OntoQL: an exploitation language for OBDBs

Stéphane Jean Guy Pierra Yamine Ait-Ameur

LISI/ENSMA, Téléport2 - 1, Avenue Clément Ader, Futuroscope, France
{jean,pierra,yamine}@ensma.fr

Abstract

A database containing an ontology which defines the semantics of the data structured by a traditional logical model is called an Ontology Based DataBase(OBDB). Data models and query languages designed for traditional databases are inappropriate for OBDBs. This paper proposes a new exploitation language for OBDBs called *OntoQL*. It allows to query the data at semantic level using the expressive power provided by the ontology model. Moreover, both data and ontology can be queried in parallel. This is particulary useful for users who do not have an exact knowledge of the used ontologies and/or logical database models. Finally, it opens interesting issues on query optimization.

1 OBDB another database model

The design of a database is founded on a sequence of steps. A conceptual model is designed from the user requirements. This model *prescribes* the information that shall be represented in the target database. Then, it is transformed into a logical model which provides data structures that implement the concepts of the conceptual model. As a consequence: 1) exchanging of data between different databases issued from different conceptual models of a common application domain is difficult since they present both structural and semantic heterogeneities 2) a large amount of semantics of the data is lost during the transformation from conceptual to logical model.

These two points have been addressed quite separately in the database field by proposing integration approaches or new database models aiming at bridging the gap between the conceptual and the logical models. Ontologies, defined by Gruber [GRU 93] as "an explicit specification of a conceptualization" play a major role in implementing these solutions. An ontology allows to *describe*, in a consensual way, the whole relevant information of a rather broad application domain. When such an ontology exists, the process of database design no longer needs to create new conceptualization, but it just needs to extract and/or import from the domain ontology, relevant information pieces required for application design. The Ontology Based Database (OBDB) model, developed in our laboratory [PIE 05], allows to represent both the ontology of an application domain, the encoded conceptual model and the logical database model together with its instances stored in a database. Each piece of data is associated to an ontological concept which defines its meaning unambiguously. Thus, this model avoids semantics loss occurring during the transformation from conceptual to logical model. Oppositely to other approaches like ontology-based search or web service discovery, where automated reasoning provided by the ontology model is a key functionality, we focus on performance and scalability of OBDB and on the processings enabled by the OBDB model (integration, data exchange, ...) Currently, the OBDB model is formally specified and an implementation is available. My PhD work is performed in the context of the development of the OBDB model. As a PhD student, I was assigned the task of specifying, designing and implementing an exploitation language, named *OntoQL*, allowing to manage OBDB databases. This paper presents the different steps we followed to design the *OntoQL* language. First, we present a set of requirements we have identified for this language. Then, section 3 briefly studies how three categories of languages fulfill (fully/partly) these requirements. Section 4 and 5 present specification and implementation issues of the *OntoQL* language for the OBDB model. Finally, section 6 describes our outgoing work.

2 Requirements for an exploitation language of an OBDB

Before presenting the requirements we identified for *OntoQL*, it's necessary to precise OBDB assumptions. Each instance of the logical/conceptual model is associated to a single description (one base class) in the ontology. On the one hand, since a data description of the conceptual/logical model may import only a subset of the corresponding concept description available in the conceptual ontology, an instance may respect only a subset of definitions of its base class: *"strong typing"* assumption linking ontology and conceptual mod-

els. On the other hand, like in traditional databases all the concepts available in the conceptual model are represented in the logical model and thus an instance must respect all the definitions of its associated description in the conceptual model: *"exact typing"* assumption. Currently, we consider the PLIB ontology model [ISO 98] and we have defined a restriction of OWL [BEC 04] to use it as a *"strong typing"* based ontology model. However, considering other ontology models is currently being studied.

Under these assumptions, as the first proposal of our PhD work, the following requirements for an exploitation language of an OBDB were established.

Req1. Ontology manipulation. It requires to be able to insert, update or delete classes, properties, definitions, comments, ... of the ontology.

