Extending XQuery for Semantic Web Reasoning

Jesús Manuel Almendros-Jiménez

Dpto. de Lenguajes y Computación,
Universidad de Almería, Spain
jalmen@ual.es

Abstract. In this paper we investigate an extension of the XQuery language for querying and reasoning with OWL-style ontologies. The proposed extension incorporates new primitives (i.e. boolean operators) in XQuery for the querying and reasoning with OWL-style triples in such a way that XQuery can be used as query language for the Semantic Web. In addition, we propose a Prolog-based implementation of the extension.

1 Introduction

XQuery [31,11] is a typed functional language devoted to express queries against XML documents. It contains *XPath 2.0* [30] as a sublanguage. *XPath 2.0* supports navigation, selection and extraction of fragments from XML documents. XQuery also includes expressions to construct new XML documents and to join multiple documents. The *Web Ontology Language (OWL)* [32] is a proposal of the *W3C consortium*[1] for ontology modeling. OWL is an ontology language based on the so-called *Description Logic (DL)* [6]. OWL is syntactically layered on the *Resource Description Framework (RDF)* [29], whose underlying model is based on triples. The *RDF Schema (RDFS)* [28] is also an ontology language, enriching RDF with meta-data, however, OWL offers more complex relationships than RDF(S). OWL includes, among others, means to infer that items with various properties are members of a particular class, means to define complex vocabularies: equivalence and inclusion between entities, symmetric, inverse and transitive properties, cardinality restrictions on properties, etc.

In this paper we present an extension of XQuery for querying and reasoning with OWL-style ontologies. Such extension incorporates to XQuery mechanisms for the traversal and reasoning with OWL statements. The main features of our proposal can be summarized as follows:

– XQuery has been developed for querying XML documents, however, Web data can be also represented by means of RDF(S) and OWL. Therefore, XQuery should support the simultaneous querying of XML data by both its structure and by its associated meta-data given in form of OWL-style ontologies. This is an important problem for supporting data discovery tasks

[1] http://www.w3.org

S. Abreu and D. Seipel (Eds.): INAP 2009, LNAI 6547, pp. 117–134, 2011.

against collections of XML data. The problem of simultaneous querying of both data and meta-data has been intensely studied for other data models, but the topic is in its infancy for XML data. Our approach could be used in large-scale document repositories allowing document meta-data to be used in the generation of reports and aggregate documents. The proposed extension allows to query XML/RDF(S)/OWL documents and to obtain as output the same kind of documents.
- In his current form, XQuery can be used for querying RDF(S) and OWL assuming a XML representation of RDF(S) and OWL. However, the representation of RDF(S) and OWL in XML is not standarized. While query languages for XML + RDF do exist, their application in practice is restricted by the assumption of a particular encoding of the ontology hierarchy in XML. In addition, expressing queries in XQuery against RDF and OWL in XML format can be too sophisticated. The proposed extension is independent of the XML encoding of ontologies, working directly on the conceptual RDF(S) (and OWL) data model: triples.
- RDF(S) and OWL querying should be combined with reasoning. RDF(S) and OWL allow to express and infer complex relationships between entities which should be exploited by means of a query language. The proposed extension is able to use semantic information inferred from RDF(S) and OWL resources.
- Finally, we will propose an implementation of the extension in Prolog.

Now, we will review the existing proposals of query languages for the Semantic Web and we will present the advantages of our proposal.

A great effort has been made for defining query languages for RDF(S)/OWL documents (see [7,17] for surveys about this topic). The proposals mainly fall on extensions of *SQL*-style syntax for handling the *triple-based RDF structure*. In this line the most representative languages are SquishQL [22], SPARQL [14] and RQL [19]. However, they are designed for RDF(S) and OWL querying in the following sense. The syntax of the previous languages resemble SQL extensions for expressing queries in which the database contains RDF triples. The answers to such queries resemble to SQL answers (i.e. tables). In our proposal, the extension of XQuery can express answers as XML documents, and, in particular, as RDF documents, and therefore XML/RDF(S)/OWL documents work as input and as output of the XQuery extension. It does not mean that the existing proposals of RDF(S)/OWL query languages cannot produce XML documents as output but they require the use of *ad-hoc* languages for formatting the output. This is the case, for instance, of SPARQL which incorporates to the queries the so-called *SPARQL Results Document* which defines the XML format of the answers. In our case, we take advantage of the XQuery mechanisms for obtaining XML documents as output.

