Numeric reasoning in the Semantic Web

Chimène Fankam, Stéphane Jean, and Guy Pierra

LISI-ENSMA and University of Poitiers
BP 40109, 86961 Futuroscope Cedex, France
{fankamc,jean,pierra}@ensma.fr

Abstract. The Semantic Web is an effort by the W3C to enable integration and sharing of information across different applications and organizations using annotations by means of ontology instances. With the growth of such data, two important problems need to be addressed: (1) a scalability issue and (2) a performance issue for reasoning. Main memory reasoners are efficient for reasoning, but they hardly support real size data. This paper discusses the use of databases, more precisely of ontology-based databases (OBDBs), for managing Semantic Web annotation data. Such systems are able to manage real-size data. But the main weakness of databases are their poor deductive reasoning capabilities. Thus, we propose an approach that consists in enriching annotation instances with new numeric-valued or string-valued properties allowing to replace deductive reasoning by numeric queries. We define formally some cases where this approach may be implemented and we propose extension of ontology languages allowing to represent explicitly their structures. Ontology being recorded in OBDB, these extensions allow the OBDB system to perform dynamically the instance enrichment and to rewrite queries as numeric queries. This approach is in particular used for geometric reasoning, and we present its implementation within OntoDB, an OBDB developed in our laboratory.

1 Introduction

The Semantic Web is an effort by the W3C to enable integration of data sources across the Web. In order to capture information semantics in a machine processable way, Web resources are annotated with terms described as ontology individuals. Such ontology individuals are called *ontology-based data*. As Semantic Web technologies become mature and standardized, they are applied to real-world applications. As a consequence, an increasing amount of ontology-based data is becoming available on the Web. Managing such data raises two major issues :

- *a scalability issue*. A lot of applications need to manage an amount of ontology-based data that don't fit in main memory;
- *a reasoning issue*. An ontology is a conceptualization based on a formal theory that allows to reason over the ontology-defined concepts and individuals. Reasoning operations need to be performed in an acceptable response time.

K. Belhajjame, M. d'Aquin, P. Haase and P. Missier (Eds.): SeMMA 2008
CEUR Workshop Proceedings, ISSN 1613-0073, online at CEUR-WS.org/Vol-346/

Solving these two problems is an important factor in realizing the Semantic Web vision. The difficulty is to provide a solution that solves them together. Indeed, during these last years, several works have addressed the scalability problem using databases. If some works have focused on using databases to store instance data [1, 2], others have proposed new database architectures to store both ontology descriptions and instance data. We call these database architectures *Ontology-Based Databases* (OBDBs) [3–7]. Evaluations of OBDBs performance have shown that some architectures scale quite well and do support real size Semantic Web data management[8, 6, 7]. However, if scalability constitues a major strength of databases, this is not the case of their deductive capabilities. Thus, several propositions have been made to combine reasoners with databases [9–11, 1]. To speed-up the query response time of such architecture, reasoning is often done offline [6]. Then, database are used to materialize all the deduced facts which lead to a strong storage overhead.

In this paper, we propose an alternative approach. The idea is to use capability of database to process efficiently numeric queries and string-oriented queries to perform some reasoning operations at runtime. Thus our approach consists (1) in using the ontology representation available in an OBDB to interpret in a semantic way both the data manipulation language and the data query language of the database and (2) in enriching the annotations instances with new property values in order to replace deductive reasoning by numeric (or string-oriented) query processing. For example, when an object property π is defined, using OWL2 constructs, as asymmetric, transitive, and inverse functional, thus defining a tree-order, this tree-order ($\prec$) may be represented by means of numeric intervals [12]. Thus, (1) when an instance that supports the π property is inserted in the database, two additional data properties (`lo_bound`, `hi_bound`) are computed by the system. These data properties reflect the tree-order, i.e,

$$\mathtt{x} \prec \mathtt{y} \Leftrightarrow \mathtt{lo\_bound}(\mathtt{y}) < \mathtt{lo\_bound}(\mathtt{x}) < \mathtt{hi\_bound}(\mathtt{x}) < \mathtt{hi\_bound}(\mathtt{y}).$$

Then, (2) when annotations instances smaller than a given instance are requested, the query interprets access to the ontology and re-writes the recursive query over π as a numeric query over `lo_bound` and `hi_bound`. This kind of index is not really new. Indeed, a number of approach [12, 13], known as labeling, have been proposed to compute transitive closures of relationships and to index them using numeric or string-oriented labels. However, these approaches are often hard encoded in the data management system for pre-defined relationships such as polymorphism through class subsumption. We propose extensions of ontology languages allowing to discover when this kind of approach may be followed and to implement it dynamically when a new ontology is loaded. The proposed framework integrates various existing labeling schemes to address most recursive containment relationships, including subclass reasoning and taxonomic queries widespread in resource annotations, and we show that the same framework may be efficiently used for DAG structures encountered in spatial and temporal application.

The remainder of this paper is organized as follows. In the next section we present an overview of OBDBs. We propose a taxonomy of existing OBDBs

and present their scalability and deductive capabilities. In section 3 we present a framework to transform deductive reasoning into numeric reasoning for property whose range are partially ordered sets. In section 4 we discuss how this framework can be implemented within OBDB and in section 5 we describe our current implementation on a real-world application. Finally, we conclude in section 6.

2 Ontology-Based Databases (OBDBs)

In the last years, many OBDB architectures have been proposed. We present first a proposed taxonomy of these architectures. Then, we discuss capabilities of existing OBDBs to solve the scalability and reasoning issues faced by the Semantic Web.

2.1 Classification of OBDBs

OBDBs recording different categories of data and in particular ontologies description and instance data, these data may be governed by various number of schemas. Thus we propose to classify OBDBs architectures according to the number of schemas used.