Req2. Creation of conceptual models from an ontology. One of the main characteristics of the OBDB model is its capability to operationally support the conceptual model of the described database. The elements of the conceptual model are built from the concepts available in the ontology. Importation of part or whole of the concepts described in the ontology allows to build such a conceptual model. Operators implementing such an importation mechanism shall be provided by the exploitation language.

Req3. Manipulation of contents. Instances of a conceptual model define the content or extent of a database. They shall be inserted, deleted and updated in the OBDB model.

Req4. Querying capabilities. Querying shall be allowed at different levels.

4.1- Query on content: retrieve information content of known conceptual models. It corresponds to the traditional query capabilities provided by classical database languages like SQL[COD 70].

4.2- Query on ontology: retrieve ontology characteristics. It may be used either to retrieve definitions (names, translations, labels, URIs, ...) of a given concept or to retrieve part or whole of the ontology which may be exported, shared and mapped onto other ontologies. The first family of operators manipulates ontology concepts in order to discover their meanings while the second family of operators manipulates ontologies themselves.

4.3- Query on ontology and on content: gather the capabilities of both the two previous levels. It allows to query both ontology concepts and contents of conceptual models. The logical model of the conceptual model is hidden since the queries are addressed directly to the ontology. It becomes possible to access both concept meanings and contents and to extract part of an ontology together with the related instances.

Req 5. Expression of mapping operators. Among the operators that manipulate ontologies (4.2), we may find mapping operators which shall allow to map an ontology onto a set of conceptual models and/or map an ontology onto a set of ontologies.

Req6. Use of the proposed language. The suggested language shall be usable interactively on any OBDB. In addition, the proposed language shall offer an application programming interface (API) allowing to manipulate and to query either concepts of the ontology or instances in the OBDB.

Req7. Formal consistent semantics. The proposed language shall be formally specified and satisfy relevant semantic properties (e.g. completeness).

3 Limits of existing solutions

When the previous requirements have been set up, we have checked, as second and bibliographic step, whether existing database and/or ontology languages fulfill these requirements.

3.1 Traditional database languages: SQL99 and OQL

SQL99[EIS 99] and OQL[CAT 93] support the definition of classes and properties but don't allow to define ontological descriptions (synonymous names, comments, illustrations, ...) of a class and of the related properties. An extent may be associated to these classes as a relation in SQL99 or as a name for OQL. However, these extents are defined using all the properties defined on the class. They respect the *exact typing* assumption. Once the extent has been defined, instances may be managed. SQL99 and OQL provide the insert, update and delete operators.

Since the ontology models we considered are described in an object oriented approach, the operators provided by SQL99 or OQL are well suited to retrieve part or whole of the ontology. However, only OQL allows to express a query directly on a given object, thus enabling to retrieve definitions of a given ontology concept. Moreover, if the conceptual model extracted from the ontology is object oriented based, SQL99 or OQL can be used to query information of this conceptual model. But, neither OQL, nor SQL99 provide specialized operators to retrieve both object value and object semantics. They don't provide operators for manipulating models and mapping too.

From an operational point of view, SQL99 and OQL may be both used interactively or embedded in a programming language. Finally, the specification of SQL99 and OQL define a formal consistent semantics.

3.2 Ontology languages: OWL-QL

The OWL Query Language (OWL-QL) [FIK 04] is a candidate standard language for querying the semantic Web represented using OWL. OWL-QL is a deductive language based on the logical resolution of first order queries. The model where the OWL predicates are interpreted is a set of RDF triples. This simple model allows to combine data and ontology con-

cepts in a query expression. However, the expressive power of OWL-QL is limited compared to classical database languages. Indeed, it doesn't support complete Boolean filters (negation, disjunction), set-based operations (union, intersection, difference), arithmetic operations on data values, non polymorphic query, ...

3.3 OBDB available languages: CQL

CQL [MIZ 02] is an exploitation language for databases based on the PLIB ontology for components library management systems. Even if this language is not fully compatible with existing standards, it is equipped with all the features of a database language. However, it is very close to the PLIB ontology model and to relational database systems. The user of CQL needs to have a deep knowledge of the PLIB ontology and of relational database concepts. It contains ad-hoc operators with no formalized semantics. Moreover, the definition does not take into account the *strong typing* assumption. Finally, CQL makes a clear distinction between queries on content and queries on the ontology. A different data model and different query operators are proposed by CQL for each part. Indeed, CQL doesn't provide set-based operations (union, intersection, difference) nor polymorphic query or functions on aggregate on the ontology part. Consequently, CQL doesn't allow to query data and ontology simultaneously.

