Proc. of Database Systems for Advanced Applications (DASFAA'2007),
Bangkok, Thailand, April 9-12, 2007.

OntoDB: An Ontology-Based Database for Data Intensive Applications

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 database data and the ontologies describing their meanings. We call these databases, ontology-based databases (OBDBs), and Ontology-based data denotes those data that represent ontology individuals (i.e., instance of ontology classes). To ensure a high performance of queries on top of these OBDBs, efficient representation of 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. In all these schemes, each instance is split into a number of tuples. 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 a same row. Columns of this table represents those properties of the ontology class that are associated with a value for at least one instance of this class. We present also the context of our project, the architecture we have developed for ontology-based databases, and a comparison of the effectiveness of our representation 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, information integration, e-commerce, data warehousing, 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 (e.g., Protégé 2000, etc.). 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 both ontologies and ontology-based data in database schemas to get benefit of the functionalities offered by DBMSs (query performance, data storage, transaction management, etc.) [2–4, 8, 11, 12] (see [16] for an

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

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: the *single table approach* and the *dual schemes approach*. In the single table approach [1, 3, 8, 11], 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], or materialized views need to be created [12]. In both cases, this approach requires extra storage cost and it may be not very efficient to process queries having a large number of join operations [2]. Moreover, the drawback of such optimizations is the update overhead.

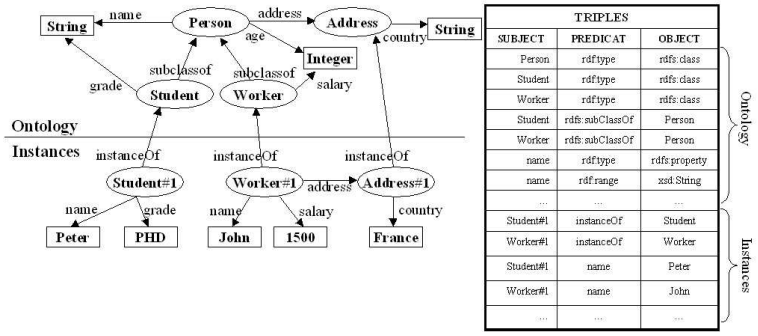


Fig. 1. vertical table approach.

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, 12]. The ontology structure depends upon the ontology model (e.g., RDF Schema, DAML+OIL, OWL). Figure 2a 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 [16]. In the two first approaches, each class is mapped onto a table. The third one, called, hybrid [2] maps all classes on the same binary table. These scheme are as follow: (1) one table per class with only one column storing all IDs of class instances [2, 15] (see *NOISA* on figure 2b). (2) one table per class with table inheritance using SQL99 capabilities [2, 4] (see *ISA* on figure 2b). (3) a single table with two columns ID and Class, representing the identifier

of an instance, and its ontology class [2]. The ID column may be either the URI or integer identifiers mapped on URI in a particular binary table (called "instances").

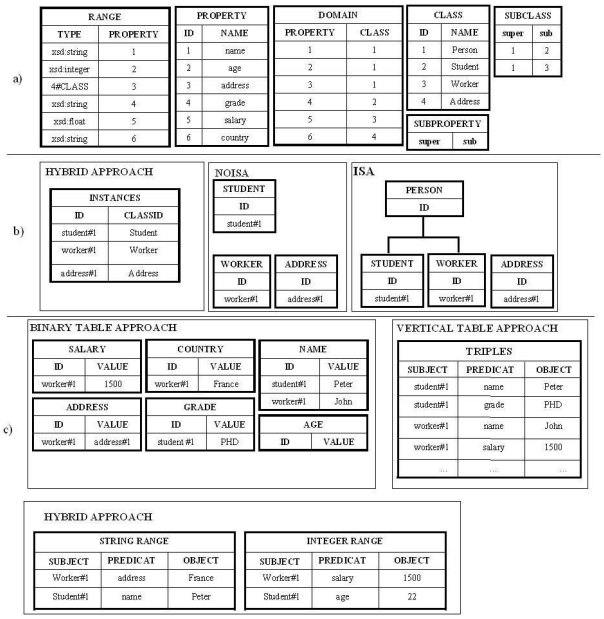


Fig. 2. (a) Ontology Schema in the dual scheme approach. (b) Instance scheme alternative representations. (c) Property value scheme alternative representations

Three representation of property values are possible: (1) binary tables, (2) vertical table of triples and (3) hybrid approach consisting of a set of triple tables (one per a range data type) (see Figure 2c). It is worth noticing that both the vertical table approach, and the dual scheme approach with hybrid representation of instances and properties involve a small number of large tables when other dual scheme a approaches involve a large number of smaller tables.