Type 1 OBDBs. In type 1 OBDBs, information is represented in a single schema composed of a unique triple table `(subject, predicate, object)` [14–17]. This table, called *vertical table* [12], may be used both for ontology descriptions and instance data. For ontology descriptions, the three columns of this table represent respectively subject ontology element identifier, predicate and object ontology element identifier. For example, the triple[1] `(Student, subClassOf, Person)` represents a subsumption relationship between classes `Student` and `Person`. For instance data, the three columns of this table represent respectively instance identifier, characteristic of an instance (i.e, property or class belonging) and value of that characteristic. For example, the triple `(Peter, grade, PhD)` represents the fact that Peter has a PhD grade. Figure 1 illustrates this approach. Figure 1 (a) presents a toy example of an ontology (upper part) with some instances (bottom part) as a graph. An extract of the corresponding vertical table is shown in Figure 1 (b).

Type 2 OBDBs. Type 2 OBDBs store separately ontology descriptions and instance data in two different schemas [3–5]. The schema for ontology descriptions depends upon the ontology model used to represent ontologies (e.g., RDFS, OWL, PLIB). It is composed of tables used to store each ontology modeling primitive such as classes, properties and subsumption relationships. For instance data, different schemas have been proposed. A vertical table can be used to store instance data as triples [5, 4]. An alternative is to use a *binary representation* where each class is represented by an unary table and each property by a binary table [3, 4, 1, 2]. Recently, *table per class representations* (also called *class-based representations*) have been proposed where a table having a column for each

[1] RDF uses URI for identifiers. For readability, we use names throughout this paper.

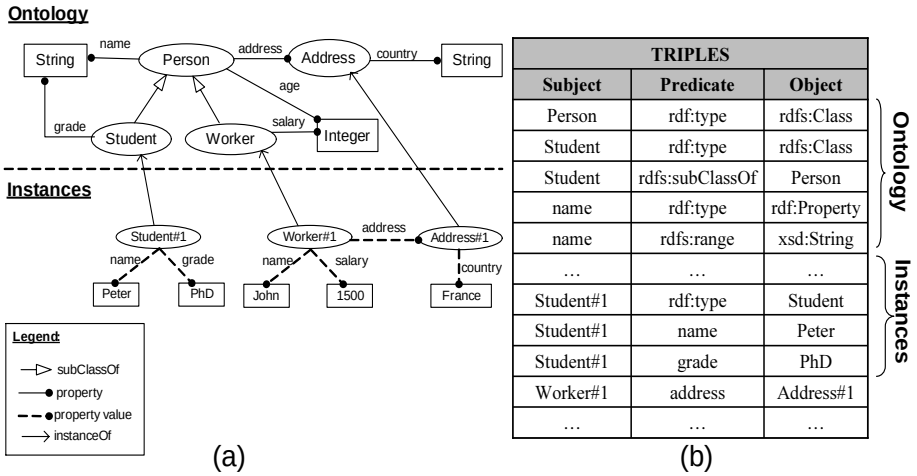


Fig. 1. Type 1 OBDBs approach

property associated with value for at least one instance of a class is associated to each class [7, 6]. These three basic approaches have also small variants (see [8] for details).

Figure 2 presents an example of type 2 OBDBs that stores data of our previous example (see Figure 1). In this example, ontology descriptions are stored using a schema for RDFS ontologies. In the bottom part, instance data are represented using a binary representation.

Type 3 OBDBs. OntoDB [7, 18] proposes to add another schema to type 2 OBDBs. This schema called *meta-schema* records the ontology model into a reflexive meta model. For the ontology schema, the meta-schema plays the same role as the one played by the system catalog in traditional databases. Indeed, meta-schema may allow: (1) generic access to the ontology, (2) support of evolution of the used ontology model, and (3) storage of different ontology models (OWL, DAML+OIL, PLIB, etc.). Figure 3 presents the meta-schema of our example.

These three categories of OBDB architectures behave differently according to the kind of information that need to be managed. In the next section we focus on their scalability capacity.

2.2 Scalability of OBDBs

Type 1 OBDBs. The vertical table approach raises serious performance issues when queries require many self-joins over this table [3]. To ensure a high performance of queries, each column of the vertical table shall be indexed [5]. Moreover, the predicate column shall be clustered [12] or materialized views need

Ontology

Class	
ID	Name
1	Person
2	Student
3	Worker
4	Address

SubClassOf	
Sub	Sup
2	1
3	1

Property	
ID	Name
1	name
2	age
3	grade
…	…

Domain	
prop	class
1	1
2	1
3	2
…	…

Range	
prop	type
1	xsd:string
2	xsd:integer
3	xsd:string
…	…

Instances

Person
ID

Student
ID
Student#1

Name	
ID	Value
Student#1	Peter
Worker#1	John

Grade	
ID	Value
Student#1	PhD

Address
ID
Address#1

Worker
ID
Worker#1

Salary	
ID	Value
Worker#1	1500

Country	
ID	Value
Address#1	France

Fig. 2. Type 2 OBDBs approach

to be created [1]. In both cases, these approaches require extra storage cost and lead to update overhead. And, even with such optimizations, several works have shown that in various conditions type 2 OBDBs outperform type 1 OBDBs [8, 5, 3].

Type 2 OBDBs. Performance of these OBDBs depend upon the representation used for instance data. Evaluation conducted in [12, 8, 1, 2] have shown that the vertical table approach for instance data suffers the same weaknesses as those encountered for Type 1 OBDB. Thus the binary representation has been considered for a long time as the best representation for instance data.

However, experimental results on the recently proposed table per class representations have challenged this idea [7, 6]. For queries where the class to be queried is specified, table per class representations outperform the classical binary table approach with ratio often bigger than 10, in particular when instances are associated with several properties [7]. Moreover, insertion and update are faster. The only case where the binary approach is better than table per class

Meta-Schema

Entity		
ID	Name	SuperEntity
1	Resource	
2	Class	1
3	Property	1

Attribute			
ID	Name	Domain	Range
1	name	1	1
2	domain	3	2
…	…	…	…

Type	
ID	Name
1	String
2	Entity#2

Fig. 3. Type 3 OBDBs Meta-Schema

representations is for queries where the class to be queried is not specified and that only request a very small number of property values.