3.4 Summary

Table 1 summarizes the previous comparison. It appears that none of the discussed approaches covers the requirements we stated in section 2. Therefore, next section outlines the specification of the proposed *OntoQL* language under development on top of OBDB. This proposition is the kernel of our PhD work currently under development either from the theoretical (addressed in section 4) and operational (addressed in section 5) points of views.

	SQL99	OQL	OWL-QL	CQL
1	no	no	yes	yes
2	no	no	yes	yes
3	yes	yes	yes	yes
4.1	yes	yes	partially	yes
4.2	partially	yes	partially	yes
4.3	no	no	partially	no
5	no	no	no	no
6	yes	yes	yes	yes
7	yes	yes	yes	no

Table 1: Fulfilled requirements by existing languages

4 Our current specification

This section presents some of the features available in the *OntoQL* proposal.

4.1 Ontology Data Definition Language (Req1)

All common characteristics of PLIB and OWL, more some crucial aspects (e.g. multilingual), represent the kernel of our ontology model. These core concepts are "hard-encoded" in *OntoQL* through a set of keywords in the language. For instance, the name, definition and the four properties of a class can be declared as follows:

```
CREATE CLASS CVehicle (
  DESCRIPTOR (
    #name[fr]='Véhicule',
    #definition[en]='a conveyance')
  PROPERTIES (
    max_speed REAL, color STRING,
    price REAL_CURRENCY_TYPE,
    owner CPerson inverse its_vehicle SET[0:n]));
```

Since the definition of a property in an ontology model is not restricted to a domain specification, a CREATE PROPERTY command is also proposed. To support full PLIB and full restriction of OWL, a modular approach is adopted. Indeed, parameters values on *OntoQL* allowing to work in full PLIB or full OWL can be set.

4.2 Content Data Definition Language (Req2)

To create new instances, one must first define an extension to the ontology class. This definition requires to choose the set of valued properties from an ontology class in the target application. For example, in the following definition of the extent of `CVehicle`, two properties of `CVehicle` among the four available properties are used.

```
CREATE EXTENT OF CVehicle (max_speed, color);
```

The representation of the logical model depends on the used database model defined by a parameter value of *OntoQL*. The extracted logical model can also be customized by explicitly specifying the data type used in the database for a given extent of a class or a property instead of using the default representation. Moreover, it is possible to specify the way of representing non atomic properties.

4.3 Query Language (Req4)

Query on ontology and on content may be used to retrieve property values of an instance together with the semantic definitions of the properties of these instances. For example, the following query retrieves the `price` value (content) of all the kinds of vehicles (* operator) and the `currency` (described in the ontology) in which this value is defined:

```
SELECT c.price, c.price.#currency
FROM CVehicle* as c;
```

Notice that if no extent of `CVehicle` is available, no value of `price` is returned but the information on `currency` is provided.

Querying both ontology and content are particularly useful for a user without exact knowledge of the ontology. For example, the previous query can be expressed without knowledge about the identifier of the property `price`:

```
SELECT c.$P, c.$P.#currency
 FROM CVehicle* AS c
 USING $P = SELECT p FROM #PROPERTY p
            WHERE p.#range=REAL_CURRENCY_TYPE
              AND p.#name[en]='price';
```

$P is used as a wildcard. Its value is defined in the specific USING clause.