There exist in the literature some proposals of extensions of *XPath*, *XSLT* and *XQuery* languages for the handling of RDF(S) and OWL. XPath, XSLT were also designed for XML, and therefore the proposals are *XML-based* approaches. In this line the most representative languages are XQuery for RDF (*'the Syntactic Web Approach"*) [23], RDF Twig [33], RDFPath [26], RDFT [12] and

XsRQL [20]. Such languages assume the serialization (i.e. encoding, representation) of RDF(S) and OWL in XML. However, such serialization is not standard. Our proposal aims to amalgamate the SPARQL (SquishQL and RQL) design with the XML-based approaches in the following sense. In our proposed extension of XQuery we do not assume a fixed serialization of RDF(S) and OWL in XML, rather than, we extend the XQuery syntax in order to admit the traversal of RDF triples similarly to SPARQL-style query languages. However, we retain the XQuery capability to generate XML documents as output. In addition, our proposal extends the syntax of XQuery but XQuery can be still used for expressing queries against XML documents. Therefore, with our extension XQuery can be used for expressing queries which combine input resources which can have XML/RDF(S)/OWL format, any of them. As far as we know, our proposal is the first one in which heterogeneous resources can be combined in a query.

Finally, there are some proposals of logic-based query languages for the Semantic Web. This is the case of TRIPLE [25], N3QL [8] and XCerpt [24,15]. They have their own syntax similar to *deductive logic languages*, and handle RDF(S)/OWL (in the case of TRIPLE and N3QL) and XML/RDF (in the case of XCerpt). Our proposal aims to define an extension of an standarized query language like XQuery rather than defining a new query language (i.e. syntax and semantics). However, our work is close to logic-based query languages in the following sense. We will propose a logic-based implementation of XQuery using Prolog as host language.

The question now is, why Prolog? The existing XQuery implementations either use functional programming (with *Objective Caml* as host language) or Relational DataBase Management Systems (RDBMS). In the first case, the *Galax* implementation [21] encodes XQuery into *Objective Caml*, in particular, encodes XPath. Since XQuery is a functional language (with some extensions) the main encoding is related with the type system for allowing XML documents and XPath expressions to occur in a functional expression. With this aim an specific type system for handling XML tags, the hierarchical structure of XML, and sequences of XML items is required. In addition, XPath expressions can implemented from this representation. In the second case, XQuery has been implemented by using a RDBMS. It evolves in most of cases the encoding of XML documents by means of relational tables and the encoding of XPath and XQuery. The most relevant contribution in this research line is *MonetDB/XQuery* [9]. It consists of the *Pathfinder* XQuery compiler [10] on top of the *MonetDB* RDBMS, although *Pathfinder* can be deployed on top of any RDBMS. MonetDB/XQuery encodes the XML tree structure in a relational table following a pre/post order traversal of the tree (with some variant). XPath can be implemented from such table-based representation. XQuery can be implemented by encoding *flwor* expressions into the *relational algebra*, extended with the so-called *loop-lifted staircase join*.

The motivation for using a logic-based language for implementing XQuery is that our extension of XQuery has to handle OWL-style ontologies. However, RDF(S) and OWL should be handled not only as a database of triples, but reasoning with RDF(S) and OWL should be incorporated. The underlying model

of RDF(S) and OWL is suitable for reasoning about data and metadata. For instance, class and property (i.e. concepts and roles) hierarchies can define complex models in which a given individual can belong to more than one complex class. Inferencing and reasoning is then required. Logic programming (and in particular, Prolog) is a suitable framework for inferencing and reasoning. It has been already noted by some authors. In this area of research, some authors [16,27] have studied the intersection of Description Logic and Logic Programming, in other words, which fragment of Description Logic can be expressed in Logic Programming. In [16], they have defined the so-called *Description Logic Programming (DLP)*, which corresponds with a fragment of $\mathcal{SHIQ}$. With this aim, firstly, the fragment of DL is encoded into a fragment of FOL; and after the fragment of FOL is encoded into logic programming. Other fragments of OWL/DL can be also encoded into logic programming, in particular, Volz [27] has encoded fragments of $\mathcal{SHOIN}$ into *Datalog*, *Datalog(=)*, *Datalog(=,IC)* and *Prolog(=,IC)*; where "=" means "with equality", and "IC" means "with Integrity constraints". Some other proposals have encoded the fragment $\mathcal{SHIQ}$ into *disjunctive Datalog* [18], and into *Datalog(IC,≠,not)* [13], where "not" means "with negation".