Type 3 OBDBs. Addition of the meta-schema in OBDBs of type 3 doesn't improve performance of queries but increase functionalities as stated previously. Availability of ontologies in the database is in particular needed to implement the dynamic labeling scheme proposed in this paper.

This survey on scalability of OBDBs show that, for a number of use cases corresponding to the performed benchmarks, type 2 or 3 OBDBs using either binary representation or the table per class representation scale quite well and do support real size Semantic Web data management. The other challenge is to provide, at the same time, reasoning capabilities.

2.3 Deductive capabilities of OBDBs

Deductive capabilities are not the major strength of databases. To perform reasoning, two main approaches may be followed. The first approach consists in performing reasoning *before* query processing and to materialize all the deduced facts and, in particular, the transitive closure (TC) of all transitive relationships. We call this approach *eager reasoning*. This approach supports efficient query processing since reasoning is not required at runtime. Its drawback is extra storage cost and update overhead. The second approach consists in performing reasoning during query processing using virtual deduced facts to provide query results. We call this approach *lazy reasoning*. This approach is dual to the previous one: it requires extra cost for query processing but doesn't impose storage and update overhead.

Currently, OBDBs mainly support usual subsumption reasoning as specified in [19] (i.e, `subClassOf` and `instanceOf` relationships). Most of them perform lazy reasoning using different database mechanisms such as views [1], labeling schemes [6] or subtable relationship of object-relational databases [3, 4].

Some OBDBs address more complex reasoning. For example, ONTOMS supports instance reasoning for inverse, symmetric, and transitive properties [6]. As a rule, these most complex reasoning tasks are performed using eager reasoning and TC which lead to serious storage and update overhead in real-size applications. Other approaches propose to use logic engines (e.g Datalog engine) of deductive databases or OWL reasoners to perform these most complex reasoning tasks [9–11, 1]. However deductive databases have not found widespread adoptions outside academia and the response time of such architectures is often not compatible with person system interaction.

In fact, beside their capabilities to manage large size data, the major strength of database is their capability to process efficiently numeric queries and string-oriented queries. Thus efficient eager reasoning may be performed if a deductive reasoning may be replaced by numeric (or string-oriented) query processing. A well known application of this approach is the so-called labeling approach [12, 13] where transitive relationships are represented either by numeric intervals or by string (or bit vector) values. These applications are very efficient, as long as

the transitive relationship defines a tree structure. When it is a DAG, labeling becomes more complex and much less efficient. In the next section we propose a framework that integrate the various labeling schemes, and we show that the same framework may be efficiently used for DAG structure encountered in spatial and temporal application.

3 Numeric reasoning over partially ordered sets

3.1 A Motivating Example

The aim of the e-Wok Hub project[2] is to manage the memory of many engineering projects on the capture and storage of CO_2. In particular, an important objective is to improve the quality of documents search on this subject. The followed approach consists in using annotations of documents defined as much as possible by automatic means. As an example, we have focused on geographical aspects of CO_2 storage. An existing ontology, called COG[3], that describes the spatial French geographic entities, is used to annotate documents. In this ontology, a *spatial area* is represented by an ontology individual characterized by a `name`, a `type` (e.g, country, department or town) and `boundaries`. Moreover, spatial areas are organized in a tree structure using a transitive relationship named `subdivision`. This relationship has the following meaning: `x subdivision y` $\Leftrightarrow$ `y` $\subset$ `x`. Thus it defines a partial order on spatial areas.

Figure 4 presents an extract of the spatial areas tree of the COG. Each node represents a spatial area and each edge represents the subdivision relationship. The root of the tree is the country `France` which is subdivided in the departments `Ile de France` and `Poitou Charentes` areas. The latter is itself divided into the towns `Poitiers` and `La Rochelle`.

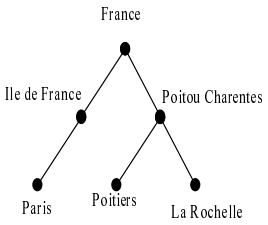


Fig. 4. COG ontology : example of inclusive relationship between individuals

Documents are automatically annotated using the COG. The annotation predicate is named `geolocalized_in`. The annotation (`doc geolocalized_in zone`) means that the document `doc` contains information about whole or part

[2] http://www-sop.inria.fr/acacia/project/ewok/index.html
[3] Code Officiel Gographique, http://rdf.insee.fr/geo/

of the spatial area `zone`. We note that this predicate has a particular behavior with respect to the `subdivision` order. If a document contains information about `Poitiers`, it contains information about some part of `Poitou Charentes`. Thus (`doc geolocalized_in Poitiers`) implies (`doc geolocalized_in Poitou Charentes`). Notice that all predicates whose range is spatial area don't have necessary this behavior. For example, the person who heads `Poitou Charentes` doesn't head the town of `Poitiers`.

This behavior has an impact on querying. Indeed, if one searches for all documents relevant to the spatial area `zone`, the system should reason over the inclusion relationship and return all documents annotated with the spatial areas included in `zone`. Eager or lazy reasoning approaches can be used to provide correct results. Eager reasoning consists in storing not only annotations defined by domain experts but also annotations that can be derived using the characteristic of `geolocalized_in`. Considering the high number of documents that may be managed combined with the number of French spatial areas (and of world spatial areas in a second step), eager reasoning requires a lot of storage space and would hardly scale. A naive lazy reasoning technique would be to compute the inclusion relationship TC using recursive operator of SQL99 (if available in the DBMS) or recursive stored procedures. Again, due to the large amount of documents combined with the number of areas, this approach would hardly scale in query processing response time. The inclusion relationship defining a tree structure, a classical labeling scheme may be used to represent in a compressed way the subsumption relationship TC. These techniques consist in assigning values to each node of a hierarchy according to the node's position. Figure 5 shows an application of *interval labeling scheme* on our previous example of the COG. On this tree, each spatial area is assigned a pair of integer values, `bound1` and `bound2`, that defines an interval. An area `zone1` is a (recursive) subdivision of `zone2` if the interval of `zone1` is included in the interval of `zone2`.

