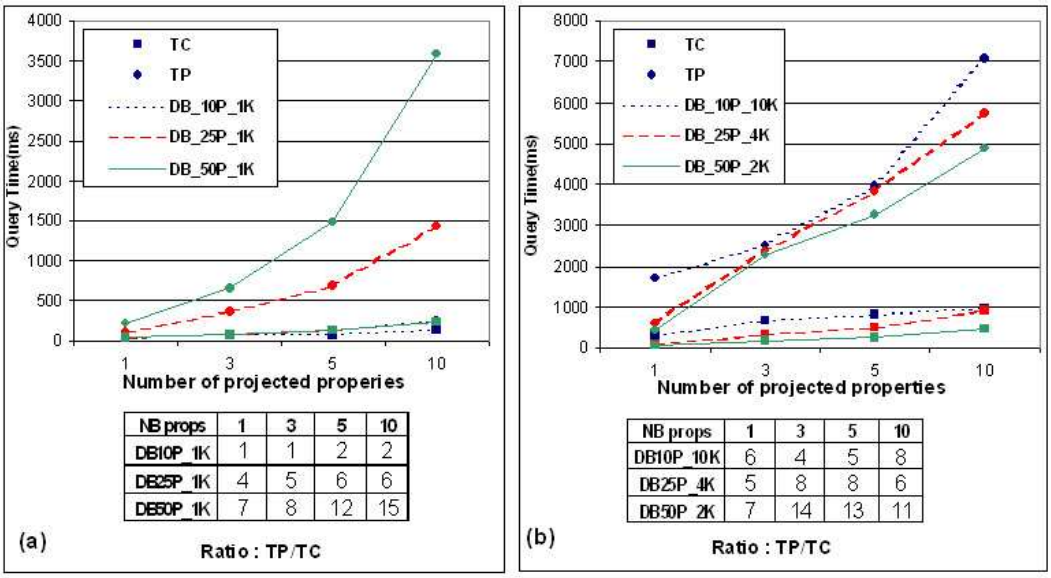
We consider three classes of queries: *targeted class queries*, *no targeted class queries* and *update queries*. In a targeted query, the user knows the classes that she wants to query. For example, "find all students that are more than 25 years old". In a non targeted class of queries, the user does not know the classes that she is looking for. For instance, "list all instances in the database that are more than 25 year old".

In this study, we consider PSJ queries (projection, selection and join). These queries are executed on leaf and non-leaf classes. As in [1], we will not consider queries using protection, selection and join operations, simultaneously.

To get reproducible experimental results, we carry out all benchmark queries in the following way. Every query is performed once to warm up the database buffer and then it is performed at least three times in order to get a mean running time.

4.3 Performance Results for Targeted Class Queries

We conduct three series of experiments: (1) projection queries, (2) selection queries, and (3) join queries. Figures below show a set of curves that give, for each particular database, the query execution time as a function of a particular criteria. The ratio table when present, precises how many times our TC representation is faster than the classical TP representation.



NB props	1	3	5	10
DB10P_1K	1	1	2	2
DB25P_1K	4	5	6	6
DB50P_1K	7	8	12	15

NB props	1	3	5	10
DB10P_10K	6	4	5	8
DB25P_4K	5	8	8	6
DB50P_2K	7	14	13	11

Fig. 8. Projection (a) for various size of databases, (b) for various structure of databases.

Projection within a leaf class In order to evaluate the impact of the number of projected properties, we performed four queries with 1, 3, 5 and 10 projected properties, respectively. Figure 8 summarizes the execution time. The response time for TC is relatively constant when we vary the number of projected properties, while for *TP* approach, it grows faster and faster when the database sizes

grow. The left part shows the behavior for several databases with same size, but with different structures is quite similar. Thus, we will not represent this kind of comparison for the other operators unless it is really significant.

In the TP approach, the variation of the number of projected properties means the augmentation of join operations. Therefore the cost increases dramatically when all the relevant binary tables cannot be simultaneously fetched in the main memory. This variation does not have a real effect on TC approach. This is because the DBMS fetches all instances of the target class (for all the classes of the benchmark) with their properties in main memory before truncating properties required by the query. Therefore, the IO cost is still the same for each query with any number of its projected properties.