A number of benchmarks were already performed to compare these various approaches [1, 2, 11, 12, 16]. The main findings may be summarized as follows :

- the vertical table approach may only provide similar query results with the dual schema when the vertical table is clustered [1], and/or when materialized views represent the dual scheme content [16]. Even in this case the vertical table approach provides worst results for taxonomic queries, i.e., those queries requiring subsumption inference [16].
- The dual scheme approach with hybrid representation of instance belonging and property values also require clustering operations of these tables [16], else this approach is outperformed by dual scheme with unary instance tables and binary property tables.

– The clustering operation is time consuming. A test was done on a small database [1]. We got about 3 mins and 30 seconds, which is very significant. This is because a clustering operation has to be performed each time for each update query. Thus the cost of updating the database schema when unary and binary representation are used is to be compared with the cost of clustering data in vertical table and hybrid approaches.
– Most popular OBDB management systems (e.g., Sesame [4], RDFSuite [2] and DLDB [12]) use binary representation to store ontologies instances.

Thus, from the previous experiments, we may conclude that the dual scheme approach with unary instance and binary property representations appear as a suitable approach for those databases that are both rather large and often updated. 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, called OntoDB and the PLIB ontology model. Section 3 presents the architecture of OntoDB 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 [10] 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 [14] 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 modeled as ontology-based 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.

[1] DB_10P_1K : database which has 10 valued properties and 1K instances per class (see section 4.1)

2.1 PLIB Ontology Model

The PLIB ontology model is technical domain-oriented as it supports four 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), thus a property may be a function, (2) the property value may be associated with a measure unit (e.g., a temperature may be expressed in degree Celsius), (3) an object must be characterized by one single " characterization class" (associated with properties), but it may also be associated with any number of discipline-specific ontology class, the point of view itself being represented by an ontology class [13], and (4) an object may be classified in any number of "classification class" (not associated with properties). It is worth noticing that this particular taxonomy of meta classes allows PLIB ontology to represent a number of Semantic Web applications such as Web Portal catalogs [2].

PLIB ontologies are domain ontologies: they describe by means of classes and 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 to define the domain of a property that would not be understood in the context of its super-class. 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.

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 characterization class (called its basis class which is the minimum for subsumption order of all the characterization classes to which the instance belongs), (2) each property is defined in the context of a characterization class that defines its domain of application, and is associated with a range and (3) only properties that are applicable in the context a characterization 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. The objectives of our architecture model were: (1) to support automatic integration and management of heterogeneous populations whose data, schemas and ontologies are loaded dynamically, (2) to support evolutions of the used ontologies and of ontology scheme, and (3) to offer data access, at the ontology level, whatever is the type of the used DBMS. Our architecture is composed of four parts. Parts 1 and 2 are traditional parts available in all DBMSs, namely the *data* part that contains instance data and *meta-base* part that contains the system catalog. Parts 3 (*ontology*) and 4 (*meta-schema*) specific to our OntoDB (Figure 3).

The *ontology* part allows to represent ontologies in the database. The ontologies supported by our architecture are all those that can be represented and exchanged as models following Bernstein's terminology, i.e., a set of objects accessible from 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. When the target DBMS is relational, the ontology part schema is defined using an object/relational mapping.

The *meta-schema* part 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, DAML+OIL, PLIB, etc.).