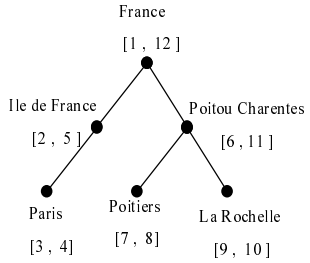


Fig. 5. translation of a tree structure into two numeric values

This approach has been implemented and scales perfectly since the annotation instances are quite stable. Thus, once all instances of the COG have been entered within the database, the labeling scheme does not need any change when new document annotations are recorded.

Unfortunately, when one annotes automatically documents, one doesn't only encounter names of countries, department and town. Other geographic areas names, such as regions, districts or localities, are also used. Note that a partial order still exists between all these spatial areas. But this order no longer defines a tree. It defines a DAG for which the interval labeling scheme is much less efficient. Moreover most existing labeling schemes need to be recomputed when the instances to be indexed are modified. We propose below a framework allowing to select various labeling schemes depending upon the problem at hand, and we introduce new labeling schemes for reasoning over spatial areas and temporal periods.

3.2 Proposed Framework

First, let us characterize formally the behavior of the `geolocalized_in` and `subdivision` relationships.

Let $\mathtt{E}$ and $\mathtt{F}$ be two sets ; $\mathcal{R} \subset \mathtt{E} \times \mathtt{F}$ and $\prec \subset \mathtt{F} \times \mathtt{F}$ be two binary relationships, with $\prec$ being an order relationship, i.e., reflexive, antisymmetric and transitive. We said that $\mathcal{R}$ is *propagated by* the order $\prec$ if and only if:

$$\forall \mathtt{x} \in \mathtt{E}, \forall \mathtt{y}, \mathtt{z} \in \mathtt{F},\ \mathtt{x}\ \mathcal{R}\ \mathtt{y} \wedge \mathtt{y} \prec \mathtt{z}\ \Rightarrow \mathtt{x}\ \mathcal{R}\ \mathtt{z}$$

and we call *propagated closure* (PC) of $\mathcal{R}$ by $\prec$, noted $\mathcal{R}^+_\prec$:

$$\mathcal{R}^+_\prec = \{(\mathtt{x}, \mathtt{z}) \in \mathtt{E} \times \mathtt{F} \mid \exists \mathtt{y} \in \mathtt{F},\ \mathtt{x}\ \mathcal{R}\ \mathtt{y} \wedge \mathtt{y} \prec \mathtt{z}\}$$

If $\mathcal{R} = \mathtt{geolocalized\_in\ in}$ and $\prec = \mathtt{subdivision}$, for a given geographic area, $\mathcal{R}^+_\prec$ contains all the documents that are annotated either by this area or by one of its (recursive) subdivisions.

Reasoning over Propagated Closure.

We note that $\mathcal{R}^+_\prec$ is the composition of the transitive closure of $\prec$, noted $\prec^*$, with $\mathcal{R}$: $\mathcal{R}^+_\prec = \prec^* \circ \mathcal{R}$. Thus efficient representation of $\prec$ TC would provide an efficient representation of $\mathcal{R}$ PC. We use a labeling scheme for that purpose. A labeling scheme $\mathcal{L}$ over $(\mathtt{F}, \prec)$ is a triple: $\mathcal{L} = (\mathtt{D}, \mathtt{label}, \mathtt{less\_or\_eq})$ where:

- $\mathtt{D}$ is an ordered ($\leq$) concrete domain;
- $\mathtt{label} : \mathtt{F} \rightarrow \mathtt{D}$ is a morphism of ordered sets:
  $$\forall \mathtt{x}, \mathtt{y} \in \mathtt{F},\ \mathtt{x} \prec \mathtt{y} \Rightarrow \mathtt{label}(\mathtt{x}) \leq \mathtt{label}(\mathtt{y})$$
- $\mathtt{less\_or\_eq} : \mathtt{D} \times \mathtt{D} \rightarrow \mathtt{Boolean}$ is a function that compares in constant time two values of $\mathtt{D}$:
  $$\forall \mathtt{a}, \mathtt{b} \in \mathtt{D},\ \mathtt{less\_or\_eq}(\mathtt{a}, \mathtt{b}) \Leftrightarrow \mathtt{a} \leq \mathtt{b}$$

Thus, if for all $\mathtt{y} \in \mathtt{F}$, `label(y)` is pre-computed and stored in the database and if `less_or_eq` may be computed in constant time (e.g., by numeric or string value comparison), the computation of the $\mathcal{R}$ PC may be done in linear time by a single traversal of the $\mathcal{R}$ relationship.

Topological and Geometrical Labeling Schemes.

Most labeling schemes that have been proposed use the topological structure of the lattice that represents the order over the $\mathtt{F}$ space to define the labels. For instance, as we have seen previously, the post-ordered interval scheme proposed

by Agrawal et al. [12], compute the numeric interval that labels each node by means of a post-order traversal of the spanning tree of the order relationship between all known instances. In the Bit Vector scheme, proposed by Wirth [20], the label of a node is represented by a vector of `n` bits where `n` is the number of instances of the `F` space. A `"1"` bit at some position uniquely identifies a node in the lattice structure and each node inherits the bits identifying its ancestor in the lattice. Thus these encodings are efficient as long as no major change occurs in the population of the `F` space, and, for the interval scheme, as long as the lattice is a tree. When signifiant changes occur in `F` instances, the labels need to be recomputed.