4.4 Mapping operators (Req5)

The *USING* clause of *OntoQL* can also be used to express mappings between concepts of different ontologies. In the following example, the class `CVehiculeNissan` is defined as a *case_of* the class `CVehiculeRenault`. The *case_of* operator is a PLIB operator defining a subsumption relationship between two classes. Compared to the *is_a* operator, it allows to inport only a subset of the properties of an upper class (partial inheritance). Thus, the query expressed on the `CVehiculeRenault` will retrieve all the `CVehiculeNissan` instances as well.

```
SELECT c.$P, c.$P.#currency
  FROM CVehiculeRenault* AS c
 USING CVehiculeNissan CASE_OF CVehiculeRenault,
  $P = SELECT p FROM #PROPERTY p
       WHERE (p.#domain=CVehiculeRenault
          OR  p.#domain=CVehiculeNissan)
          AND p.#range=REAL_CURRENCY_TYPE
          AND p.#name[en]='price';
```

Notice that this query may retrieve the `price` of the vehicles in different currencies. Indeed, the property whose english name is `price` may be defined differently in the ontologies defining the `CVehiculeRenault` (e.g. Euro currency) and `CVehiculeNissan` (e.g. Yen currency) classes.

4.5 API (Req6)

Our API is encoded in the JAVA language. First, this interface offers functionalities similar to JDBC, i.e it allows to send an *OntoQL* query. The result is processed in a `Resultset` object. Second, this interface offers the JAVA representation of the used ontology model. A `load` method is defined in each of the classes corresponding to the ontology model. It allows to load the main attributes of an object whose identifier is known. The object is then loaded on demand, i.e an *OntoQL* query is called whenever a non loaded attribute is requested. Third, the defined interface allows to get the result of an *OntoQL* query as an extension of the `Resultset` class (`OntoQLResultset`) containing objects of the java representation of the ontology model.

4.6 Formal consistent semantics (Req7)

An algebra representing the concepts of the OBDB data model and the operators allowing to manipulate them are under development. Equations are associated to each operator to define their formal semantics.

5 Our current implementation

An implementation prototype of *OntoQL* and its API actually run. This implementation is based on the OntoDB operational prototype of OBDB [PIE 05]. This implementation uses the PostgreSQL relational-object database system.

5.1 Implementation choices

When implementing the OBDB model in a relational-object system, two techniques to process *OntoQL* queries could have been applied 1) design a new query module (query optimizer and query processor) 2) use the semantics of the OBDB model to translate *OntoQL* queries into relational-object queries on the implementation of this model in the relational-object system.

At this early stage of our work, the second approach has been implemented for rapid prototyping reasons. We obtained some feedback on our proposed query language which help for evolving the specification. In order to make our proposal independent of any OBDB implementation, it must translate an *OntoQL* query into a sequence of API operators calls. Such an API has been implemented and it provides access to the data contained in the OBDB (Figure 1: Internal API). Next section presents our proposal to visually edit an *OntoQL* query and section 5.3 describes how such a query is processed.

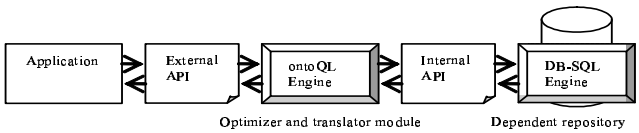

Figure 1: Implementation architecture of OntoQL

5.2 OntoQL visual query language

Defining a query on both ontology and extent may be rather complex. We have decided to implement in parallel both textual (presented before) and a QBE-like [ZLO 77] graphical language. Our current QBE-like implementation is written as a module of the PLibEditor software, developed for managing PLIB ontologies. Currently, our implementation allows the users to

query data from the ontology using polymorphic query and path expression (part of Req4.3). Figure 2 shows how such a query can be interactively constructed. The query construction begins by the choice of a class in the ontology hierarchy (tag1). Then, the properties used for restriction and selection in the query are chosen among those that are in the class domain through a dialog box accessible by a contextual menu (tag2). It creates automatically a table (tag3) with columns corresponding to the chosen properties. This table is the template of the query result. Then, the selection criteria are defined in a disjunctive form using a new line of the table for each element of the disjunctive condition. Finally, the user may choose if the query is polymorphic through a contextual menu. The panel above the template table (tag4) is used for the ontology description of a chosen property in the query. It will be used latter as a template for querying the ontology level (Req4.2).

