Proc. of Software Engineering and Knowledge Engineering (SEKE2006), San Francisco Bay, USA, July 5-7, 2006.

Querying ontology based databases. The OntoQL proposal

Stéphane Jean, Yamine Aït-Ameur, Guy Pierra
LISI/ENSMA BP 40109, 86961 Futuroscope Cedex, France
E-mail: {jean,yamine,pierra}@ensma.fr

Abstract

Several approaches for storing ontologies and their instances in databases have been proposed. As a consequence, the need of defining languages to query such ontology based databases (OBDB) appeared. In this paper, we present OntoQL, an ontology query language designed for OBDB databases. This paper particulary emphasizes on the language constructs. Several query examples showing the interest of this language compared to traditional database query languages are given. Finally, we overview the proposed OntoQL implementation on a prototype of OBDB.

1 Introduction

Nowadays, ontologies are used in a lot of diverse research fields. They provide with the capability to represent a huge set of information contents. Several data models recommending the use of ontologies to describe data and their semantics have been developed. These models consider that an user or a system is able to retrieve the definition, meaning, translation and/or identifier of a given data item corresponding to a given data concept stored in an ontology.

Therefore, the idea of describing the ontology as well as the database model in a single model emerged in several work [18, 2, 16, 10, 5]. We call Ontology Based Database (OBDB) the database models allowing to store the ontology and the data in a common and single data model.

The goal of this paper is to describe OntoQL, an ontology query language for an OBDB model, called OntoDB, designed according to a database approach. In this paper, we present the basic constructs of this language and an overview of an implementation of this language on a prototype of OBDB.

This paper is structured as follows. Sections 2 gives a brief state of the art of OBDB models. Section 3 presents the OBDB model addressed in this paper. Section 4 introduces the OntoQL language by a set of queries on an example of ontology based database. Section 5 overviews the development conducted around the OntoQL language. Section 6 discusses related work. Section 7 concludes the paper by summarizing the main results and suggesting future work.

2 OBDB models: state of the art

Storing ontologies and their instances in databases has been the subject of several research studies and proposals. In the context of the Semantic Web, several OBDB models [2, 5, 16, 10] have been proposed to manage data described by ontologies represented in the standard ontology models RDFS [14] or OWL [3]. In these approaches, an instance, often called an individual, has its own property and class structure. Therefore, to manage instances, a generic database schema, not meaningful to a user and not customizable, is used. The simplest and more general one uses an unique table of triples [10] for storing both the ontology and its instances. Other approaches [2, 5] separate the representation of the ontology and its instances in two parts. The most common practice for storing instance data is to used the so-called vertical model [1] where information is stored in triples `(subject, property, value)` a variant of which, called the binary model is to have one table per property that contains only pairs of the form `(subject, value)`.

Our approach differs from the ones listed previously. A table is associated to each class of the ontology. The columns of these tables correspond to the properties of these classes. Before presenting the OntoQL query language exploiting this OBDB proposal for storing ontologies and their instances, let us describe the OntoDB OBDB model.

3 The OntoDB ontology based database model

The OntoDB model is composed of two main parts. The *ontology* part storing ontology definitions and the *content* part storing the instances which descriptions and semantics are described by the stored ontologies. In order to illustrate the OntoQL proposal, a practical toy example is shared by all the sections of this paper.

3.1 The ontology part

The *ontology* part gathers the basic shared constructions of the PLIB [17], RDFS [14] and OWL [3] ontology mod-

els. Some of these constructions are issued from object oriented modelling.

- Each ontology concept has an unique identifier (*oid*);
- Each ontology concept is an instance of the *concept* entity of the ontology model.
- Each ontology concept is described by attributes values whose types may be either primitive types, association or collection types.
- The entities of the ontology model may be linked by an inheritance (multiple) relationship.

The *ontology* part may be described by the simplified UML class diagram of figure 1. This figure shows that an ontology contains a set of concept definitions gathered by classes and properties. An ontology defines a mechanism for uniquely identifying concepts (`name spaces`). Each class is described by a set of multi-lingual attributes (`name`, `definition` ...). Classes are linked by an inheritance relationship (`superClasses`). A set of properties may be used to characterize instances of classes (`properties`). Each property is defined in the scope of a class for which it is applicable (or of a superclass of this class) (`scope`).