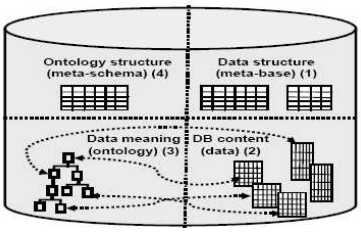


Fig. 3. The OntoDB Architecture

By means of naming convention, the meta-base part also represents the logical model of the content, and its link with the ontology, thus representing im-

plicitly the conceptual model of data in database relations. Therefore our OBDB model, called OntoDB represents explicitly: (1) ontologies, (2) data scheme, (3) data, (4) the links between the data and their schema and (5) the link between the data and the ontology.

Let us discuss the representation of ontology-based data. Ontologies allow representation of the semantic of objects of a given domain. This is done 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. All the OBDB scheme discussed in section 1 support this capability.

The drawbacks of such structure become evident when, like in engineering, each instance is characterized by a significant number of properties (e.g., 50), and when this number is quite smaller than the total number of properties of the various class instances (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 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. 4. table per class representation approach.

Thanks to the strong typing assumptions presented in section 2.2, 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 4 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. Note that our OntoDB model records explicitly the structure of each class table and that taxonomies queries (i.e., queries that require subsumption inference) are first evaluated intentionally (to know which classes use each particular property) before querying the data part.

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 scheme: unary instance and binary property representations (*binary* for short) vs *table per class*. Our initial intent was to compare also with the *vertical* representation. Finally, we restricted to binary representation for two reasons: (1) the existing benchmarks all that confirm that binary approach equals or outperforms other classical schemes (see section 1), and (2) some experiments we did on the vertical table approach proven that, in our context also, it was less efficient than the binary approach.

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.

4.1 Databases

To perform our experiments, we use an ontology that is both real and 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 [9] 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 50 valued properties and 2K instances for each class in the database.

	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	134K	134K	1340K	536K	268K
Data size in DB	0,341 GO	0,84GO	1,68GO	3,41GO	3,41GO	3,41GO

Fig. 5. Databases created.

To conduct our experiments, we create six databases. The description of these databases is shown in Figure 5. 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* $\times$ *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. Note the serie 2 databases containt 13.5 millions of RDF triples.

4.2 Query Taxonomy

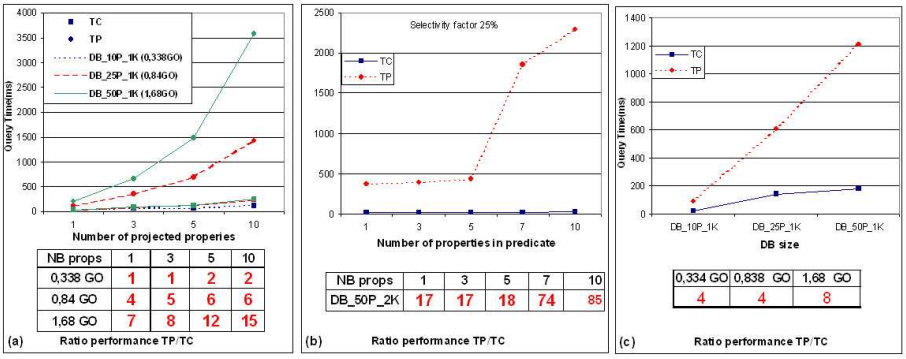
We consider three classes of queries: *targeted class queries*, *no targeted class queries* and *update queries*. In a targeted query, the user knows the classes that she wants to query. For example, "find all students that are more than 25 years old". In a non targeted class of queries, the user does 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, as their result may be deduced from PSJ results.

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. Figure 6 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 the classical TP representation is slower than our TC representation. Due to space limitations, we are not showing the complete experimented results that are described in [7]. In particular, the query behaviour for several databases with same size, but with different structures is quite similar. Thus, we will not represent this kind of comparison.



NB props	1	3	5	10
0,338 GO	1	1	2	2
0,84 GO	4	5	6	6
1,68 GO	7	8	12	15

NB props	1	3	5	7	10
DB_50P_2K	17	17	18	74	85

0,334 GO	0,838 GO	1,68 GO
4	4	8

Fig. 6. (a) Projection for various number of properties and various size of databases, (b) Selection with various number of properties (DB_50P_2K)(c) Join within a leaf class for various size of databases.

