Ontology Query Languages for Ontology-Based Databases: a Survey

Stéphane Jean
LISI/ENSMA and University of Poitiers, France
Yamine Aït Ameur
LISI/ENSMA and University of Poitiers, France
Guy Pierra
LISI/ENSMA and University of Poitiers, France

ABSTRACT

Current databases and their associated languages allow a user to exploit data according to their logical model. Usually, there is a gap between this logical model and the actual concepts represented. As a consequence, exploiting, exchanging and integrating data stored in databases are difficult. To overcome these problems, several approaches have proposed to extend current databases with ontologies. We called *Ontology-Based Databases (OBDB)* such databases. However, current database languages such as SQL have not been designed to exploit ontologies. Thus, a new generation of languages we called *ontology query languages* has emerged. The goal of this chapter is to provide an up to date survey on ontology query languages. We survey languages coming from the Semantic Web community as well as those coming from the database community.

Keywords: Ontology, Ontology-Based Database (OBDB), Ontology Query Language

INTRODUCTION

Data warehouses are designed to aggregate data and allow decision makers to obtain accurate, complete and up to date information. In current data warehouses, queries are issued to the logical model of data, making direct use of the table and column information that describes the persistence structure. Usually, there is a gap between this logical model and the actual business concepts used by decision makers. As a consequence, users of data warehouses have to use existing documentation (e.g, data dictionaries) - if any and if not out of date – to discover the meaning of tables and columns. This makes using and querying a data warehouse problematic for all users that have not designed it.

Defined by Gruber as "an explicit specification of a conceptualization", ontologies have been proposed to explicit the semantics of data. They allow to describe, in a consensual way, the relevant concepts of a given application domain. Thus, the idea to describe a data warehouse with ontologies in order to make explicit the semantics of data stored has emerged (Xuan et al., 2006). This idea is concretized by introducing an ontological layer in a data warehouse. This layer can be used by decision makers to express queries using business concepts they are used to manipulate. However, existing query languages such as SQL or OQL have not been designed to exploit ontologies. For this purpose, a new generation of query languages, called *ontology query languages*, has emerged.

The goal of this chapter is to provide an up to date survey on the capabilities of existing ontology query languages to manage databases extended with ontologies we call *ontology-based databases (OBDBs)*. Previous surveys on ontology query languages (Bailey et al. 2005; Haase et al. 2004) have focused on the capabilities of Semantic Web query languages to manage RDF-Schema ontologies and data whatever be the used storage system. In this chapter, we propose to complete these previous surveys by taking a database-oriented point of view. Thus, this chapter discusses two representatives of Semantic Web query languages, namely SPARQL (Prud'hommeaux and Seaborne, 2008) and RQL (Karvounarakis et al., 2004) as well as two representatives of database-oriented ontology query languages namely, Oracle extension to SQL (Chong et al., 2005; Wu et al., 2008) and OntoQL (Jean, 2007; Jean et al., 2006) .

This chapter is organized as follows. In section 2, we present our point of view on domain ontologies and their classification. We conclude this section by describing our proposed extension of the traditional ANSI/SPARC database architecture with ontologies. Requirements for an exploitation language of this proposed architecture are then defined in section 3. These requirements are used to compare some existing Semantic Web Query Languages and database-oriented ontology query languages in section 4 and 5. Finally, section 6 concludes this comparison and introduces future work.

DOMAIN ONTOLOGIES AND THEIR APPLICATION TO DATABASE

In this section, we present the database-oriented point of view on ontologies we take (Jean et al., 2007b; Fankam et al., 2008) to compare existing ontology query languages.

Definition and Classification of Domain Ontologies

Several definitions have been proposed for an ontology (Gruber, 1993; Guarino, 1998). In our work a domain ontology is "a formal and consensual dictionary of categories and properties of entities of a domain and the relationships that hold among them". This definition emphasizes three criteria that distinguish ontologies from other models used in Computer Science. An ontology is:

1. *formal* : it is based on a logical axioms and may be processed by computers; so checking consistency and performing automatic reasoning are made possible;
2. *consensual* in a community, i.e. several members have agreed upon the concepts represented in the ontology;
3. has the *capability to be referenced.* A universally unique identifier can be used to define the semantic of a piece of data, whatever are the modeling schema of the ontology and the data model.

All ontologies are not similar. We distinguish the three following categories :

- *Conceptual Canonical Ontologies (CCOs)* provide concepts definitions without redundancy. In CCOs (for example, (IEC61360-4, 1999)), information is represented in terms of classes and properties that have unique identifiers.
- *Non Conceptual Canonical Ontologies (NCCOs)* contain not only primitives concepts (canonical) but also defined concepts, i.e. those concepts for which the ontology provides

a complete axiomatic definition by means of necessary and sufficient conditions expressed in terms of other - primitive or defined - concepts.

- *Linguistic Ontologies (LOs)* define terms appearing in the universe of discourse of a given domain. Relationships between terms are captured in a semi-formal way by linguistic relationships (e.g, synonymy or hyponymy). An example of LO is Wordnet.

These three categories of ontologies can be combined into a layered model, called the Onion Model, shown in Figure 1. At the heart of this model is a CCO. It provides with a formal basis to model and to exchange efficiently the knowledge of a domain. From primitive concepts of the CCO, a NCCO can be designed. This NCCO provides constructors to relate different conceptualizations made on this domain. Finally, a LO may provide a natural language representation of NCCO or CCO concepts, possibly in the various natural languages where these concepts are meaningful.

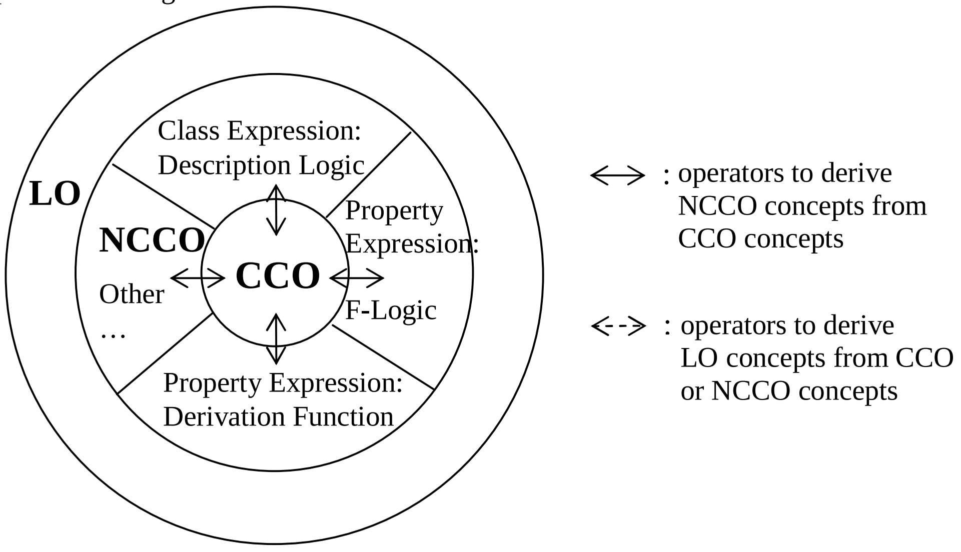
LO
NCCO
CCO
Class Expression: Description Logic
Property Expression: F-Logic
Property Expression: Derivation Function
Other …
<-> : operators to derive NCCO concepts from CCO concepts
<--> : operators to derive LO concepts from CCO or NCCO concepts

Figure 1. The onion model of domain Ontology

When they are designed according to the onion model, ontologies have always a canonical layer, they may have a non canonical layer and they always have a minimum of linguistic aspects and in particular, terms that denote the represented concept. In this chapter, we focus on the exploitation of such ontologies in databases.