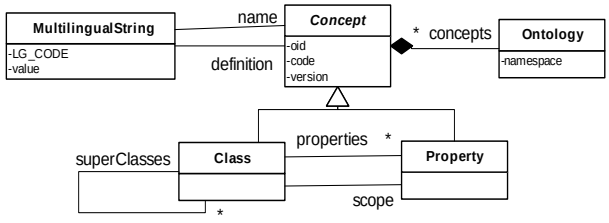

Figure 1. An UML model for the "ontology" part of an OBDB

Figure 1 represents the kernel of the ontology model addressed by the OntoQL language presented in this paper. This kernel can be extended by specific features related to any specific ontology model. For example, in the PLIB ontology model, classes are also described by *illustrations*, *external files*, *notes*, *remarks*, ... which are added to this kernel.

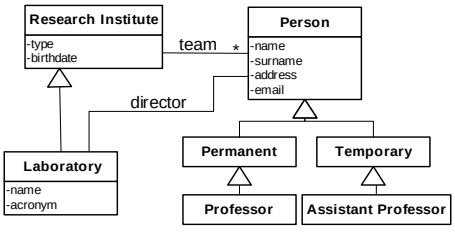

Figure 2. An example of ontology

In order to illustrate our model, let us consider a simple example describing data related to research institutions (figure 2). Each `research institute` is composed of research `teams` with several `persons` (`permanent` or `temporary`). These persons have different statuses (`professor`, `assistant professor`, `associate professor`, ...). Among the research institutes, we distinguish `laboratories` that are characterized by an `acronym` and headed by a `director`.

3.2 The content part

The *content* part contains data and their logical schemas. However, contrary to classical databases, the logical schemas are linked to the ontology. Several work have studied this link [19]. In our approach, the logical schemas are built from a subset of the ontology. Therefore, different logical schemas can be derived from a single ontology.

An user determines which classes must be represented in the content part (extension of the class), and for each class the user determines which available properties (thanks to the scope attribute) are needed to describe the instances of a class. A table is derived to represent these instances. The link between this table and the ontology is automatically built and stored in the database.

Figure 3 presents a simple logical schema of a database that can be built from the defined ontology. In this example, the chosen properties are `name`, `director` and `team` for the `Laboratory` class and `name`, `surname` and `email` for the `Person` class. The property `team` being $n - n$, this schema contains the intermediate table `Laboratory_Team`.

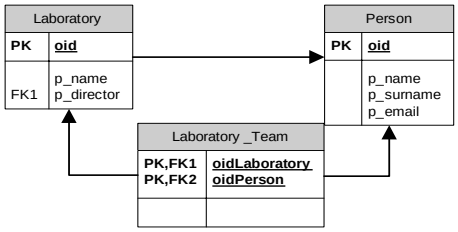

Figure 3. An example of logical database model for content part

3.3 Yet another database language ?

The answer to the question *Why another database language ?* is found in the specificities of the ontology based database model chosen to encode ontologies and their instances. Indeed, not all the properties of an ontology class are valuated in the content part. Therefore, it is necessary to store on the one hand the ontology in order to get the class and properties descriptions and on the other hand the logical schema (may be more than one logical schemas) storing the content and instances of classes and values of properties.

Having defined the ontology and their content, we are able to present the OntoQL language we have developed.

4 The OntoQL language

The OntoQL language is defined from the traditional components available in database exploitation languages. This section presents the language in a syntactical way and gives relevant queries on our example of ontology.