Projection within a leaf class. We performed four queries with 1, 3, 5 and 10 projected properties, respectively. Figure 6a summarizes the execution time. The response time for TC is relatively constant when the number of projected properties increases, while for *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. 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. Figure 6b shows performance of selection queries on one of the biggest database, namely, *DB_50P_2K*. We varied the number of properties in the selection predicate from 1 to 10. 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 17 and 85.*

Join Operations within a leaf class. Figure 6c 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 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.*

Projection and selection within non-leave classes. We have also evaluated projection and selection within a non-leaf class with seven subclasses. Query response time in a non-leaf class is the *sum* of queries response time performed in each subclass of the non-leaf class. So, the shape of the curves of performances are identical in queries on leaves and non-leaves classes. 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 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 ran these queries against databases of growing sizes (Series 1). We varied the number of projected properties to 1, 3, 5 and 10 to represent realistic queries. *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*. It 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 this step 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.

4.5 Update Queries

Figures 7a and 7b show performance results of insertion and update queries. We run queries on databases of growing sizes. 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.*

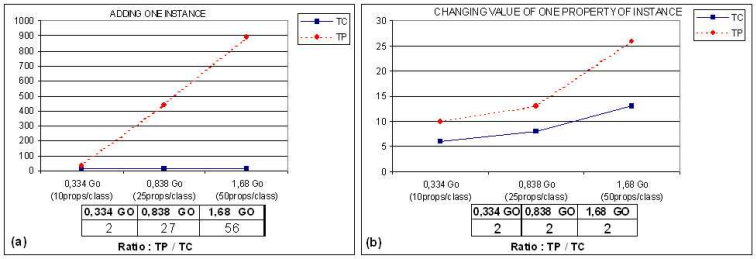


Fig. 7. Insert and Update Queries

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 our 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 with real size databases that contain up to 15 millions of RDF triples. 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 queries (1) and (3), 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 is for the no targeted class queries, when the user only 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 instances 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 most of RQL and of SQL99 capabilities.

References

1. R. Agrawal, A. Somani, and Y. Xu. Storage and querying of e-commerce data. In *Proc. VLDB'01*, pages 149–158, 2001.
2. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. On storing voluminous rdf descriptions: The case of web portal catalogs. In *Proc. ofWebDB'01 (co-located with ACM SIGMOD'01)*, 2001.
3. B.McBride. Jena: Implementing the rdf model and syntax specification. *In Proc. of the 2nd Intern. 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 *Proc. of the First Inter. Semantic Web Conf.*, pages 54–68, 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. H. Dehainsala, G. Pierra, and L. Bellatreche. Managing instance data in ontology-based databases. Technical report, LISI-ENSMA,http://www.lisi.ensma.fr/ftp/pub/documents/reports/2006/2006-LISI-003-DEHAINSALA.pdf, 2006.

8. S. Harris and N. Gibbins. 3store: Efficient bulk rdf storage. In *Proc. of the 1st Intern. Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, 2003.
9. IEC. Iec 61360 - component data dictionary. *International Electrotechnical Commission. Available at http://dom2.iec.ch/iec61360?OpenFrameset*, 2001.
10. 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.
11. 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.
12. Z. Pan and J. Heflin. Dldb: Extending relational databases to support semantic web queries. *ISWC'2003*, 2003.
13. G. Pierra. A multiple perspective object oriented model for engineering design. *in New Advances in Computer Aided Design & Comp. Graphics*, pages 368–373, 1993.
14. G. Pierra. Context-explication in conceptual ontologies: Plib ontologies and their use for industrial data. *to appear in Journal of Advanced Manufacturing Systems, World Scientific Publishing Company*, available at http://www.lisi.ensma.fr/ftp/pub/documents/papers/2006/2006-JAMS-Pierra.pdf 2006.
15. K. Stoffel, M.G. Taylor, and J.A. Hendler. Efficient management of very large ontologies. In *Proc. of American Association for Artificial Intelligence Conference (AAAI'97)*, 1997.
16. V. Christophides Y. Theoharis and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *Fourth International Semantic Web Conference (ISWC'05)*, November 2005.