Extension of the ANSI/SPARC Architecture with Ontologies

The major objectives of a database are to ensure an efficient management of data and to provide access to data independently of their physical representation. The ANSI/SPARC architecture has been proposed to fulfil these objectives. It distinguishes two main access levels:

- the *physical level* which defines how data are stored and managed using a file management system;
- the *logical level* which defines how data are structured using the database data model (e.g., the relational or object model).

When designing a database according to this architecture, a large amount of data semantics may be lost during the transformation from the conceptual model (CM) to a logical model.

Moreover, the meaning of the CM is not formally documented, and thus it cannot be stored in the database. To solve these difficulties, references to an ontology appear as a relevant solution. Thus, we propose to extend this architecture with the ontological level. This level defines explicitly data semantics. Moreover, the CM will be modelled by referring to the ontology concept to which they correspond. This extended architecture is shown in Figure 2.

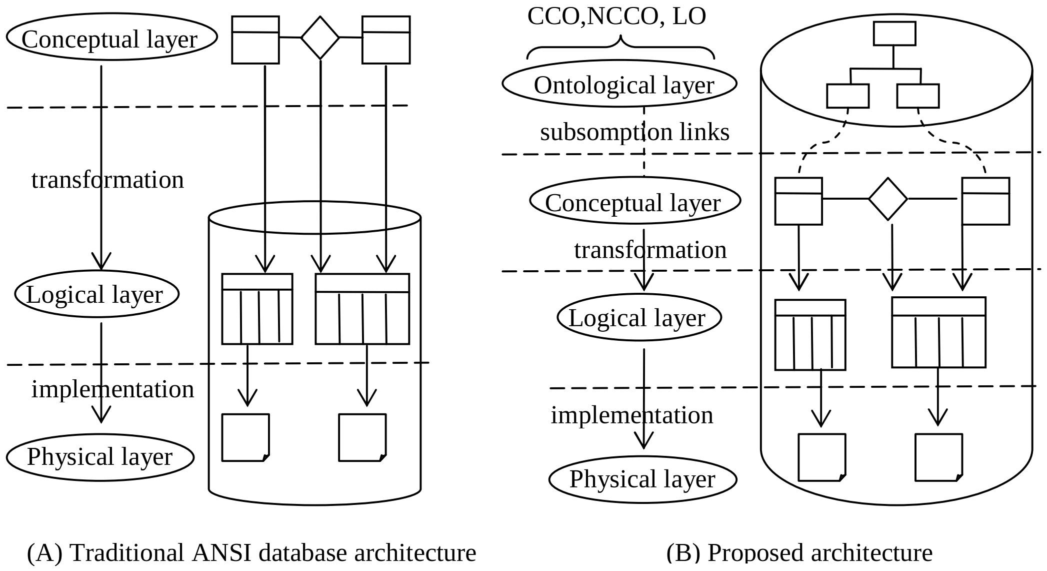

Figure 2. Proposed extension of ANSI/SPARC Architecture

Figure 2(A) presents the traditional database architecture. A conceptual model is represented in a modeling language like Entity-Relationship. Then, it is often used to generate automatically the logical model of data. This logical model is represented at the physical level by a set of files. In Figure 2(B) we propose to extend this architecture with the following elements.

- *Ontological level.* It is composed of one (or several if the scope of the system encompasses the domain of several existing ontologies) ontology defining the concepts of the universe of discourse in terms of well-defined classes and properties.
- *Subsumption links.* They link the ontological and conceptual levels, thus defining the set of ontology concepts used to fulfil applications requirements. The meaning of these links, represented in the CM by the absolute identifier of an ontology concept, is that the CM concept is equal or is a special case of the referenced ontology concept (i.e., subsumption).

The goal of this chapter is to study capabilities of existing ontology query languages to exploit the OBDB architecture proposed in this section. This language should (1) be *homogeneous*, i.e. it should provide an access to the different levels of this architecture meaningful for the end-user and (2) it should exploit the specificities of these different levels. We detail these requirements in the next section.

REQUIREMENTS FOR AN EXPLOITATION LANGUAGE OF OBDB

To illustrate our proposed requirements for an exploitation language of the OBDB architecture presented in the previous section, we use the ontology presented as a graph on Figure 3. This example is an extract of the SIOC ontology (*http:// sioc-project.org/*). This ontology describes the domain of on-line communities. A forum (`Forum`) is hosted by a site (`Site`) administered by a moderator (`has_moderator`). Users (`User`) can subscribe to forums (`subscriber_of`) and create messages (`Post`) on these forums (`has_container`). Several answers may be provided for a given message (`has_reply`).

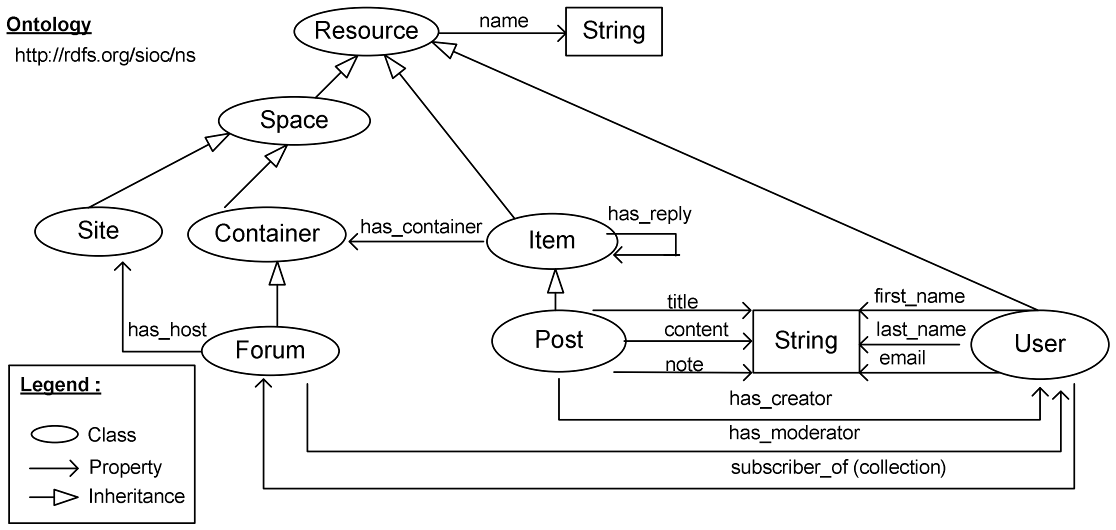

Figure 3. Running example

Requirements Resulting from the Onion Model

One of the most important characteristics of the ANSI/SPARC architecture is to separate the physical and logical representation of data. Defining, manipulating and querying data at the logical level independently of their physical representation are possible. Our architecture adds one more independency, representing both the conceptual and the ontological description of the represented data. Thus, exploitation of data independently of their logical schema is possible.