4.1 The data definition language

4.1.1 Definition of the ontology part

OntoQL provides with the resources to create, update and delete concepts of an ontology (classes, properties, ...) and of attribute values (names, definitions, ...). The following instruction creates a class with an English name `Laboratory`. It extends the class `Research Institute` and gives `definitions` and `names` in other languages (French and Spanish). Four properties are defined and created: the `name` of the laboratory, its `director`, its `acronym` and its `team`.

```
CREATE CLASS Laboratory  EXTENDS "Research Institute"(
  DESCRIPTOR (
    #name[fr,es] = ('Laboratoire','Laboratorio'),
    #definition = 'workplace for conducting
                                  research activities',
    #definition[fr] = 'lieu pour mener des recherches')
  PROPERTIES (
    name String, director Person,
    acronym String, team SET OF Person));
```

Two specific clauses are introduced: **`DESCRIPTOR`** for introducing class attributes and descriptions (prefixed by #) and **`PROPERTIES`** to describe the relevant characterization properties of the concept. The valuations of these properties define the instances of concepts.

A property may be modified. The following clause changes the English `name` and adds an `illustration` to the `director` property.

```
ALTER PROPERTY director ADD DESCRIPTOR (
  #name[en] = 'headmaster',
  #illustration='headmaster.jpg');
```

4.1.2 Definition of the content part

The extent of a class is defined from the ontology by choosing which properties are valued for a given class.

```
CREATE EXTENT OF Laboratory (name, team, director);
```

This clause creates a container (table) of instances of the class `Laboratory` with the properties `name`, `team` and `director`. The link with the ontology and its concept definitions is also kept in the database.

4.2 The data manipulation language

When the extent of a class is defined, like in SQL3 [7], class instances can be inserted, deleted and updated. The following clause creates a new instance of the `Laboratory` class.

```
INSERT INTO Laboratory (name, acronym)
     VALUES ('Laboratoire d'Informatique Scientifique
                          et Industrielle', 'LISI');
```

The properties valued in an **`INSERT`** clause may be not described in the extent of a class (the `acronym` property in the previous clause). In this case, OntoQL offers three options: 1) either a *NULL* value is inserted or 2) an error is returned and the clause is rejected or 3) the extent of the class is completed by a new property and all the values of this property are completed with *NULL* values for the other instances.

To manipulate a property which type is another class, there is need to use nested clauses. For example,

```
UPDATE ontology_sic:Laboratory
   SET director =
     (SELECT p FROM p in ontology_sic:Person
      WHERE p.name='LIENHARDT' and p.surname='Pascal')
 WHERE acronym = 'SIC';
```

modifies the `director` of the `laboratory` which `acronym` is `SIC` by retrieving the `director` in a class `Person` of the ontology `ontology_sic`.

4.3 The data query language

This section reviews the most important and novel aspects of the querying part of the OntoQL language. We focus on specific constructions that have emerged from the ontology based database model we have implemented. The language offers the possibility to query ontologies, contents (instances) and both ontology and content thanks to the implementation of the link between these two parts.

4.3.1 Querying the ontology

The ontology model of figure 1 contains some object oriented constructions. Therefore, and in order to keep aligned with relational object database languages, OntoQL proposes a syntax close to the one of languages like OQL [6] or SQL3[7].

Object oriented database constructions of OntoQL

Path expressions. OntoQL allows the use of *path expressions* in a **`SELECT`** clause. The following query is used to retrieve the identifier of the class domain of a property which identifier is `7B13543`.

```
SELECT #scope.#oid FROM #property WHERE #code='7B13543'
```

Manipulation of collections. To traverse collections, OntoQL expresses *dependent collections* in the **`FROM`** clause. Next query retrieves the code and version of classes defining a property named `acronym`.

```
SELECT c.#code, c.#version
  FROM c in #class, p in c.#properties
 WHERE p.#name = 'acronym'
```

In our example, the previous query returns the `code` and the `version` of the class `Laboratory` and its potential subclasses.

Moreover, queries may be *nested* in the clauses **SELECT**, **FROM** or **WHERE**. The following query returns the `name` and the `properties` of all classes which name starts by the letter `"A"`.

```
SELECT c.#name, (SELECT p.#name FROM p in #property
                 WHERE p.#scope=c)
  FROM c in #class
 WHERE c.#name like 'A%'
```

OntoQL provides with quantification operators **ALL** and **EXISTS** like in the clause

```
SELECT c.#name FROM c in #class
 WHERE '001' < ALL (SELECT p.#version
                    FROM p in c.#properties)
```

which returns the name of classes having all applicable properties with a version greater than `001`.