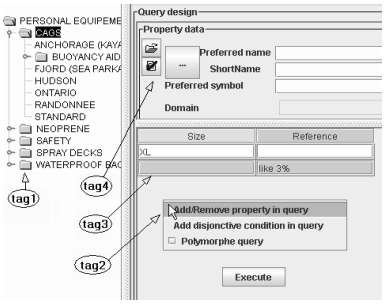

Figure 2: QBE for OntoQL

5.3 OntoQL query processing

Let us describe the processing of the following query inside the *OntoQL* engine of figure 1.

```
SELECT max_speed, color
FROM CVehicle* WHERE price < 10000;
```

Once parsed, this query is processed by an optimizer module. The *strong typing* semantics specifies that a property of an ontology class may not be implemented in the related logical model. Thus, whenever a selection operator involves a non used property of an ontology class, it is not necessary to process the query: the result is the empty set. This capability can be generalized to path expression properties: whenever a path expression implies a non used property it's unnecessary to compute its value. Using this optimization technique, the previous query is only processed on the subclasses of the `CVehicle` class where the property `price` is used and valued. Supposing that it is the case for the classes `CCar` and `CTruck`, the optimizer module transforms our query into the following one:

```
(SELECT max_speed, color
 FROM CCar* WHERE price < 10000) UNION
(SELECT max_speed, color
 FROM CTruck* WHERE price < 10000)
```

Once optimized, the query is passed to the translator module (figure 1) which translates through the internal API our query into a repository dependent one. In our implementation, the previous *OntoQL* query will be translated into the following SQL query.

```
(SELECT max_speed, color
 FROM TCar WHERE price < 10000) UNION
(SELECT max_speed, color
 FROM TSportCar WHERE price < 10000) UNION
(SELECT max_speed, color
 FROM TTruck WHERE price < 10000)
```

where `TCar` (resp. `TSportCar`, `TTruck`) are the internal names of the extent of `CCar` (resp. `CSportCar`, `CTruck`).

6 Future work

Our work will pursue the investigation of both specification and implementation of *OntoQL*. At the specification level, it is planned to establish the formal properties required for *OntoQL* (e.g. completeness of the queries and information extraction). At the implementation level, two directions will be followed. First, pursue the implementation of an optimized and efficient query module on top of OBDB. Second, complete the QBE-like language by offering the possibility to query both the ontology and the content level.

References

[BEC 04] Bechhofer S., van Harmelen F., Hendler J., Horrocks I., McGuinness D. L., ad Lynn Andrea Stein P. F. P.-S., "OWL Web Ontology Language Reference", World Wide Web Consortium, Feb. 2004.

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

[COD 70] Codd E. F., "A Relational Model of Data for Large Shared Data Banks.", *Commun. ACM*, vol. 13, num. 6, 1970, p. 377-387.

[EIS 99] Eisenberg A., Melton J., "SQL: 1999, formerly known as SQL 3", *SIGMOD Record*, vol. 28, num. 1, 1999, p. 131-138.

[FIK 04] Fikes R., Hayes P. J., Horrocks I., "OWL-QL - a language for deductive query answering on the Semantic Web.", *J. Web Sem.*, vol. 2, num. 1, 2004, p. 19-29.

[GRU 93] Gruber T. R., "A translation approach to portable ontology specifications", *Knowl. Acquis.*, vol. 5, num. 2, 1993, p. 199–220, Academic Press Ltd.

[ISO 98] ISO13584-42, "Industrial Automation Systems and Integration Parts Library Part 42 : Description methodology : Methodology for Structuring Parts families", report , 1998, International Standards Organization.

[MIZ 02] Mizoguchi-Shimogori Y., Murayama H., Minamino N., "Class Query Language and its application to ISO13584 Parts Library Standard", *In 9th European Concurrent Engineering Conference, ECEC 2002*, Modena, Italy, 2002, p. 128-135.

[PIE 05] Pierra G., Dehainsala H., Ait-Ameur Y., Bellatreche L., "Base de Données à Base Ontologique : principes et mise en œuvre.", *To appear in Ingénierie des Systèmes d'Information (ISI)*, 2005.

[ZLO 77] Zloof M. M., "Query-by-Example: A Data Base Language.", *IBM Systems Journal*, vol. 16, num. 4, 1977, p. 324-343.