For the biggest database of our benchmark (1,7 to 3,4 GO), projection on 10 properties is about 10 to 15 times faster with TC than TP.

Selection Within a leaf class, Figures 9a and 9b show performance of selection queries on one of the biggest database, namely, *DB_50P_2K*. In Figures 9a, we vary the number of properties in the selection predicate from 1 to 4. The augmentation of the number of properties increases query response time in *TP* approach. Once again, the worst performance is justified by the number of join operations and the sizes of property tables that may cause an important IO overhead. Changing the selectivity factor of the predicate that contains only one attribute does not change significantly the behavior of both representations. Globally, TC representation outperforms TP by a factor between 50 and 100.

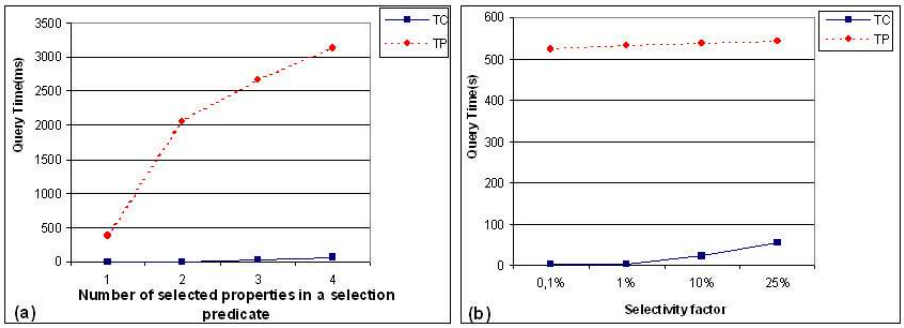


Fig. 9. Selection (a) with various number of properties, (b) with various selectivity factor.

Join Operations within a leaf class Figure 10 shows the performance of join queries performed on databases of Series1. The queries return 1 property value per class. The join selectivity is fixed to 0.25%. *TC* approach has better performance than *TP*. Variation of databases size increases the ratio between

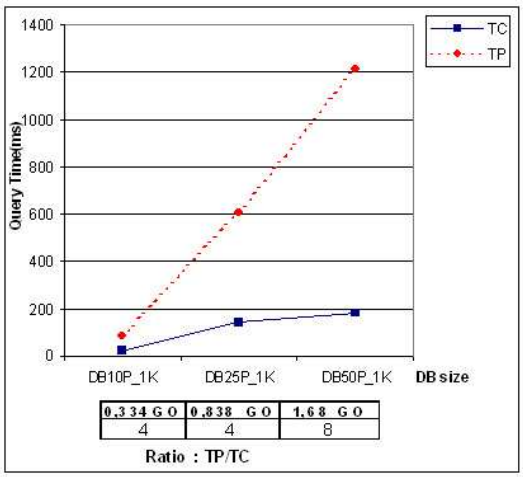


Fig. 10. Join within a leaf class.

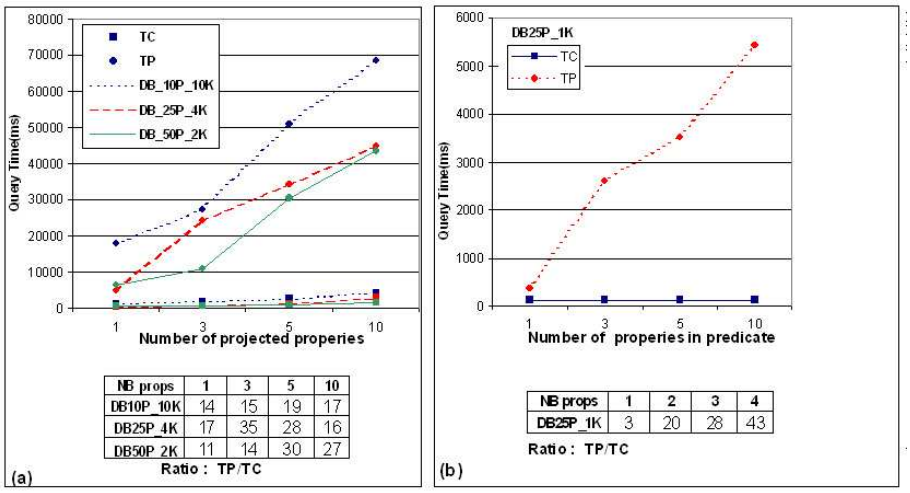
TP and TC. The reason of worst performance of TP is justified by the size of the binary property tables and the fact that a preliminary join is needed between the class table and the property binary tables. In our domain of study, TC outperforms TP between 4 and 8.