Finally, OntoQL is equipped with aggregate operators (**count, sum, avg, min, max**), access to the i-th element of an indexed collection, sorting (**ORDER BY**) and set operations (**UNION, INTERSECT, EXCEPT, GROUP BY**).

Specific OntoQL constructions.

Significant differences between the ontology model and object oriented models are available in the ontology model we defined. These differences led to specific constructions of the OntoQL language.

The multi-lingual aspect is one of these differences. Translations of the attributes may be defined. For example, the following query returns the names in French and in English of all the available classes.

```
SELECT #name[FR,EN] FROM #class
```

A second difference is the availability of internal (known by database implementors) and external (known by users, e.g., URI) identifiers for all the concepts more their names. Each of these identifiers may be used in an OntoQL query. For example, the names can be used to retrieve other elements of the ontology. The following query uses the name of a class to retrieve the French names of all its properties.

```
SELECT #name[FR] FROM "Research Institute".#properties
```

Notice that, for efficiency, the OntoQL engine uses the internal identifiers to run this query.

Moreover, name spaces may be used to refer to elements of an ontology. The **USING** clause is used for this purpose, it refers to a given name space outside the current one. The previous query, run on the `sic_ontology` name space, can be written as:

```
SELECT p.#name[FR]
FROM ns_sic:Laboratory.#properties
USING NAME_SPACE ns_sic AS sic_ontology
```

We have shown how OntoQL manages the *ontology* part and allows an user to retrieve the relevant descriptive information available in an ontology. Let us describe now, how *contents* may be queried.

4.3.2 Querying the content

OntoQL allows the query of contents. Moreover, since the content is linked to the ontology, querying the content does not rely on any specific logical database model. Therefore, two applications sharing a common ontology will have the right to run a common query even if the underlying logical database models are different.

Before giving the intuition of the content querying, let us recall some basic characteristics of instances.

- Each instance has an unique identifier (*oid*).
- Each instance has a basic class in the ontology.
- Each instance is described by the values of the properties defined in the extent of the class.
- Ontology classes may be linked by an inheritance relationship.

Querying the content will be similar to querying the ontology, except that that properties will not be prefixed by the # symbol.

Next query returns the `laboratory names` whose members do not have homonymous `name` with the `director` of this `laboratory`.

```
SELECT l.name
  FROM l in Laboratory
 WHERE l.director.name <> ALL (SELECT member.name
                               FROM member in l.team)
```

Notice that the same query written in SQL on the logical model presented on figure 3 is much more complicated. It requires to retrieve all the tables associated to the class `Laboratory`.

A polymorphic search operator * allows a query to retrieve the instances of a class and of all its subclasses. As illustrated below, the first query retrieves the names of instances which basic class is `Research Institute` while the second one returns the names for the `Laboratory` class as well.

```
SELECT name FROM "Research Institute"

SELECT nom FROM "Institut de Recherche"*
```

Notice that the second query of the previous example is written in French. This possibility is provided thanks to the capability of the ontology to support multilingual attribute names and concept language translations.

4.3.3 Querying both ontology and content

Since the links between ontologies and their contents are kept in the Ontology Based Database model, OntoQL has exploited this capability to allow querying both ontology and content in the same query.

From ontology to content

To query contents, OntoQL suggests an iterator `i` on instances of a class `C` by writing `i in C` or `C as i`. Next query returns the identifiers of instances belonging to the polymorphic extent of a class which French `name` begins with the string `"Per"`.

```
SELECT i.oid
  FROM C in #class, i in C*
 WHERE C.#name[fr] like 'Per%'
```

Moreover, it permits to retrieve and/or use the values of a property discovered by the query itself on the ontology part. The following query allows the retrieval of the values of the properties of the instances obtained in the previous query.

```
SELECT i.oid, p.#name[fr], i.p
  FROM C in #class, p in C.#properties, i in C*
 WHERE C.#name[fr] like 'Per%'
```