Requirement 1: (Queries at the ontological level) The language should allow to express queries at the ontological level independently of the logical representation (schema) of data.

Example. Retrieve all (direct and/or indirect) instances of the class `Item`.

Another fundamental characteristic of the ANSI/SPARC architecture is to define an external level. This level defines external schemas (views) that reflect users perception of the application domain (e.g., woman in place of person whose gender is female). These views can be used to define queries. The query engine rewrites them according to the logical model of data. In our architecture, the NCCO layer of an ontology provide views at the onlogical level. It allows each user to represent its own perception of the application domain by defining non canonical concepts expressed in terms of other - canonical or non canonical - concepts. Thus, the language should support the exploitation of such concepts.

Requirement 2: (Definition of non canonical concepts) The language should support the definition of non canonical concepts. Queries may be expressed using these concepts, the query engine interpreting them in terms of canonical concepts.

Example. Create the class `PostDupont` defined as all messages of the user `Dupont`.

The last layer of the Onion Model is composed of the LO part. When an ontology is designed according to the Onion Model, its LO part associates to each concept, one or several terms and textual definitions. These linguistic definitions allow human users to understand the ontology and to reference ontology concepts using their names. These linguistic definitions are often given in different natural languages. To make it easy for members of different countries to use the same ontologies, the language may support the definition and exploitation of multilingual LO.

Requirement 3: (Linguistic exploitation) The language should support the definition and exploitation of linguistic definitions of concepts that may be defined in different natural languages.

Example. Return the first name and last name of users with a query written using English names of concepts and one using French names.

The Onion Model is based on the complementarity of ontology models. Indeed, many ontology models have been proposed such as RDF-Schema (Brickley and Guha, 2004), OWL (Dean and Schreiber, 2004) or PLIB (Pierra, 2008). All these ontology models have constructors to define a CCO composed of classes, properties, datatypes, and instances. In addition to these core constructors, each ontology model provides specific constructors to define an ontology. For example, PLIB provide constructors to define precisely and contextually primitive concepts while OWL provides conceptual equivalence constructors to define a NCCO. All these constructors are useful to define an ontology according to the Onion Model. Thus, the language should be able to integrate constructors coming from different ontology models. This feature would also help to support evolution of existing ontology models (for example, the support of OWL2 (Patel-Schneider and Motik, 2008)).

Requirement 4: (Ontology model extensibility) The ontology model supported by the language should be extendable to support its evolution and to manage ontologies defined with an other ontology model.

Example. Add the OWL constructor `AllValuesFrom` (a class composed of all instances whose values of a given property are all instances of a given class) to the ontology model supported by the language.

Requirements Resulting from Preserving Compatibility with the ANSI/SPARC Architecture

Our proposed architecture extends the ANSI/SPARC architecture, and thus, it also includes the usual logical level. Since many applications have been built using the SQL language to manipulate data at this level, the language should support manipulation of data not only at the ontological level (see previous section) but also at the logical level. Thus, the language will keep upward compatibility for existing applications and will permit to manipulate data at the different levels of our proposed architecture.

Requirement 5: (SQL compatibility) The language should permit the manipulation of data at the logical level preserving SQL compatibility.

.

Example. Assuming that instances of the class `User` are stored in the table `T_User (uri, first_name, last_name, email)`, retrieve all users using a SQL query.

Designing a layered architecture has the drawback to increase the complexity of data processing in upper levels. The introduced complexity has in general consequences on efficiency of query processing. In such architecture, one way to optimize processing at a given level is to use the lower level (e.g., create an index to optimize logical queries). In the proposed architecture, the ontological level has been added on top of the conceptual level. Thus, to optimize query processing at this level, the DBMS shall provide an access to the lower level, i.e. the conceptual level.

Requirement 6: (Access to the conceptual model of data) The language should allow to define, manipulate and query the conceptual model of data from the ontological level.

Example. Create the conceptual model of data corresponding to the class `User` knowing that only the properties `first_name`, `last_name` and `email` will be used in the target application.

Requirements on the Expressive Power of the Language

In traditional databases, the data definition language (DDL) is used to create tables and the data manipulation language (DML) to insert rows with their properties values. The DML has the advantage to be a powerful declarative language being combined with the query language. In the proposed architecture both ontologies and data are stored. A data definition and manipulation languages is required to define and manipulate them.

Requirement 7: (Ontology & data definition and manipulation) The language should offer a definition and manipulation language for ontologies and data.

Example. Create the class `User` of our example and add some instances.

Ontologies are a conceptualization of a domain aiming at covering a wide and diverse range of technical and business requirements. As a consequence, ontologies usually define a lot of concepts. For example, the IEC ontology (IEC61360-4, 1999) on the domain of electronic components contains approximately 100 classes and more than 1000 properties. Moreover, users of an ontology are rarely its designer. Thus, users should have a mean to discover ontologies with the language.

In addition, since an ontology defines generally a lot of concepts, a hierarchy of classes may have many levels. When a query is expressed on a class at a given level, the result may contain instances of this class and of its subclasses. Thus, these instances can be associated to different ontological description that the user may want to retrieve. This example shows the need to be able to query both ontologies and data. Notice that this capability would also be useful to extract a part of an ontology with its instances. In traditional languages, this operation requires the composition of two queries. Thus, this capability would be useful in distributed architecture where the network trips have to be minimized.

Requirement 8: (Queries on ontologies and on ontologies and data) The language should offer querying capabilities on ontologies and both on ontologies and data.

Example. Return all instances of `Resource`. For each instance, retrieve also the classes it belongs to.

In this section, we have defined a requirements specification for an exploitation language of the OBDB architecture defined in section 3. A language fulfilling these requirements would allow to fully exploit this architecture. Using these requirements, we are now able to compare existing ontology query languages.

ANALYSIS OF SEMANTIC WEB ONTOLOGY QUERY LANGUAGES

A lot of ontology query languages have been proposed in the context of the Semantic Web. In a recent survey (Bailey et al., 2005), these languages are classified in seven categories (SPARQL family, RQL family, languages inspired from XPath, XSLT or XQuery, languages in controlled English, languages with reactives rules, deductive languages and other languages). In this section, we present a representative language of the two first categories that fit the most with the requirements we have defined.

SPARQL

The first category of language is named "SPARQL familiy". It is composed of the languages SPARQL (Prud'hommeaux and Seaborne, 2008), RDQL (Seaborne, 2004), SquishQL (Miller et al., 2002) and TriQL (Caroll et al., 2005). These languages consider all information (both ontologies and instances data) as a set of RDF triples `(subject, predicate, object)`. As a representative we have chosen SPARQL which is a W3C Recommendation.

SPARQL is a graph-matching query language. A query consists of a pattern (a set of triples with variables and filters) defined in the `WHERE` clause. This pattern is matched against a data source, and the values obtained from this matching are processed in the `SELECT` clause to give the answer. We describe more precisely this language by discussing its capability w.r.t. the defined requirements.

Requirement 1 (Queries at the ontological level)

Example. The following SPARQL query[1] can be used to rretrieve the instances of the class `Item.`

```
SELECT ?i
WHERE {?i rdf:type sioc:Item}
```

Explanation. The triple in the `FROM` clause introduces the variable `?i` (a variable is prefixed by `?`) to iterate over instances of the class `Item.` This variable is specified in the `SELECT` clause to return its values.