Note that obviously, the number of projected properties has an important impact on query performance (for instance for five properties in the SELECT statement, the ratio is between 14 and 18).

Projection and selection within non-leaves classes Figure 11 shows performance results for selection and projection queries within a non-leaf class which has seven (7) subclasses. When we compare these results with queries in leaves classes we notice that the ratio performances are similar. Query response time in a non-leaf class is the *sum* of queries response time performed in each subclass of the non-leaf class. This explain why the shape of the curves of performances are identical in queries on leaves and non-leaves classes (see figures 8b and 11a for the projection queries and figures 9b and 11b for selection queries). In these experiments, the ratio TP/TC is between 11 and 35.

4.4 No Targeted Class queries

When the class to be queried is unknown for the user, the advantages of the *table per class* approach may disappear. Such queries may be formulated as follows: "find all instances in the database that have value val_1 for a property P_1 AND/OR val_2 for a property P_2", etc. Execution of this kind of queries in *TC* approach is performed in two steps. In the first step, one finds all classes in the databases which uses properties (P_1, P_2). In the second step, selection



NB props	1	3	5	10
DB10P_10K	14	15	19	17
DB25P_4K	17	35	28	16
DB50P_2K	11	14	30	27

Ratio : TP/TC

NB props	1	2	3	4
DB25P_1K	3	20	28	43

Ratio : TP/TC

Fig. 11. (a) Projection within a non-leaf class. (b) Selection within a non-leaf class

queries are performed in all the classes found in the first step. In *TP* approach, execution of *non-targeted queries* are performed directly by a join of the tables of the properties present in the query predicates.

We note that this kind of query is hardly used in our application domain: "we never request an object with the weight equals 1 kilogram". Moreover, if one does not know the class of an object, we need, at least, several properties for characterizing this object. Therefore, such queries request projection on several properties.

We run these queries against databases of growing sizes (Series 1). We vary the number of projected properties to 1, 3, 5 and 10 to represent realistic queries. We remark that *TP* approach is more efficient than *TC* approach as long as queries return less than 5 properties. Beyond this number of properties, *TC* approach becomes more efficient.
The worst performance of *TC* approach when a small number of properties is requested is due to access time to the *ontology part* to obtain all classes which use the properties referenced in the queries. Query on *ontology part* needs to traverse all classes and to test for each class if the searched properties belong to it. Notice that the time to get all classes in the first step during execution of *non-targeted queries* is relatively constant when we vary the number of properties in the queries, contrary to *TP* approach where every new property cause one more join in the queries. So, when the number of requested properties increases, to compute classes in the first step in *TC* approach becomes smaller than the time of joins in the *TP* approach. This explain why in figure 12d (projection on 10 properties) *TC* approach has better performance than *TP* approach.