The previous query returns a tuple per property, but to obtain a single tuple per instance, one can write:

```
SELECT i.oid, (SELECT i.p, p.#name[fr]
               FROM p in C.#properties)
  FROM C in #class, i in C*
 WHERE C.#name[fr] like 'Per%'
```

From content to ontology

OntoQL proposes the use of the **`typeOf`** operator to make distinction between properties and retrieve information from the ontology part starting from the content part. This **`typeOf`** operator is implemented thanks to the link between ontology and content stored in the OBDB database. For example, the following query

```
SELECT i.name, i.surname, typeOf(i).#name[fr]
  FROM i in Person*
```

returns the French `name` of the basic class of the polymorphic instances of the class `Person`.

This kind of queries is particularly useful in a system where instances are originated from different sources and specializing a shared ontology. It allows the implementation of several automatic integration strategies [4].

5 The OntoQL toolkit

Around the OntoQL language, we have designed and implemented[1] the tools that we describe in this section.

OntoQL engine. The OntoQL engine implements the algebra, called *OntoAlgebra* [11], defining the semantics of the OntoQL language and the resulting optimizations on an OntoDB prototype.

OntoQL*Plus. OntoQL*Plus is a command line OntoQL language interface. It provides a syntax highlighting and a history of the executed commands.

[1] demonstrations are available at:
http://www.plib.ensma.fr/plib/demos/ontodb/index.html

OntoQBE. OntoQBE is a graphical OntoQL language interface. Figure 4 shows an interactively constructed OntoQL query. This interface allows to express object-oriented constructors of OntoQL such as path expressions (`TAG1`) or polymorphism (`TAG2`). Moreover, this interface presents the ontological definitions (illustration, code, version ...) of the concepts involved in the query (`TAG3`).

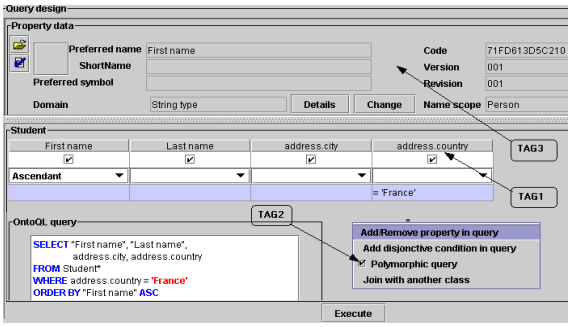

Figure 4. OntoQBE: A graphical OntoQL language interface.

OntoAPI. OntoAPI is a JAVA representation of the ontology model presented in section 3.1. This API (Application Programming Interface) allows to load classes and properties of an ontology from an OBDB without knowing the OntoQL language.

JOBDBC. JOBDBC API provides an access to an OBDB from the JAVA programming language. It extends the JDBC API providing methods to retrieve instances of OntoAPI interfaces as the result of a query and giving ontological definitions as metadata of a query.

6 Comparison with Related Work

OntoQL has been defined as an extension of SQL to exploit an OBDB model defined for semantic integration. With respect to this origin, related languages are multidatabase languages like SchemaSQL [13] or MSQL [15]. OntoQL shares with these languages the capability to express queries on data independently of their schemas. However, whereas these languages use the system catalog as an abstraction from database schemas, OntoQL uses ontologies. Consequently, OntoQL presents many differences with these languages such as its object-oriented nature or its independency of the model (relational, object-relational, object) used to represent the schemas of the data.

Query languages for the semantic web like RQL [12] or OWL-QL [8] have been defined on OBDB models presented in section 2. Like OntoQL, these languages offer the possibility to query ontologies, instances and both ontology and instances. However, in these approaches, an instance is an URI independently of its values of properties. It can belong to different classes not related by the subsumption

relationship. Consequently, OntoQL presents the following differences with these languages.

- *Object orientation*. The result of an OntoQL query searching instances of a class/entity is an object (internal identifier and state) and not an URI (external identifier).
- *Manipulation of schema*. OntoQL allows to create, alter and drop the schema of the data and offers the possibility to query data from these schemas (SQL upward compatibility).
- *Type checking*. OntoQL checks whether properties/attributes used in the **`SELECT`** or **`WHERE`** clauses are defined on some classes/entities in the current scope.
- *Usual syntax*. OntoQL provides usual constructors of traditional languages. For example, **`SELECT *`** can be used to retrieve the state of an instance.