Implementing XQuery by means of Prolog we will have a logic based implementation in which we can accommodate the Semantic Web extension. In previous works [4,5] we have proposed a Prolog based implementation of the XQuery (and XPath) languages. In such proposal, XML documents are translated into a Prolog program by means of facts and rules. Then, a query of XQuery is encoded by means of Prolog rules, and an specific goal is used for obtaining the answer. The rules of the encoding of the query specializes the Prolog program representing the input XML document. From the Prolog computed answers we are able to rebuild the output XML document. The contribution of this paper is that both XQuery constructs and RDF(S)/OWL reasoning can be expressed by means of rules and therefore a Prolog based implementation of the extension of XQuery can be easily defined.

In our proposal, we have focused on one of the Volz's fragments [27] which can be encoded into *Datalog*. It is a fragment of $\mathcal{SHOIN}$, which includes the OWL vocabulary: $rdf : type$, $rdfs : subClassOf$, $rdfs : subPropertyOf$, $owl : equivalent\ Property$, $owl : equivalentClass$, $rdfs : domain$, $rdfs : range$, $owl : someValues\ From$, $owl : has\ Value$ and $owl : allValuesFrom$, and handles $owl : union$ and $owl : inter\ section$ operators, and $owl : TransitiveProperty$, $owl : SymmetricPro\ perty$ and $owl : inverseOf$ properties. All of them are used with some restrictions. Since the main aim of our approach is to exhibit the combination of querying of XML, RDF and OWL, the restriction to a simple kind of ontology makes our work easier. However, it does not affect substantially to the relevance of the approach since interesting examples can be handled in our query language. We believe that more complex ontologies which can be encoded into extensions of Datalog could be also integrated in our approach following the same ideas presented here. Such extensions are considered as future work.

We would like to remark that our work also continues a previous work about XQuery. In [1], we have studied how to define an extension of XQuery for

$C \sqsubseteq D$	(`rdfs:subClassOf`)	$E \equiv F$	(`owl:equivalentClass`)
$P \sqsubseteq Q$	(`rdfs:subPropertyOf`)	$P \equiv Q$	(`owl:equivalentProperty`)
$P \equiv Q^-$	(`owl:inverseOf`)	$P \equiv P^-$	(`owl:SymmetricProperty`)
$P^+ \sqsubseteq P$	(`owl:TransitiveProperty`)	$\top \sqsubseteq \forall P^-.D$	(`rdfs:domain`)
$\top \sqsubseteq \forall P.D$	(`rdfs:range`)	$P(a, b)$	(property fillers)
$A(a)$	(individual assertions)		

Fig. 1. TBox and ABox Formulas (see [27])

querying RDF documents. Similarly to the current proposal, such extension allows to query XML and RDF resources with XQuery, and queries can take as input XML and RDF documents, obtaining also as output both kind of documents. The proposed RDF extension of XQuery uses the **for** construction of XQuery for traversing RDF triples. In addition, our RDF extension of XQuery is equipped with built-in boolean operators for RDF/RDFS properties like *rdf:type*, *rdfs:domain*, *rdfs:range*, *rdfs:subClassOf*, *rdfs:subPropertyOf*, and so on. Such built-in boolean operators can be used for reasoning about RDF, that is, for using the RDFS entailment relationship in the queries. In addition, we have studied a Prolog based implementation of this extension.

Finally, we have tested the proposal of this paper by means of a prototype implemented in SWI-Prolog. The prototype consists of a XQuery implementation and a RDF(S)/OWL reasoning module. The *XQuery* implementation can be downloaded from `http://indalog.ual.es/XQuery` and the RDF(S)/OWL module from `http://indalog.ual.es/OProlog`.