[1] For readability and conciseness, we omit specifications of namespaces and use names instead of URI.

However, the result of the previous query depends on the triples represented in the OBDB. On the one hand, if for a class `C`, a triple `(i, rdf:type, C)` is represented for each direct or indirect instance of `C`, then the previous query returns also all the instances of the class `Post` subclass of `Item`. On the other hand, if a triple `(i, rdf:type, C)` is represented only for each direct instance of `C`, then the query returns only direct instances of `C` unless the query interpreter computes the transitive closure of the subsumption relationship. Thus, the SPARQL fulfill partially this requirement. Ontological queries can be expressed but their results depend on the represented triples or of their interpretation.

Requirement 2 (Definition of non canonical concepts)

SPARQL does not provide a data definition language. Thus, it can not be used to define non canonical concepts. However, if these concepts have been defined in the data source, SPARQL provides a `CONSTRUCT` query form that can be used to compute automatically their instances.

Example. Compute instances of the class `PostDupont`

```
CONSTRUCT ?p rdf:type sioc:PostDupont
WHERE {?p rdf:type sioc:Post . ?p sioc:has_creator “Dupont”}
```

Explanation. The `WHERE` clause retrieves instances (`?p`) of the class `Post` created by the user `Dupont`. The `CONSTRUCT` clause replaces the traditional `SELECT` clause. It is constructed as a `WHERE` clause (a set of triples with variables and filters). The result is a new RDF graph constructed by replacing variables in the `CONSTRUCT` clause by values satisfying the `WHERE` clause. Thus, each post created by `Dupont` is associated to the class `PostDupont` in the resulting graph.

Thus, SPARQL fulfill partially requirement 2. It can not be used to define non canonical concepts but permits to compute their instances (*extension*). Notice that the `CONSTRUCT` query form produces a new RDF graph. As a consequence and contrary to views of databases, extensions of non canonical concepts have to be recomputed whenever extensions of corresponding canonical concepts are modified (in our example, each time a message is added or deleted by `Dupont`).

Requirement 3 (Linguistic exploitation)

SPARQL is equipped with operators to manipulate string defined in different natural languages.

Example. Return the first name and last name of users with a query written using English names of concepts and one using French names.

```
SELECT ?fn ?ln
WHERE {?u rdf:type ?c .
      ?c rdfs:label “User”@en .
      ?u sioc:first_name ?fn .
      ?u sioc:last_name ?ln}
```

```
SELECT ?fn ?ln
WHERE {?u rdf:type ?c .
?c rdfs:label “Utilisateur”@fr .
       ?u sioc:first_name ?fn .
       ?u sioc:last_name ?ln}
```

Explanation. The query written using English (resp. French) names is on the left (resp. right) part. The `WHERE` clause of this query retrieves instances (`?u`) of a class (`?c`) whose name in English (suffix `@en`) is `User`. The query written in French is equivalent to this one. The suffix `@fr` allow to use the French name of the class `User (“Utilisateur”)`.

Thus, SPARQL fulfil requirement 3. It provides also other functions to manipulate multilingual ontologies. For example, the `lang` function return the natural language of a given string.

Requirement 4 (Ontology model extensibility)

SPARQL does not provide a definition language for ontology and thus we can not define the OWL `AllValuesFrom` constructor (sample example). However, since SPARQL has been designed for RDF, it considers an OWL ontology as a set of triples that use the OWL constructors defined in the OWL namespace. Thus, it can be used to query OWL ontologies but the semantics of OWL constructors has to be coded in the query interpreter.

Requirements 5 and 6 (Compatibility with the traditional database architecture)

Even if a SPARQL query has a form similar to an SQL query (clauses `SELECT-FROM-WHERE`), it is adapted to RDF (triples) querying and thus it is different from SQL (requirement 5). In addition, data schema is considered fixed (subject, predicate, object) and thus SPARQL does not provide any operator to modify it (requirement 6).

Requirement 7 (Ontology & data definition and manipulation)

Currently, SPARQL provides only a query language; it is not equipped with a definition and manipulation language (requirement 7). Thus, definition and manipulation of data are considered external functionalities provided by the data source. The need of a standard mechanism to update data sources has been addressed in (Seaborne and Manjunath, 2008) with the proposition of SPARQL/Update. But, this language is not yet integrated in the recommendation of the W3C.

Requirement 8 (Queries on ontologies and on ontologies and data)

Since, SPARQL considers all information as RDF data, it can be easily used to combined ontology and data querying.

Example. Return all instances of `Resource` with the classes it belongs to.

```
SELECT ?i ?c
 WHERE {?i rdf:type sioc:Resource . ?i rdf:type ?c}
```

Explanation. The first triple of the `WHERE` clause retrieves instances (`?i`) of the class `Resource`. The second triples retrieves classes (`?c`) these instances belong to.

Notice that in this example, like in all SPARQL queries, results depend on the triples represented in the data source and/or the interpretation of these triples by the query interpreter.

RQL

The second category of Semantic Web languages is called "RQL family". It is mainly composed of the languages RQL (Karvounarakis et al., 2004), SeRQL (Broeskstra and Kampman, 2003) and eRQL (Tolle and Wleklinski, 2004). We have chosen to discuss the RQL language which is the most complete.

RQL has been designed following a functional approach similar to the object-oriented language OQL (Cattell, 1993). Thus, simple queries consist of function calls (e.g, `SubClassOf(Resource)` to retrieve all subclasses of `Resource`). More elaborate queries can be defined using a traditional `SELECT-FROM-WHERE` syntax. The `FROM` clause introduces path expressions (with variables) built from a set of predefined basic path expressions and operators (e.g., `C{X}` is a basic path expression that introduces a variable `X` on all instances of the class `C`). The `WHERE` clause is used to define conditions on variables introduced in the `FROM` clause. Finally, the `SELECT` clause defines the variable projected in the result (like in SPARQL).

RQL is also composed of a manipulation language named RUL (Magiridou et al., 2005) and a view language named RVL (Magkanaraki et al., 2004). We present these different languages by discussing capabilities of RQL to fulfil the proposed requirements.

<u>Requirement 1 (Queries at the ontological level)</u>

The data model of RQL is based on RDF-Schema. To distinguish clearly the data, ontology and ontology model levels, the data model of RQL has some restrictions compared to RDF-Schema (e.g., a class can not be subsumed by an ontology model constructor). Yet, this model contains the main constructors of RDF-Schema (class, property and subsumption) and thus, queries at the ontological level are possible.

Example. Retrieve all instances of the class `Item`

```
SELECT I
  FROM sioc:Item{I}
```

Explanation. The `FROM` clause introduces the variable `I` on all (direct and indirect) instances of the class `Item`. The `SELECT` clause projects URI of these instances. To retrieve only direct instances, the class `Item` must be prefixed with `^` (i.e., `^Item`)

<u>Requirement 2 (Definition of non canonical concepts)</u>

The view language associated to RQL (RVL) can be used to represent non canonical concepts such as OWL restrictions.