In fact, when reasoning over spatial or temporal domain, the underlying space has not only a topological structure but also a geometrical structure. Thus, it is associated with a metric that may be used for defining labels. Indeed, in Figure 4, both bounding rectangles and bounding circles might also be used for labeling geographic areas. When reasoning over geological periods where various geological time scales are used, an approximate mapping of each geological period onto geologic time (expressed in mya: "million of years ago") may be done. Notice that, unlike topology-based labels, these labels are absolute labels. They represent an additional knowledge that cannot be automatically computed from the known instances of `F` names or relationships, but they don't need to be changed when the content of `F` is updated. All these various labels may be represented as labeling scheme within OBDBs allowing efficient reasoning over PCs. We note that geometrical labels have two differences with topological labels : (1) they are invariant for a given instance, whatever other instances are considered, (2) they cannot be derived from non geometrical or non-temporal properties of an ontology individual. Thus, spatial and temporal properties are primitives ontological properties of temporal or spatial objects. As such, it is reasonable to consider that their values, for given individuals, may be either available in some place (e.g., through a web service) or exchanged together with the individual descriptions. My date of birth as well as the geolocalization of Paris are both ontological properties that are available somewhere and that could be managed in ontology-based data source. One difficulty is that geometrical description may involve complex data structure available only in specific systems (e.g., GIS). In fact, important geometric reasoning only needs very simple data. Spatial inclusion of convex bodies may be evaluated on the basis of bounding rectangles or of bounding circles. Temporal precedence just need to compare two float values or two intervals. Thus it is both possible to restrict the set of geometrical representations allowed and to support a large range of (approximate) spatial or temporal reasonings. Our suggestion is to support only interval (in one dimension (1D)), rectangle and circle (in two dimension (2D)).

4 Design and implementation

This section presents how our approach can be implemented in the different OBDB architectures to allow automation of the property propagation mecha-

nism. We assume now that `E` and `F` are two ontological classes. To represent that a property $\mathcal{R} : \mathtt{E} \times \mathtt{F}$ is propagated by a partial order $\prec$ over `F`, we need to represent: (1) the fact that $\prec$ is an order, (2) the labeling scheme of this order $\mathcal{L} = (\mathtt{D}, \mathtt{label}, \mathtt{less\_or\_eq})$, and (3) the fact that $\mathcal{R}$ must be propagated by $\mathcal{L}$. Existing ontology languages don't provide modeling primitives to represent these three pieces of information. As a consequence, both ontology models and OBDBs need to be extended. In an OWL database for instance, the first information needs to add a new value (named `orderProperty`) to the enumerated set of values of OWL property characteristics (`transitiveProperty`, `symmetricProperty`, etc.) since `antisymmetric` is available neither in OWL1 nor in OWL2. The third information needs to add a new value (named `propagatedBy`) to the single existing value of property-to-property relationship (`inverseOf`). Thus, these two information needs extension of ontology models. Finally, for the second information, we need to create two additional (meta-)tables in the OBDB. The first one describes the labeling schemes available in the OBDB. The second one defines which labeling scheme is assigned to a particular order property. These two tables are the extensions to OBDB that are required by our labeling model. Below, we outline the implementation process and discuss representation issues. The different steps of this implementation can be supported by both type 2 and type 3 OBDB architectures.

4.1 Extension of the Ontology Models part of OBDB

In this section, we present the information that needs to be recorded as n-ary tables. Notice that if binary representation is used, these tables must be splitted into binary tables. The two first tables represent information that we propose to add to ontology definition language. The two last tables are systems tables.
- Table 1 `property_characteristic` contains the characteristics of property. The required extension of ontology model is the capability to represent `orderProperty` as a characteristic.

Table 1. `property_characteristic` Table Columns

Column	Description
`propertyId`	refers to the unique identifier of the property in the property table.
`characteristic`	the characteristic of the property (for example `orderProperty`, `symmetricProperty`, etc.)

- Table 2 `property_to_property` contains the relationships between two properties. The required extension of ontology model is the capability to represent `propagated by` relation between a property and another property that defines an order. When a geometric labeling is used, and when this label is provided, for example as a bounding rectangle, together with the instance data, the inclusion relationship may often be implicit : it is to be computed by inclusion

of geometrical shapes defined by the geometric labels. In this case, the orderId is replace by a reserved word that may be `*geo_rectangle*, *geo_circle*, *geo_interval*`. This means that the propertyId is propagated by the inclusion (increasing) order of the corresponding geometric shapes. The columns that contain the geometric labels are named as specified in table 3. For a `propagated by` relationship, the column `direction` specifies whether the propagation is done in a *direct* way (the same direction as the order property) or in a *reverse* way. For example, `applicable laws` in Poitou Charentes include those defined in areas encompassing Poitou Charentes; this implies a propagation in the reverse way as compared to the order property subdivision, on the contrary of the property `geolocalized in` where the propagation is done in a direct way.

Table 2. `property_to_property` Table Columns

Column	Description
`PropertyId`	the unique identifier of the propagated property.
`orderId`	the unique identifier of the property in `property_characteristic` table or a reserved word (`*geo_rectange*`, ...)
`relationName`	the name of the semantic relation linking the two properties; for example `propagated by`, `inverse of`
`direction`	defines the propagation direction according to the order relationship. Allowed values : `direct`, `reverse`.

- Table 3 `labeling_scheme` contains information about the various labeling schemes available in the particular OBDB.
This table is supposed to be defined by the database administrator (DBA). Nevertheless, it contains lines whose four first attributes have predefined content. Theses lines specify how a geometrical labeling must be identified to be recognized by the system. These lines are defined in Table 4.
- Table 5 `property_schemes` contains information about the various scheme associated to each particular property. This table is automatically generated by the system. When a `propagated_by` property is introduced in table 2, the default labeling scheme defined in table 3 is automatically implemented if the orderId identifies a property. If the orderId is a reserved word the system just checks that the needed labeling columns are presents and a line in table 5 is also added. The DBA may change the default labeling scheme when needed.

4.2 Representation of Individuals

Individuals of ontology classes will be represented according to the data structure strategy used in each particular OBDB. To automate the generation of specifics properties used to record labels, they are manage like other properties and they are initialized to `NULL` if no value is provided.