The structure of the paper is as follows. Section 2 will present the kind of ontologies we consider in our framework. Section 3 will describe the extension of XQuery for RDF(S)/OWL. Section 4 will show the implementation in Prolog and, finally, Section 5 will conclude and present future work.

2 Ontology Representation

In this section we will show which kind of ontologies will be handled in our framework. We will restrict ourselves to the case of a subset of DL expressible in Datalog [27], and therefore in Prolog. Such kind of DL ontologies contains a **TBox** and a **ABox** of axioms whose syntax is shown in Figure 1. In Figure 1, E and F are class descriptions of type $\mathcal{E}$ (see Figure 2), C is a class description of left-hand side type (type $\mathcal{L}$, see Figure 2), and D is a class description of right-hand side type (type $\mathcal{R}$, see Figure 2), P and Q are property names, a, b are individual names, and A is an atomic class.

Basically, the proposed subset of DL restricts the form of class descriptions in right and left hand sides of subclass and class equivalence axioms. Such restriction is required to be encoded by means of Datalog rules. Roughly speaking, the universal quantification is only allowed in the right hand side of DL formulas, which corresponds in the encoding to the occurrences of the same quantifier in the left hand side (i.e. head) of rules. The same kind of reasoning can be used

type $\mathcal{E}$	
A	atomic class
$E_1 \sqcap E_2$	`owl:intersectionOf`
$\exists P.\{o\}$	`owl:hasValue`

type $\mathcal{L}$	
A	atomic class
$C_1 \sqcap C_2$	`owl:intersectionOf`
$\exists P.\{o\}$	`owl:hasValue`
$C_1 \sqcup C_2$	`owl:unionOf`
$\exists P.C$	`owl:someValuesFrom`

type $\mathcal{R}$	
A	atomic class
$D_1 \sqcap D_2$	`owl:intersectionOf`
$\exists P.\{o\}$	`owl:hasValue`
$\forall P.D$	`owl:allValuesFrom`

Fig. 2. Allowed Types (see [27])

TBox	
(1) $Man \sqsubseteq Person$	*(2)* $Woman \sqsubseteq Person$
(3) $Person \sqcap \exists author\_of.Manuscript \sqsubseteq Writer$	*(4)* $Paper \sqcup Book \sqsubseteq Manuscript$
(5) $Book \sqcap \exists topic.\{\text{“XML”}\} \sqsubseteq XMLbook$	*(6)* $Manuscript \sqcap \exists reviewed\_by.Person \sqsubseteq Reviewed$
(7) $Manuscript \sqsubseteq \forall rating.Score$	*(8)* $Manuscript \sqsubseteq \forall topic.Topic$
(9) $author\_of \equiv writes$	*(10)* $average\_rating \sqsubseteq rating$
(11) $authored\_by \equiv author\_of^{-}$	*(12)* $\top \sqsubseteq \forall\ author\_of.Manuscript$
(13) $\top \sqsubseteq \forall\ author\_of^{-}.Person$	*(14)* $\top \sqsubseteq \forall\ reviewed\_by.Person$
(15) $\top \sqsubseteq \forall\ reviewed\_by^{-}.Manuscript$	
ABox	
(1) Man(“Abiteboul”)	
(2) Man(“Buneman”)	*(3) Man(“Suciu”)*
(4) Book(“Data on the Web”)	*(5) Book(“XML in Scottland”)*
(6) Paper(“Growing XQuery”)	*(7) Person(“Anonymous”)*
(8) author_of(“Abiteboul”, “Data on the Web”)	*(9) authored_by(“Data on the Web”, “Buneman”)*
(10) author_of(“Suciu”, “Data on the Web”)	*(11) author_of(“Buneman”, “XML in Scottland”)*
(12) writes(“Simeon”, “Growing XQuery”)	*(13) reviewed_by(“Data on the Web”, “Anonymous”)*
(14) reviewed_by(“Growing”, “Almendros”)	*(15) average_rating(“Data on the Web”, “good”)*
(16) rating(“XML in Scottland”, “excellent”)	*(17) average_rating(“Growing XQuery”, “good”)*
(18) topic(“Data on the Web”, “XML”)	*(19) topic(“Data on the Web”, “Web”)*
(20) topic(“XML in Scottland”, “XML”)	

Fig. 3. An Example of Ontology

for explaining why existential quantifiers can only occur in left hand sides of DL formulas. Union formulas are required to appear in left hand sides which corresponds with the definition of two or more rules in the encoding. The details about the encoding can be found in Section 4.1. Let us see now an example of a such DL ontology (see Figure 3).