Example. Create the class `PostDupont` defined as all messages of the user `Dupont`.

```
CREATE NAMESPACE myview=&http://www.lisi.ensma.fr/ex-view.rdf#
  VIEW rdfs:Class("PostDupont")
       Property(P, PostDupont, range(P))
       FROM Property{P}
       WHERE domain(P) >= sioc:Post
  VIEW PostDupont(P), title(P, T), has_creator(P, C)
  FROM sioc:Post{P}.sioc:title{T}, {P}sioc:has_creator{C}
 WHERE C = “Dupont”
```

Explanation. In RVL, views (non canonical concepts) are separated from classes (canonical concepts). Thus, a new namespace is created for views (`http://www.lisi.ensma.fr/ex-view.rdf`). The first `VIEW` clause is used to create the view `PostDupont` with all properties defined on the class `Post`. This is done using a RQL query that searches all properties defined on `Post` or on a superclass of `Post (domain(P) >= sioc:Post)`. The second `VIEW` clause is used to compute instances of the view `PostDupont` by searching all messages (`P`) with their title (`T`) that have been created by `Dupont` (`C`). Notice that, for conciseness, we have only retrieved values of the `title` property. To define the complete view, other properties values must be searched.

Thus, RQL fulfil requirement 2. However, notice that the distinction between canonical and non canonical concepts forbids the definition of subsumption relationships between these two kind of concepts. Thus, it is necessary to reproduces manually this behaviour (by importing properties and their values in the NCCO).

Requirement 3 (Linguistic exploitation)

RDF-Schema allows to associate names defined in different natural languages to classes and properties. It permits also to define string values in different natural languages. RQL does not exploit these features.

Requirement 4 (Ontology model extensibility)

The ontology model supported by RQL is composed of the constructors `Class` and `Property`. This ontology model can be extended by specialization of these two constructors. But, a constructor can not be added if it does not inherit from `Class` or `Property`. For example, this limitation prevents to add the `Document` constructor of PLIB (to describe a concept by a document) or the `Ontology` constructor of OWL (to regroup all concepts defined in an ontology). Moreover, if these capabilities are defined on the data model of RQL, the language does not provide any operator to use them. Thus, we can not use RQL to define the OWL `AllValuesFrom` constructor.

Requirements 5 and 6 (Compatibility with the traditional database architecture)

The syntax of RQL is close to the one of object-oriented languages. However, it keeps no compatibility with SQL (requirement 5). In addition, RQL considers all instances as a URI independently of the classes it belongs to and of its properties values. Thus, it can not be used to manipulate the data schema (requirement 6).

Requirement 7 (Ontology & data definition and manipulation)

As we have seen in requirement 4, RQL does not provide a syntax to define the ontology model used. But RVL provides constructors to create new classes and properties and RUL can be used to insert new instances.

Example. Create the class `User` of our example and add some instances.

```
VIEW Class(“User”)<Resource>
     Property(“first_name”, User, xsd:string)
     Property(“susbscriber_of”, User, Forum)

INSERT User(&http://www.lisi.ensma.fr/Dupont)
INSERT first_name (&http://www.lisi.ensma.fr/Dupont, “patrick”)
```

Explanation. The constructor `Class` takes only one parameter: the URI of the class to define (for readability we use only the name `User`). This class is defined as a subclass of `Resource` (operator `<>`). Its properties are defined with the `Property` constructor. It takes three parameters: URI, domain and range of the property to define. The language RUL is then used to insert one instance of the class `User`. The operation `INSERT` is first used to define the URI of the instance and then to define its properties values (we define only value of the property `first_name`).

As this example shows, the RVL language is not a complete data definition language. It can only be used to create classes with a URI and properties with a URI, their domain and range. Other characteristics of classes and properties (e.g., names in different natural languages) can not be specified. On the other hand, RUL provides a complete set of operations (`INSERT-MODIFY-DELETE`) to manipulate data.

Requirement 8 (Queries on ontologies and on ontologies and data)

RQL is equipped with powerful path expressions that make it possible to query both ontologies and data.

Example. Return all instances of `Resource` with the classes it belongs to.

```
SELECT U $C
  FROM $C{U}
 WHERE C <= sioc:Resource
```

Explanation. The basic path expression `$C{U}` of the `FROM` clause introduces a variable `U` that denotes direct instances of a class `$C` (variables for classes are prefixed by `$` in RQL). The `WHERE` clause adds a condition to retrieve only direct instances of `Resource` or of a subclass of `Resource`. Finally, the `SELECT` clause projects the URI of instances and classes.

In this section, we have discussed capabilities of Semantic Web query languages w.r.t. the defined requirements. In next section, we study other languages that have been designed to keep some degree of compatibility with traditional databases.

ANALYSIS OF DATABASE-ORIENTED ONTOLOGY QUERY LANGUAGES

Several languages have been defined specifically for exploiting ontologies stored in databases such as Oracle extension to SQL (Chong et al., 2005; Wu et al., 2008), OntoQL (Jean et al., 2006), SOQA-QL (Ziegler et al., 2005) or CQL (Mizoguchi-Shimogori et al., 2002). In this section we discuss the most recent and active approaches i.e., Oracle extension to SQL and the OntoQL exploitation language.

Oracle Extension to SQL

The main objective of Oracle (Chong et al., 2005; Wu et al., 2008) was to provide efficient RDF data querying capabilities that integrate smoothly with SQL queries. The proposed solution is a SQL table function to query RDF data.

The table function named `RDF_MATCH` takes four parameters. The first parameter is a graph pattern to be matched. This graph pattern is defined with a syntax similar to the clause `WHERE` of SPARQL (basically a set of triples with variables). The second parameter specifies the RDF graph to be queried. The third parameter specifies the rulebase (if any) that must be used to infer new RDF data. A rulebase is composed of a set of rules. Each rule consists of a left hand side graph pattern for the antecedents, filter conditions, and a right hand side graph pattern for the consequents. Notice that RDF-Schema inference rules (e.g, transitive closure of the subsumption relationship) is created by the system (named `rdfs`) and available to users. Finally, the last parameter specifies user-defined namespaces aliases. The `RDF_MATCH` function returns a table having a column for each variable used in the graph pattern. Thus, this function can be seamlessly combined with SQL queries.

A strong effort has been made to optimize the `RDF_MATCH` function. Indeed, this function is rewritten has a SQL query so that it can be optimized with the rest of the query. Moreover, indexes and materialized views are used to execute efficiently queries. Scalability of the proposed approach has been demonstrated on 80 million RDF triples.

Requirement 1 (Queries at the ontological level)

The `RDF_MATCH` function can be used to express ontological queries decomposed in a triples pattern.

Example. Retrieve instances of the class `Item.`

```
SELECT t.i
  FROM TABLE(RDF_MATCH('(?i rdf:type Item)', NULL, NULL, NULL))
```

Explanation. The table function `RDF_MATCH` is used in the clause `FROM` of the query. It only takes a simple graph pattern as parameter to search instances of the class `Item` (`?i`). This function returns a table with a column named `i` for the variable `?i`. This variable can be projected in the `SELECT` clause.

Like SPARQL, result of the previous query depends on the triple represented in the RDF data source. For the `RDF_MATCH` function, it depends also of the rulebase specified. Indeed, if we specify the `rdfs` rulebase in the previous example, all (direct and indirect) instances of `Item` will be returned (the transitive closure of the subsumption relationship is computed by a rule).