Table 3. `labeling_scheme` Table Columns

Column	Description
`schemeId`	refers to the unique identifier associated to the labeling scheme
`numberOfColumns`	the number of columns used to represent the domain `D` (for example 2 for the interval labeling scheme)
`listColumnsSuffixes`	a list of column's suffixes used to represent `D` (for example {`bound1`, `bound2`})
`listColumnsTypes`	a list of column's types associated to the column's names in `listColumnsNames` (for example {`int`, `int`})
`label`	the optional name of the SQL/PSM function to be used for computing the label associated with those instances whose labels value equals `NULL`. This function is called on `F` each time one or a set of new instances of `F` are entered in the database within the same transaction. This name does not exist (`NULL`) when the label must be provided externally with each instance (e.g., for geometrical labeling schemes).
`less_or_eq`	the name of the SQL/PSM boolean function to be used for evaluating if one individual is smaller or equal to another for the order defined by propertyId. If $\mathcal{L}$ is the interval labeling scheme over the `F` space, and `i1` and `i2` are two individuals, `i2` $\prec$ `i1` if the call `less_or_eq(i2.bound1, i2.bound2,i1.bound1, i1.bound2)` returns `true`.
`defaultScheme`	a boolean value. The default scheme to be associated to a new property defining an order.

Table 4. predefined labeling scheme in the `labeling-scheme` table

schemeId	numberOfColumns	listColumnsSuffixes	listColumnsTypes	...
`*geo_interval*`	2	{bound1, bound2}	{float, float}	...
`*geo_rectangle*`	4	{xmin, xmax, ymin, ymax}	{float, float, float, float}	...
`*geo_circle*`	3	{xcenter, ycenter, radius}	{float, float, float}	...

Table 5. `property_schemes` Table Columns

Column	Description
`propId`	the unique identifier of the order property in `property_characteristic` table or to the identifier of a propagated property when its orderId is a reserved word.
`schemeId`	the unique identifier of the scheme
`listProperties`	the list of identifiers of the properties associated to `listColumnsNames` in the class `F`
`activeScheme`	a boolean value. `true` if the scheme is active.

4.3 Representation of Annotations

Two strategies can be used for representing annotations :

- using a distinct binary table for each annotation property:
  - the first column `resourceId` refers to the unique identifier of a resource;
  - the second column `individuaId` refers to the unique identifier of the ontological instance used to annotate the resource.

  Using this representation, one or several joins (depending on the structure of the data part) will be necessary during query processing to retrieve label property(ies) value(s) associated to the property defining the order relation;
- using a distinct materialized view for each annotation property containing `resourceId` and `individualId` columns as defined above, but also all the columns in `listColumnsNames` labeling scheme representation. Notice that if materialized views are used, a management policy must be defined for data updating.

4.4 Queries Processing

Our goal is to support fully automatic numeric reasoning for queries using properties that have specific characteristic like order relation or propagation by an order. Each incoming query must then be treated by an OBDB interpreter as follows:

- identification of the query category: each ontology query must be analyzed at the ontological level in order to determine whether it involves or not properties with specific characteristics. This will be done using tables `property_characteristic`, `property_to_property` and `property_schemes`;
- query interpretation: if the query does not require special treatment, it will be processed as usual; if not, the query will first be translated into a numeric query using information stored in the tables `labeling_scheme` and `property_schemes`. This translation also depends on the annotation representation. The resulting rewritten query will then be efficiently processed by the database.

Figure 6 summarizes the different steps followed to process query. Below, we describes the effective implementation of our approach in OntoDB OBDB.

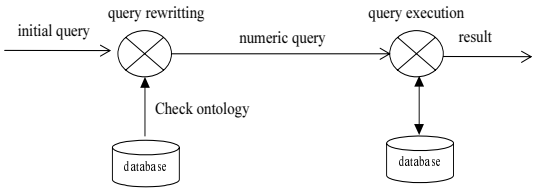


Fig. 6. Query processing steps.

5 Application to the COG Ontology in OntoDB Database

This section describes how our approach has effectively been implemented on the OntoDB database[7, 18] using the COG ontology by means of the OntoQL query language [21]. Only the bounding rectangle and the numeric interval labeling schemes have been implemented. First we briefly present the OntoDB database.

5.1 OntoDB

OntoDB is a type 3 OBDB designed to support evolutions of the ontology schema, and to offer data access at the ontology level. Currently, OntoDB is implemented on top of Postgres. It consists of 4 parts. Parts 1 and 2 are the traditional parts available in all DBMSs, namely the data part that contains instance data and the meta-base part that contains the system catalog. Parts 3 (ontology) and 4 (meta-schema) are specific to OntoDB. Ontology-based data are represented in OntoDB using an horizontal approach; one table is created for each ontological class; its columns consists of a subset of the class applicable properties (i.e, that include the class in their domain), namely those that are used by at least one instance of the class. This representation scales well when numerous properties per instances are used [7].

5.2 Implementation

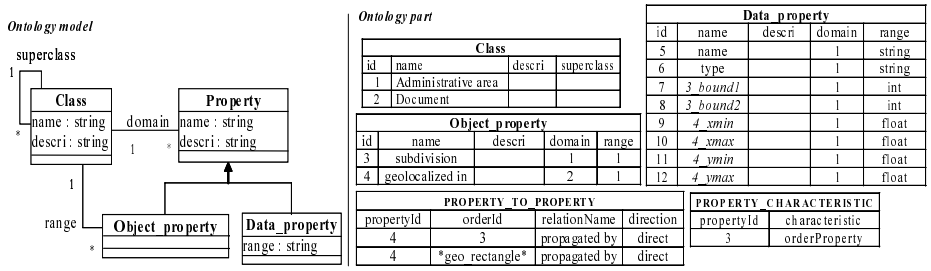


Fig. 7. COG - OntoDB : ontology part

The COG ontology model is stored in the ontology part of OntoDB. Figure 7 shows the content of the tables of the ontology part using a simplified ontology model. This ontology model is instantiated in the meta-schema part, this allows to automate the generation of the structure of the ontology part using a model transformation. When the property `geolocalized_in` is inserted in the `property_to_property` table, if the orderId referenced is `subdivision` and if the interval labeling scheme is active as the default scheme, then, the system will automatically add the properties `bound1` and `bound2`, required by the default labeling scheme, to its range `spatial area`. When the property `geolocalized_in`

is inserted again in the `property_to_property` table, then, if the orderId referenced is `*geo_rectangle*`, the system checks that properties xmin, xmax, ymin, ymax are provided in its range `spatial area` and generates a new line in the `labeling scheme` table. We have extended the ontology part of OntoDB with the tables `labeling_scheme` and `property_schemes` described in section 4.1 and instantiated them with appropriate information as shown in Figure 8.