Moreover, OntoQL presents the following features not yet provided by semantic web query languages [9] :

- *Exploitation of multi-lingual definitions*. An OntoQL query on content may be expressed in different natural languages. Moreover, attributes may have values for different natural languages.
- *Useful operators*. OntoQL provides grouping (**`GROUP BY`**), sorting (**`ORDER BY`**) and collection manipulation operators.

7 Conclusion

The contribution of this paper is double. On the one hand, we have discussed and shown the differences existing between classical database models and ontology based database models. The need of new tools to manage the latter was a result of this study. On the other hand, we proposed a database language for managing the ontology based databases.

This paper showed a new language allowing the exploitation of ontology based databases. The need of such a language was motivated by the fact that both ontologies and their contents are stored in a database. This language is fully implemented on an OBDB prototype on top of PostgreSQL and actually runs on several applications. A QBE like interface is also available and a demo of the usage of this language can be found at:
`http://www.plib.ensma.fr/plib/demos/ontodb/`.

For the future it is planned to study the link existing between database based approaches for ontologies and the logic based approaches for ontologies. Our claim is that it is possible to offer an efficient storage capability for the instances described in the logic based approaches for ontologies like in OWL. As a benefit, we would be able to provide a database management to the instance together with a reasoning engine.

References

[1] R. Agrawal, A. Somani, and Y. Xu. Storage and querying of e-commerce data. In *VLDB*, pages 149–158. Morgan Kaufmann Publishers Inc., 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] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, and P. F. P.-S. ad Lynn Andrea Stein. *OWL Web Ontology Language Reference*. World Wide Web Consortium, Feb. 2004.

[4] L. Bellatreche, G. Pierra, D. N. Xuan, H. Dehainsala, and Y. Aït-Ameur. An a priori approach for automatic integration of heterogeneous and autonomous databases. In *DEXA*, pages 475–485, 2004.

[5] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *SemWeb*, pages 54–68, 2002.

[6] R. G. G. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1993.

[7] A. Eisenberg and J. Melton. Sql: 1999, formerly known as sql 3. *SIGMOD Record*, 28(1):131–138, 1999.

[8] R. Fikes, P. J. Hayes, and I. Horrocks. Owl-ql - a language for deductive query answering on the semantic web. *Journal of Web Semantics*, 2(1):19–29, 2004.

[9] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of rdf query languages. In *SemWeb*, November 2004.

[10] S. Harris and N. Gibbins. 3store: Efficient bulk rdf storage. In *PSSS*, 2003.

[11] S. Jean, Y. Aït-Ameur, and G. Pierra. An object-oriented based algebra for ontologies and their instances. Technical report, LISI/ENSMA, january 2006.

[12] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. Rql: a declarative query language for rdf. In *WWW*, pages 592–603, 2002.

[13] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. Schemasql - a language for interoperability in relational multi-database systems. In *VLDB*, pages 239–250, 1996.

[14] O. Lassila and R. Swick. Resource description framework (rdf) model and syntax specification, 1999.

[15] W. Litwin, A. Abdellatif, A. Zeroual, B. Nicolas, and P. Vigier. Msql: A multidatabase language. *Inf. Sci.*, 49(1-3):59–101, 1989.

[16] Z. Pan and J. Heflin. Dldb: Extending relational databases to support semantic web queries. In *PSSS*, 2003.

[17] G. Pierra. Context-explication in conceptual ontologies: Plib ontologies and their use for industrial data. *Journal of Advanced Manufacturing Systems*, 2004.

[18] G. Pierra, H. Dehainsala, Y. Aït-Ameur, and L. Bellatreche. Base de données à base ontologique : principes et mise en œuvre. *Ingénierie des Systèmes d'Information*, 10(2):91–115, 2005.

[19] H. Wache, T. Vgele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information — a survey of existing approaches. In *IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.