Requirement 2 (Definition of non canonical concepts)

The data manipulation language of SQL can be used to define the non canonical class `PostDupont` A rule can be used to compute automatically its extension (set of instances).

Example[2]. Create the class `PostDupont` defined as all messages of the user `Dupont`.

```
INSERT INTO rdf_example
       VALUES (‘PostDupont’, rdf:type, rdfs:Class)

SELECT i FROM TABLE(
    RDF_MATCH(‘(?i rdf:type PostDupont)’, NULL, rb_example, NULL)
```

The rulebase `rb_example` contains the following rule:

```
(‘(?p rdf:type Post)(?p has_creator ?c)’,
 ‘?c = Dupont’,
 ‘(?p rdf:type PostDupont)’)
```

Explanation. In this example, we suppose that a table named `rdf_example` has been created to store RDF data. An `Insert` statement is used to insert the class `PostDupont`. Instances of `PostDupont` can be retrieved through a SQL query that uses the `RDF_MATCH` function. This function infers all instances of `PostDupont` thanks to a rule. The meaning of this rule is the following. If a message `?p` has been created by `?c` (antecedent defined in the first parameter) which is `Dupont` (filter defined in the second parameter), then `?p` is inferred to be an instance of `PostDupont` (consequent defined in the third parameter).

Requirement 3 (Linguistic exploitation)

Like SPARQL, the graph pattern of the `RDF_MATCH` function can use string values suffixed by a language code (e.g., `@fr` for a French string). Thus, our sample queries can be written like in SPARQL (see section 4).

Requirement 4 (Ontology model extensibility)

Like SPARQL, the `RDF_MATCH` function has been designed for RDF data and thus, it considers OWL ontology as a set of triples that use the OWL constructors defined in the OWL namespace. However, contrary to SPARQL, predefined rulebases are provided to take into account semantics of OWL constructors.

Requirements 5 and 6 (Compatibility with the traditional database architecture)

The `RDF_MATCH` function is directly integrated in SQL (requirement 5). To use this function, a table has to be created to store triples. Materialized views are automatically created to optimize

[2] For conciseness, we use a simplified syntax. For more details, the interested reader can consult the Oracle documentation (http://www.oracle.com/technology/tech/semantic_technologies/index.html)

queries on this table. However the representation of triples in this table can not be customized (requirement 6).

Requirement 7 (Ontology & data definition and manipulation)

The data manipulation language of SQL can be used to create classes and instances.

Example. Create the class `User` of our example and add some instances.

```
INSERT INTO rdf_example VALUES (‘User’, rdf:type, rdfs:Class)
INSERT INTO rdf_example
       VALUES (‘User’, rdfs:subClassOf, ‘Resource’)
INSERT INTO rdf_example
       VALUES (‘first_name’, rdf:type, rdf:Property) ...

INSERT INTO rdf_example VALUES (‘User1’, rdf:type, User)
INSERT INTO rdf_example VALUES (‘User1’, first_name, ‘Patrick’) …
```

Explanation. All information has to be inserted as RDF triples. Thus, many `INSERT` statements are necessary to insert the class `User` with its properties and its instances.

As this example shows, usage of the DML of SQL can be tedious to create ontologies and their instances.

Requirement 8 (Queries on ontologies and on ontologies and data)

Since, the `RDF_MATCH` function provides similar graph pattern matching capabilities as SPARQL, it can be used to combined ontology and data querying.

Thus, the `RDF_MATCH` is a powerful extension of SQL to query RDF data. In the next section we discuss the OntoQL exploitation language that extends SQL following a different approach.

OntoQL

OntoQL (Jean, 2007; Jean et al., 2006) has been defined specifically for OBDBs. To exploit data, OntoQL has been defined in different layers:

- data access at the logical level by compatibility with SQL;
- data access at the ontological level, CCO layer. Primitive concepts being mainly defined with object-oriented constructors, OntoQL adapts and extends SQL99 providing powerful relational-object operators;
- data access at the ontological level, NCCO layer. OntoQL provides a View Definition Language for defining and querying defined concepts;
- data access at the ontological level, LO layer. Each class and each property can be referenced by a name (in a given natural language) in an OntoQL query.

Ontologies being recorded in OBDBs, OntoQL provides an Ontology Definition, Manipulation and Query Language (ODL, OML and OQL) to exploit them. To keep a uniform syntax, these languages have been designed to keep a syntax near SQL99. They are based on a core ontology model that contains the main constructors of existing ontology models. This core ontology model is an object-oriented model composed of classes and properties named entities and attributes to distinguish them from ontology classes and properties. This core ontology model can be extended with new entities and new attributes using the ODL. Since this model can be extended, names of entities and attributes are not encoded as keywords in the OntoQL grammar but they are prefixed by the character #.

Requirement 1 (Queries at the ontological level)

Ontological queries can be expressed following a SQL-like syntax.

Example. Retrieve instances of the class `Item.`

```
SELECT uri
  FROM Item
 USING NAMESPACE ‘http://rdfs.org/sioc/ns’
```

Explanation. The `USING` clause is used to define a default namespace for the query. When the SIOC namespace is set, each element without prefix (e.g., `Item`) is searched in this ontology. This query retrieves all instances of `Item`. To retrieve only direct instances, one can use the `ONLY` operator like in SQL99 (i.e., `ONLY(Item)`).

Requirement 2 (Definition of non canonical concepts)

OntoQL provides a View Definition Language to define non canonical concepts.

Example. Create the class `PostDupont` defined as all messages of the user `Dupont`.

```
CREATE #Class PostDupont AS VIEW UNDER Post

CREATE VIEW OF PostDupont AS
  SELECT * FROM Post AS p
   WHERE p.has_creator.last_name = ‘Dupont’
```

Explanation. The first statement creates the class `PostDupond` as a non canonical concept (keyword `VIEW`), subclass of `Post`. The second statement defines the extension of this class by providing an OntoQL query that computes its instances.

When non canonical classes are defined using OWL constructors (e.g., restrictions), an inference engine can automatically compute subsumption relationships between canonical and non canonical concepts. OntoQL (like RQL) has not this capability: canonical and non canonical classes must be placed manually in the hierarchy.

Requirement 3 (Linguistic exploitation)

In OntoQL, each class and each property can be referenced by a name in a given natural language. It makes it possible to write the same query in many natural languages.

Example. Return the first name and last name of users with a query written using English names of concepts and one using French names.

```
SELECT “first name”,
       “last name”
  FROM User
USING LANGUAGE EN
```

```
SELECT prénom,
       nom
  FROM Utilisateur
 USING LANGUAGE FR
```

Explanation. The `USING` clause of OntoQL can be used to specify the natural language in which a query is expressed. The left query used English names. For names with a space (e.g., `first name`) double quotes are used. The right query is equivalent but written with French names.

<u>Requirement 4 (Ontology model extensibility)</u>

The ontology model supported by OntoQL can be extended using its ODL.

Example. Add the OWL constructor `AllValuesFrom`

```
CREATE ENTITY #AllValuesFrom UNDER #Class(
   #onProperty REF(#Property),
   #allValuesFrom REF(#Class)
)
```

Explanation. An OWL restriction being a class, the `ALLValuesFrom` constructor is added as a subentity of `#Class`. This entity has an attribute `#allValuesFrom` to specify the class in which its instances take their values for the property defined by the attribute `#onProperty`.