LABELING_SCHEME

schemeId	numberOfColumns	listColumnsNames	listColumnsTypes	label	less_or_eq	defaultScheme
5	2	{bound1, bound2}	{int, int}	interval	include	TRUE
geo_rectangle	4	{xmin, xmax, ymin, ymax}	{float, float, float, float}	NULL	MBRContains	FALSE

PROPERTY_SCHEMES

propId	schemeId	listProperties	activeScheme
3	5	{7,8}	TRUE
4	*geo_rectangle*	{9,10,11,12}	FALSE

Fig. 8. COG - OntoDB : system extension of the ontology part

Once the COG ontology model and property characteristics and relationships have been represented, individuals have also to be represented in the data part using the OntoDB strategy (table per class representation) as shown in Figure 9. Labels values for individuals can either be assigned outside the OBDB, for predefined labeling schemes, or computed automatically by the OBDB, for topological labeling schemes (e.g. for scheme 5, the `interval` function is automatically triggered by the system). OntoDB uses the ontological class IDs to generate the tables names and property IDs to generate columns names in the data part. This mechanism establishes the link between the ontology part and the data part. For better readability, we use the names (names of classes, names of properties, etc.) instead of IDs. Currently, we have only implemented topological labeling schemes by numeric interval and geometrical labeling by bounding rectangle as it is the case for the property `geolocalized_in` and the relation `subdivision`. The values of the coordinates of bounding rectangles being not currently available in the COG ontology, they were fetched from another source and enter into OntoDB.

subdivision

Subdivision1	Subdivision2
France	Ile de france
France	Poitou Charentes
Ile de france	Paris
Poitou Charentes	La rochelle
Poitou Charentes	Poitiers

spatial area

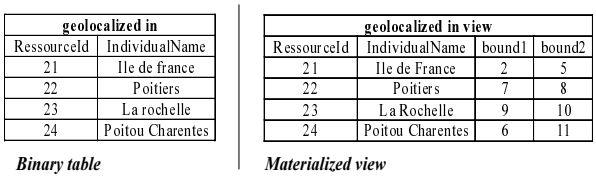
id	name	type	...	bound1	bound2
11	France	country		1	12
12	Ile de France	department		2	5
13	Paris	town		3	4
14	Poitou Charentes	department		6	11
15	Poitiers	town		7	8
16	La Rochelle	town		9	10

Document

id
21
22
23

Fig. 9. COG - OntoDB : the data part

As mentioned in section 4.3, two strategies can be used to represent annotations. Figure 10 shows the implementation of the two strategies in OntoDB. In the second strategy using a materialized view, for each annotation only the values `bound1` and `bound2` of the individuals are duplicated because scheme 1 (interval) is the active labeling scheme for the property 3 (`subdivision`).



geolocalized in

RessourceId	IndividualName
21	Ile de france
22	Poitiers
23	La rochelle
24	Poitou Charentes

Binary table

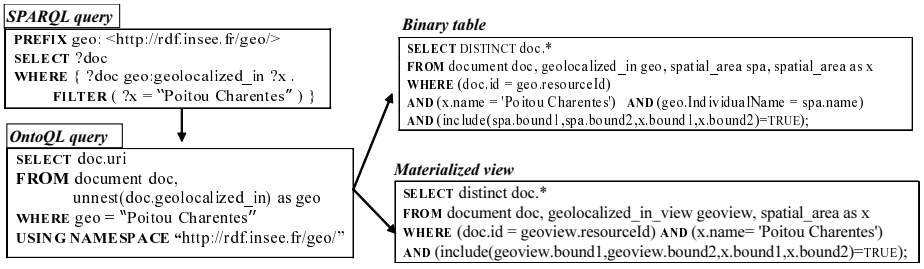
geolocalized in view

RessourceId	IndividualName	bound1	bound2
21	Ile de France	2	5
22	Poitiers	7	8
23	La Rochelle	9	10
24	Poitou Charentes	6	11

Materialized view

Fig. 10. COG - OntoDB : representation of annotations

We can now turn to the transformation of queries so that they can be efficiently performed within OntoDB using numeric reasoning capabilities of Postgres. The incoming queries (that may be, for instance, rather simple SPARQL queries as it is done in the eWok-Hub project) are first rewrited as OntoQL query, an SQL-like query language [21]. Then we have classified incoming queries in two groups; queries which do not require a special treatment and query which need to be rewritten. This classification is done referring to the tables `property_to_property` and `property_scheme` as mentioned in section 4.4. Figure 11 shows an example of query rewriting where an incoming SPARQL query is rewritten into a numeric query according to the binary table representation and the materialized view representation.