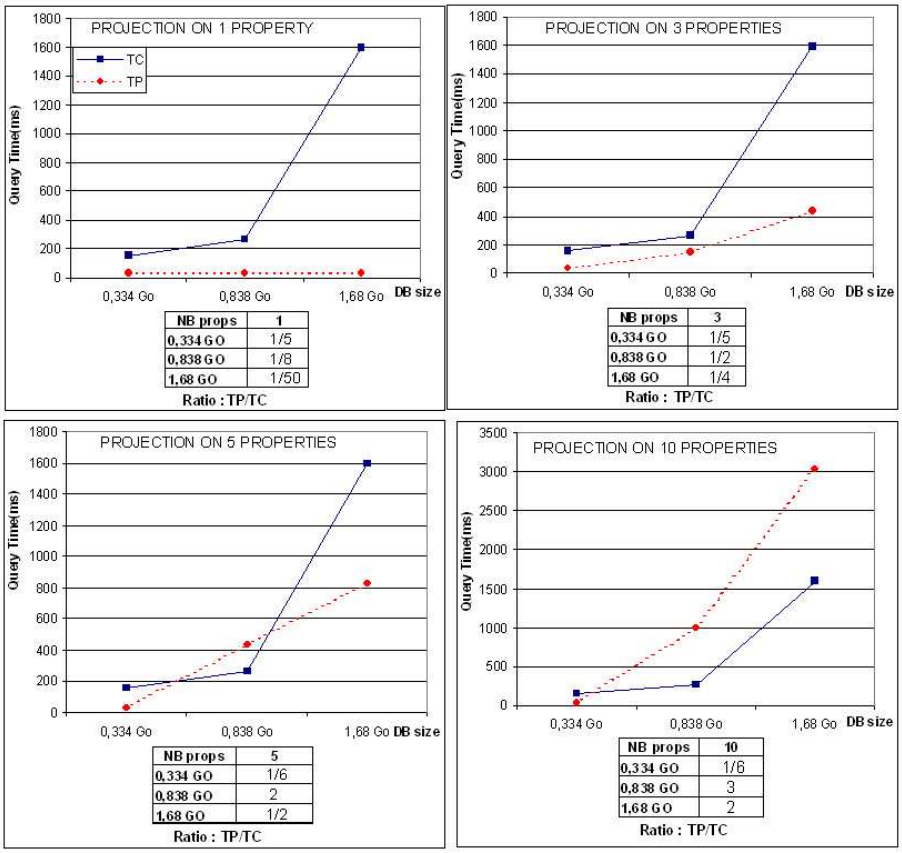


Fig. 12. No Targeted Class Queries

4.5 Update Queries

Figures 13a and 13b show performance results of insertion and update queries. We run queries on databases of growing size. Both figures show that *TC* is more efficient than *TP*. In case of insertion, the worst performance of *TP* results from the fact that all tables concerned by insertion of valued properties need to loaded in the memory, while *TC* approach, needs only one loading of a single table. For update queries (concerning only one property value), the worst performance of *TP* approach is due to the size of the property table that needs to be loaded. The cost ratio between TP and TC ranges from 2 to 56 for insertion and is about 2 from each update for a singe property.

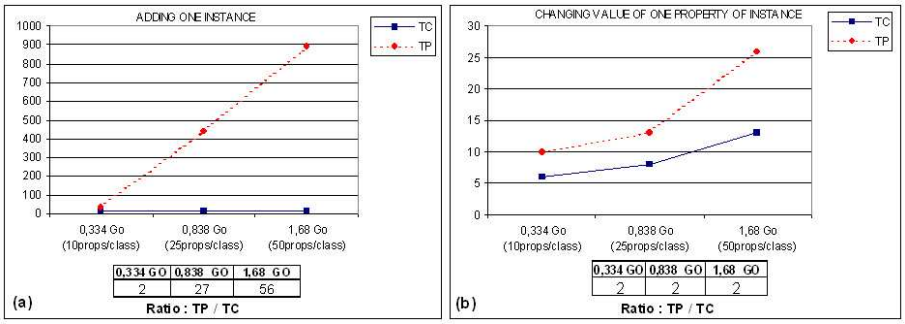


Fig. 13. Insert and Update Queries

5 Conclusion

A number of ontology-based database structures have been proposed during the last five years. Most of them are targeted to support real scale Semantic Web applications. Several benchmarks were proposed to compare their performance using Semantic Web oriented data. These benchmarks focus mainly on the class structure and taxonomy queries (i.e., retrieving the proper or transitive instances of a particular class or property). There exists other applications of ontology-based database focusing mainly on property-value pairs. It is the case of engineering databases, electronic catalogues of industrial components and a number of B2B applications. In such context, instance data consists of a class belonging and a number of property-value pairs. Most queries associate with instances a number of properties.