When an extension is made by specialization, like in the previous example, new entities inherit the behaviour of their super entities. This behaviour is defined in the operational semantics of the core ontology model. Thus, every specialization of the entity `#Class` defines a new category of classes which supports by inheritance the usual behaviour of a class. As a consequence, each OWL restriction may be associated to a container to store its instances. These instances may be computed using a view (all instances having all values of a property in a given class). However, OntoQL does not provide yet the capability to express this semantics.

<u>Requirement 5 (SQL compatibility)</u>

If a default namespace is not specified, each element of an OntoQL query without a namespace alias prefix is considered as an element of the logical model (a table or a column). Thus, an OntoQL query without namespace specification is considered as a SQL query (requirement 5). Moreover, the semantics of OntoQL (an algebra) has been defined so that each operator returns a relation (Jean et al., 2007a). Thus, queries at the logical level and ontological level can be combined (like Oracle).

Requirement 6 (Access to the conceptual model of data)

OntoQL provides basic functionalities to define and access the conceptual model of data.

Example. Create the conceptual model corresponding to the class `User` knowing that only the properties `first_name`, `last_name` and `email` will be used in the target application.

```
CREATE EXTENT OF User (“first name”, “last name”, email)
           TABLE T_User (first_name, last_name, email)
```

Explanation. The `CREATE EXTENT` statement defines the conceptual model corresponding to a class specifying the set of properties used to describe instances of this class. The `TABLE` clause is used to choose the logical implementation of this conceptual model (i.e., tables and columns necessary).

Currently, in OntoQL, the conceptual model of data can only be defined through a subset of the ontology.

Requirement 7 (Ontology & data definition and manipulation)

The data manipulation language of SQL can be used to create classes and instances.

Example. Create the class `User` of our example and add some instances.

```
CREATE #Class User UNDER Resource (
  DESCRIPTOR (#name[fr] = ‘Utilisateur)
  #property (“first name” String, “last name” String, …)
)

INSERT INTO User VALUES(‘User1’, ‘Patrick’, ‘Dupond’)
```

Explanation. The first statement creates the class `User`. This class is defined as a class (`#Class`), subclass of `Resource`. The name in French of this class is specified in the `DESCRIPTOR` clause. The `#property` clause is used to create properties having `User` as domain. The second statement supposes that we have created an extent for the class `User` (see previous requirement). It uses a SQL-like `INSERT` statement to insert an instance.

One limitation of the data manipulation language of OntoQL is that it does not support multi-instanciation, i.e., that an instance may belong to many classes not linked by subsumption relationships. This limitation allows to define an efficient storage structure when instances have many properties values (Dehainsala et al., 2007).

Requirement 8 (Queries on ontologies and on ontologies and data)

OntoQL is equipped of operators to combine ontologies and data querying. In particular, it provides the `TYPEOF` operator to return the class an instance belongs to.

Example. Return all instances of `Resource` with the classes it belongs to.

```
SELECT r.uri, TYPEOF(r).#name[en]
  FROM Resource AS r
```

Explanation. The `FROM` clauses introduces the alias `r` on instances of the class `Resource`. For each instance, its URI is projected as well as the name in English of the class it belongs to (retrieved with the `TYPEOF` operator).

This brief presentation of OntoQL ends our analysis of capabilities of ontology query languages to exploit OBDBs. We draw main conclusions in the next section.

CONCLUSIONS

In this chapter, we have presented capabilities of the main ontology query languages to exploit databases extended with ontologies. To compare these languages, we have first defined a set of requirements for an architecture that extends the traditional ANSI/SPARC architecture with conceptual and ontological levels. Then, we have discussed the capabilities of two Semantic Web query languages (SPARQL and RQL), and two database-oriented ontology query languages (Oracle extension to SQL and OntoQL) to fulfill these requirements. The result of this study is summarized in Table 1. In this table, the symbol ● is used when the requirement is fulfilled, O when it is partially fulfilled, and - when it is not fulfilled.

	SPARQL	**RQL**	**Oracle**	**OntoQL**
Queries at the ontological level	O	●	O	●
Definition of non canonical concepts	O	O	●	O
Linguistic exploitation	●	-	●	●
Ontology model extensibility	O	O	●	O
SQL compatibility	-	-	●	●
Access to the conceptual model of data	-	-	-	O
Ontology & data definition and manipulation	-	O	O	O
Queries on ontologies and on ontologies and data	●	●	●	●

Table 1. Analysis of the main ontology query languages w.r.t. to the defined requirements

Results presented in Table 1 lead us to draw the following conclusions. The main drawback of SPARQL and Oracle approaches to query OBDBs is that they consider all information as RDF data. As a consequence, when querying at the ontological level, the semantics of the ontology models has to be coded either in the query interpreter of SPARQL or by a set of deductive rules in the Oracle approach. Moreover, they don't provide operators to exploit this semantics (e.g., an operator to retrieve direct instances of a class). Another consequence is that the syntax of these two languages is adapted to query triple data. As a consequence, an ontological query has to be decomposed in triples (e.g, `(?i rdf:type User) (?i sioc:name ?n)` for retrieving names of users) which can be tedious for users.

On the contrary, the semantics of RQL and OntoQL are based on the core constructors of ontology models. They provide a syntax near the one of object-oriented query languages. We think that this syntax is more adapted for ontologies than a triple syntax because ontologies share many constructors with the object-oriented data model.

However, RQL and OntoQL may benefit from efforts made by Oracle to provide semantics technologies. Indeed, Oracle provides customizable and optimized triple storage with capability to load quickly a huge amount of RDF data. Thus, Oracle provides the built-in functions to serve as a scalable storage structure for RQL or OntoQL. Moreover, Oracle is now equipped with an inference engine for RDFS/OWL constructs which will be particularly useful for RQL and OntoQL to take into account the semantics of OWL. As a future work we plan to put this observation in application by implementing OntoQL on top of Oracle.

REFERENCES

Bailey, J., Bry, F., Furche, T., & Schaffert, S. (2005). Web and Semantic Web Query Languages: A Survey. In *Reasoning Web, First International Summer School*, Lecture Notes in Computer Science, pages 35-133. Springer.

Brickley, D. & Guha, R. V. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. World Wide Web Consortium. http://www.w3.org/TR/rdf-schema.

Broeskstra, J. & Kampman, A. (2003). SeRQL: A Second Generation RDF Query Language. In *SWADEurope Workshop on Semantic Web Storage and Retrieval*.

Carroll, J. J., Bizer, C., Hayes, P., & Stickler, P. (2005). Named Graphs, Provenance and Trust. In *Proceedings of the 14th international conference on World Wide Web (WWW'05)*, pages 613–622, New York, NY, USA. ACM Press.

Cattell, R. G. G. (1993). *The Object Database Standard : ODMG-93*. Morgan Kaufmann.

Chong, E. I., Das, S., Eadon, G., & Srinivasan, J. (2005). An Efficient SQL based RDF Querying Scheme. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, pages 1216–1227.

Dean, M. & Schreiber, G. (2004). *OWL Web Ontology Language Reference*. World Wide Web Consortium. http://www.w3.org/TR/owl-ref.

Dehainsala, H., Pierra, G., & Bellatreche, L. (2007). OntoDB: An Ontology-Based Database for Data Intensive Applications. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, volume 4443 of Lecture Notes in Computer Science, pages 497–508. Springer.