SPARQL query

```
PREFIX geo: <http://rdf.insee.fr/geo/>
SELECT ?doc
WHERE { ?doc geo:geolocalized_in ?x .
        FILTER ( ?x = "Poitou Charentes" ) }
```

OntoQL query

```
SELECT doc.uri
FROM document doc,
     unnest(doc.geolocalized_in) as geo
WHERE geo = "Poitou Charentes"
USING NAMESPACE "http://rdf.insee.fr/geo/"
```

Binary table

```
SELECT DISTINCT doc.*
FROM document doc, geolocalized_in geo, spatial_area spa, spatial_area as x
WHERE (doc.id = geo.resourceId)
AND (x.name = 'Poitou Charentes') AND (geo.IndividualName = spa.name)
AND (include(spa.bound1,spa.bound2,x.bound1,x.bound2)=TRUE);
```

Materialized view

```
SELECT distinct doc.*
FROM document doc, geolocalized_in_view geoview, spatial_area as x
WHERE (doc.id = geoview.resourceId) AND (x.name= 'Poitou Charentes')
AND (include(geoview.bound1,geoview.bound2,x.bound1,x.bound2)=TRUE);
```

Fig. 11. COG - OntoDB : example query rewriting

6 Conclusion

Realization of the Semantic Web vision requires scalable ontology data management tools that perform reasoning operations over ontology-based annotations

in an acceptable response time. In this paper we have first described the current state of the art of OBDBs that allow to store both ontology descriptions and instance data in a database. If some OBDB architectures scale quite well for various applications, reasoning capabilities are mainly provided by representing explicitly all the facts that may be deduced by a reasoner. This may lead to a strong storage overhead for real-world applications. As an alternative we have proposed an approach that consists in enriching the ontology instances used as annotations with new property values in order to replace deductive reasoning by numeric (or string-oriented) query processing.

The kind of reasoning we have addressed is both transitive reasoning over partially ordered sets and reasoning over composition of two properties, the second one being transitive. These cases encompass evaluation of taxonomy-valued properties, subsumption reasoning and spatial and temporal inclusion reasoning. We have proposed a framework that allows to characterize such cases at the ontology level allowing OBDB systems to implement dynamically numeric reasonings when such ontologies are loaded. This framework consists of three pieces of information, each of them corresponding to extension of existing ontology models:

- the fact that a property defines an *order* i.e, transitive, reflexive and antisymmetric, or a *tree-order* i.e, also inverse functional;
- the fact that one property may be *propagated* by another property that is transitive;
- a labeling scheme allowing to specify what kind of labeling should be used to replace deductive reasoning by numeric (or string-valued) reasoning.

Two kinds of labeling scheme exist. Topological labeling schemes correspond to the various labeling schemes already proposed for tree structured or DAG-structured data. Such labels may be computed by the OBDB system. Our approach allows both to specify in a declarative way which particular scheme must be used for a particular property and to define which default scheme must be used when no scheme is specified. Geometrical labeling schemes are used for spatial or temporal reasoning. The labels values must be provided to the system as properties, but our approach allows to specify which properties must be used as labels. In both case, the OBDB query interpreter may automatically rewrite query to make profit of the labeling. This approach has been implemented in the OntoDB OBDB and we have presented the two approaches that may be used for rewriting queries.

Acknowledgment. The research described in this project was supported by ANR under Grant 05RNTL02706, eWok-Hub. The authors want to thanks all members of the project for fruitful discussions. A particular thanks to Eric Sardet, CRITT Informatique, who contributed to the design and implementation on OntoDB.

References

1. Pan, Z., Heflin, J.: DLDB: Extending Relational Databases to Support Semantic Web Queries. In: Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03). (2003) 109–113
2. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07). (2007) 411–422
3. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Tolle, K.: The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In: Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb'01). (2001) 1–13
4. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: Proceedings of the 1st International Semantic Web Conference (ISWC'02). (2002) 54–68
5. Ma, L., Su, Z., Pan, Y., Zhang, L., Liu, T.: RStar: an RDF Storage and Query System for Enterprise Resource Management. In: Proceedings of the 30th International Conference on Information and Knowledge Management (CIKM'04). (2004) 484–491
6. Park, M.J., Lee, J.H., Lee, C.H., Lin, J., Serres, O., Chung, C.W.: An efficient and scalable management of ontology. In: Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07). (2007) 975–980
7. Dehainsala, H., Pierra, G., Bellatreche, L.: OntoDB: An Ontology-Based Database for Data Intensive Applications. In: Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07). (2007) 497–508
8. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking Database Representations of RDF/S Stores. In: Proceedings of the 4th International Semantic Web Conference (ISWC'05). (2005) 685–701
9. Mei, J., Ma, L., Pan, Y.: Ontology query answering on databases. In: Proceedings of the 5th International Semantic Web Conference (ISWC'06). (2006) 445–458
10. Volz, R., Staab, S., Motik, B.: Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases. Journal of Data Semantics II **3360** (2005) 1–34
11. Borgida, A., Brachman, R.J.: Loading data into description reasoners. SIGMOD Record **22**(2) (1993) 217–226
12. Agrawal, R., Somani, A., Xu, Y.: Storage and Querying of E-Commerce Data. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01). (2001) 149–158
13. Christophides, V., Plexousakis, D., Scholl, M., Tourtounis, S.: On labeling schemes for the semantic web. In: Proceedings of the 12th International World Wide Web Conference (WWW'03). (2003) 544–555
14. Harris, S., Gibbins, N.: 3store: Efficient Bulk RDF Storage. In: Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03). (2003) 1–15
15. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: Proceedings of the 1st International Workshop on Semantic Web and Databases (SWDB'03). (2003) 131–150

16. Chong, E.I., Das, S., Eadon, G., Srinivasan, J.: An Efficient SQL-based RDF Querying Scheme. In: Proceedings of the 31st international conference on Very Large Data Bases (VLDB'05). (2005) 1216–1227
17. Petrini, J., Risch, T.: SWARD: Semantic Web Abridged Relational Databases. In: Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA'07). (2007) 455–459
18. Pierra, G., Dehainsala, H., Aït-Ameur, Y., Bellatreche, L.: Base de Données à Base Ontologique : principes et mise en œuvre. Ingénierie des Systèmes d'Information **10**(2) (2005) 91–115
19. Hayes, P.: RDF Semantics. World Wide Web Consortium. (2004) http://www.w3.org/TR/rdf-mt/.
20. Wirth, N.: Type extensions. ACM Transactions on Programming Languages and Systems (TOPLAS) **10**(2) (1988) 204–214
21. Jean, S., Aït-Ameur, Y., Pierra, G.: Querying Ontology Based Database Using OntoQL (an Ontology Query Language). In: Proceedings of On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences (ODBASE'06). Volume 4275 of Lecture Notes in Computer Science., Springer (2006) 704–721