In this paper, we firstly presented the ontology model we developed for the engineering domain, secondly an OBDB architecture, called, OntoDB and finally, the data structure we propose for storing instance data. This structure, called *table per class*, associates to each ontology class a table that contain as columns those applicable properties of the class that are associated with a value for at

least one instance of the class. Our proposed benchmark for comparing this approach with the best approach known in the Semantic Web context, namely the binary table approach, used a real standardized ontology. Thus, it reflects the need of our application domain. Our benchmark is based on three kinds of queries: (1) targeted class queries, where the user is supposed to know the root class of the subsumption tree to be queried, (1) non targeted class queries, where the user does not know what kind of ontology concepts she is looking for, and (3) insertion and update queries.

For all these queries, the table per class approach outperforms the classical binary table approach with ratio often bigger than 10. The only case where the binary approach is better than our approach is for the no targeted class queries, when the user also requests a very small number of property values. We note that this kind of queries nearly never happens in our application domain. Engineers always knows what they are looking for before searching for property values.

Our OntoDB prototype is already supporting more than millions of instance with dozen of properties, but it mainly uses PLIB ontologies. We are currently working to make its ontology model more flexible, to integrate other kind of ontologies. We are also improving the ontology implementation to speed up the ontology browsing process. Finally, we are developing an SQL oriented OBDB query language that integrates a number of RQL and of SQL99 capabilities.

References

1. R. Agrawal, A. Somani, and Y. Xu. Storage and querying of e-commerce data. In *VLDB*, pages 149–158, 2001.
2. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing voluminous RDF description bases. In *SemWeb*, 2001.
3. B.McBride. Jena: Implementing the rdf model and syntax specification. *Proceedings of the 2nd International Workshop on the Semantic Web*, 2001.
4. J. Broekstra, A. Kampman, and F.V. Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In I. Horrocks and J. Hendler, editors, *Proceedings of the First Internation Semantic Web Conference*, pages 54–68, July 2002.
5. S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, pages 7–18, Marsh 1994.
6. M. Dean and Schreiber. Wl web ontology language reference. *W3C Recommendation (2004)*, February 2004.
7. N. X. Dung, L. Bellatreche, and G. Pierra. A versioning management model for ontology-based data warehouses. *To Appear in Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2006)*, September 2006.
8. JH. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, M. Crubézy, H. Eriksson, N. Noy, and Samson W. Tu. The evolution of protg: an environment for knowledge-based systems development. *Int. J. Hum.-Comput. Stud.*, 58(1):89–123, 2003.

9. S. Harris and N. Gibbins. Store: Efficient bulk rdf storage, 2003.
10. ICS-FORTH. The ics-forth rdfsuite. *http://139.91.183.30:9090/RDF*, page web site, 2001.
11. IEC. Iec 61360 - component data dictionary. *International Electrotechnical Commission. Available at http://dom2.iec.ch/iec61360?OpenFrameset*, 2001.
12. ISO13584-42. Industrial automation systems and integration parts library part 42 : Description methodology : Methodology for structuring parts families. Technical report, International Standards Organization, Genève, 1998.
13. L.Ma, Z. Su, Y. Pan, L. Zhang, and T. Liu. Rstar: an rdf storage and query system for enterprise resource management. *thirteenth ACM international conference on Information and knowledge management*, 2004:484 – 491.
14. B. Omelayenko and D. Fensel. A two-layered integration approach for product information in b2b e-commerce. *Proceedings of the Second International Conference on Electronic Commerce and Web Technologies*, pages 226–239, September 2001.
15. Z. Pan and J. Heflin. Dldb: Extending relational databases to support semantic web queries. *Workshop on Practical and Scalable Semantic Systems ISWC2003*, 2003.
16. G. Pierra. A multiple perspective object oriented model for engineering design. *in New Advances in Computer Aided Design & Computer Graphics*, pages 368–373, 1993.
17. G. Pierra. Modeling classes of preexisting components in a cim perspective: The iso 13584/env 400014 approach. *Revue Internationale de CFAO et d'Infographie*, 9:435–454, 1994.
18. H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information - a survey of existing approaches. *Proceedings of the International Workshop on Ontologies and Information Sharing*, pages 108–117, August 2001.
19. V. Christophides Y. Theoharis and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *Fourth International Semantic Web Conference (ISWC'05)*, November 2005.