Fankam, C., Jean, S., Bellatreche, L., & Ameur, Y. A. (2008). Extending the ANSI/SPARC Architecture Database with Explicit Data Semantics: An Ontology-Based Approach. In Morrison, R., Balasubramaniam, D., and Falkner, K. E., editors, *Proceedings of the 2nd European Conference on Software Architecture (ECSA'08)*, volume 5292 of Lecture Notes in Computer Science, pages 318-321.Springer.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199-220.

Guarino, N. (1998). Formal Ontology and Information Systems. In Guarino, N., editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems (FOIS'98)*, pages 315. IOS Press.

Haase, P., Broekstra, J., Eberhart, A., & Volz, R. (2004). A Comparison of RDF Query Languages. In *Proceedings of the 3nd International Semantic Web Conference (ISWC'04)*, pages 502-517.

IEC61360-4 (1999). Standard data element types with associated classification scheme for electric components - Part 4 : IEC reference collection of standard data element types, component classes and terms. Technical report, International Standards Organization.

Jean, S., Aït-Ameur, Y., & Pierra, G. (2006). 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, pages 704-721. Springer.

Jean, S. (2007). *OntoQL, un langage d'exploitation des bases de donnes base ontologique*. PhD thesis, LISI/ENSMA and University of Poitiers.

Jean, S., Aït-Ameur, Y., & Pierra, G. (2007a). An Object-Oriented Based Algebra for Ontologies and their Instances. In *Proceedings of the 11th East European Conference in Advances in Databases and Information Systems (ADBIS'07)*, volume 4690 of Lecture Notes in Computer Science, pages 141-156. Springer.

Jean, S., Pierra, G., & Aït-Ameur, Y. (2007b). Domain Ontologies: a Database-Oriented Analysis, volume 1 of *Lecture Notes in Business Information Processing*, pages 238-254. Springer Berlin Heidelberg.

Karvounarakis, G., Magkanaraki, A., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M., & Tolle, K. (2004). RQL: A Functional Query Language for RDF. In Gray, P. M. D., Kerschberg, L., King, P. J. H., and Poulovassilis, A., editors, *The Functional Approach to Data Management: Modelling, Analyzing and Integrating Heterogeneous Data*, LNCS, pages 435-465.Springer-Verlag.

Magiridou, M., Sahtouris, S., Christophides, V., & Koubarakis, M. (2005). RUL : A Declarative Update Language for RDF. In *Proceedings of the 4th International Semantic Web Conference (ISWC’05)*, pages 506–521.

Magkanaraki, A., Tannen, V., Christophides, V., & Plexousakis, D. (2004). Viewing the Semantic Web Through RVL Lenses. *Journal of Web Semantics*, 1(4) :359–375.

Miller, L., Seaborne, A., & Reggiori, A. (2002). Three Implementations of SquishQL, a Simple RDF Query Language. In *Proceedings of the 1st International Semantic Web Conference (ISWC’02)*, pages 423–435.

Mizoguchi-Shimogori, Y., Murayama, H., & Minamino, N. (2002). Class Query Language and its application to ISO13584 Parts Library Standard. In *Proceedings of the 9th European Concurrent Engineering Conference (ECEC’02)*, pages 128–135.

Patel-Schneider, P. F. & Motik, B. (2008). OWL2 Web Ontology Language: Mapping to RDF Graphs. *W3C Working Draft 08 October 2008*. http://www.w3.org/TR/owl-ref.

Pierra, G. (2008). Context Representation in Domain Ontologies and its Use for Semantic Integration of Data. *Journal Of Data Semantics (JODS),* X:173-210.

Prud'hommeaux, E. & Seaborne, A. (2008). SPARQL Query Language for RDF. W3C *Candidate Recommendation 15 January 2008,* from http://www.w3.org/TR/rdf-sparql-query/.

Seaborne, A. (2004). RDQL– A Query Language for RDF. *W3C Member Submission 9 January 2004*. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.

Seaborne, A. & Manjunath, G. (2008). SPARQL/Update A language for updating RDF graphs. ttp://jena.hpl.hp.com/~afs/SPARQL-Update.html.

Tolle, K. & Wleklinski, F. (2004). *easy RDF Query Language (eRQL).* http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/eRQL.

Wu, Z., Eadon, G., Das, S., Chong, E. I., Kolovski, V., Annamalai, M., & Srinivasan, J. (2008). Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In *Proceedings of the 24th International Conference on Data Engineering (ICDE'08),* pages 1239-1248.

Xuan, D. N., Bellatreche, L., & Pierra, G. (2006). A Versioning Management Model for Ontology-Based Data Warehouses. In *Proceedings of the 8th International Conference on Data Warehousing and Knowledge Discovery (DaWak'06),* pages 195-206.

Ziegler, P., Sturm, C., & Dittrich, K. R. (2005). Unified Querying of Ontology Languages with the SIRUP Ontology Query API. In *Datenbanksysteme in Business, Technologie und Web (BTW'05),* pages 325-344.

KEY TERMS & DEFINITIONS

Ontology: a formal and consensual dictionary of categories and properties of entities of a domain and the relationships that hold among them.

Ontology-based Database (OBDB): a data source which contains i) a (local) ontology, ii) possibly some references from this ontology to external (shared) ontologies, iii) a set of data, iv) and finally a relationship between each data and the ontological notion which explicit its meaning. An OBDB has two main characteristics: i) both ontologies and data are represented in a unique database and the same processing can be applied on them (Insert, Update, Querying, Versioning, etc.); ii) any data is associated to an ontological element which defines it meaning and vice versa (Dehainsala et al., 2007).

Ontology query language: a language that has been designed to exploit ontologies and their instances.

BIOGRAPHY

Stephane JEAN is assistant professor at the University of Poitiers. He is a member of the data engineering research team of the laboratory of computing (LISI) at the National School of Engineers in Mechanics and Aeronautics (ENSMA). He conducts research in the area of ontologies management especially in the context of databases. During his PhD thesis he has designed the OntoQL exploitation language for databases that store ontologies and the data they describe. This language is currently used in several projects in the engineering area.

Yamine AIT AMEUR is full professor at the National School of Engineers in Mechanics and Aeronautics (ENSMA). He is the director of the laboratory of computing (LISI) at ENSMA and head of the data engineering research team. Formal methods, ontology based data modeling and database are his main topics of interest. He has written several research papers and supervised several PhD thesis. His research results have been applied in various engineering areas like aeronautics, embedded systems and petroleum industry.

Guy Pierra is professor of computer science at ENSMA, Poitiers. He funded the Laboratory of computing (LISI) in Poitiers in 1992 as a laboratory common to ENSMA and Poitiers University. His main interests are data engineering, software engineering, ontology-based modeling, CAD/CAM and human-computer interaction. He published one book and more than hundred papers on these topics. Since the late 80's, G. Pierra and his colleagues of the LISI data engineering team have developed an ontology model, and an ontology-based approach, providing for modeling technical knowledge and industrial products. This approach, known as PLIB, has also been published as a series of international standard identified as ISO 13584. The PLIB ontology model and approach are now largely used in the engineering domain both for exchanging electronic catalogues and for developing ontology-based database of industrial components.