The ontology of Figure 3 describes, in the **TBox**, meta-data in which the elements of *Man* and the elements of *Woman* are elements of *Person* (cases (1) and (2)); and the elements of *Paper* and *Book* are elements of *Manuscript* (case (4)). In addition, a *Writer* is a *Person* who is the *author_of* a *Manuscript* (case (3)), and the class *Reviewed* contains the elements of *Manuscript reviewed_by* a *Person* (case (6)). Moreover, the *XMLBook* class contains the elements of *Manuscript* which have as *topic* the value “XML” (case (5)). The classes *Score* and *Topic* contain, respectively, the values of the properties *rating* and *topic* associated to *Manuscript* (cases (7) and (8)). The property *average_rating* is a subproperty of *rating* (case (10)). The property *writes* is equivalent to *author_of* (case (9)), and *authored_by* is the inverse property of *author_of* (case (11)). Finally, the property *author_of*, and conversely, *reviewed_by*, has as domain a *Person* and as range a *Manuscript* (cases (12)-(15)).

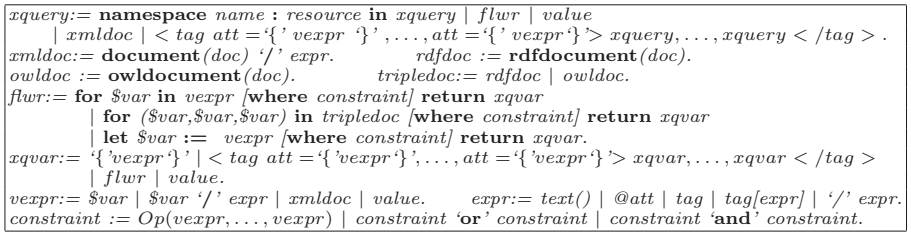

Fig. 4. Extension of the Core of XQuery

The **ABox** describes data about two elements of *Book*: "Data on the Web" and "XML in Scottland" and a *Paper*: "Growing XQuery". It describes the *author_of* and *authored_by* relationships for the elements of *Book* and the *writes* relation for the elements of *Paper*. In addition, the elements of *Book* and *Paper* have been reviewed and rated, and they are described by means of a topic.

3 Extended XQuery

Now, we would like to show how to query an OWL-style ontology by means of XQuery. The grammar of the extended XQuery for querying XML, RDF(S) and OWL is described in Figure 4, where "name **:** resource" assigns name spaces to URL resources; "value" can be strings, numbers or XML trees; "tag" elements are XML labels; "att" elements are attribute names; "doc" elements are URLs; and finally, *Op* elements can be selected from the usual binary operators: <=, >=, <, >, =, =/=, and from *OWL/RDF(S) built-in boolean operators*.
Basically, the XQuery language has been extended as follows:

- The **namespace** statement has been added allowing the declaration of URIs.
- A new kind of **for** expression has been added for traversing triples from a RDF document whose location is specified by means of the **rdfdocument** primitive; analogously, a new kind of **for** expression has been added for traversing triples from an OWL document whose location is specified by means of the **owldocument** primitive.
- In addition, the **where** construction includes boolean conditions of the form *Op* (*vexpr*, . . . , *vexpr*) which can be used for checking RDF(S)/OWL properties. The boolean operator *Op* can be one of *rdf:type*, *rdfs:subClassOf*, *owl:equivalentClass*, etc.

The above XQuery is a typed language in which there are two kinds of variables: those variables used in XPath expressions, and those used in RDF(S)/OWL triples. However they can be compared by means of boolean expressions, and they can be used together for the construction of the answer. We have considered a subset of the XQuery language in which some other built-in constructions for XPath can be added, and also it can be enriched with other XQuery

constructions, following the W3C recommendations [31]. However, with this small extension of the core of XQuery we are able to express interesting queries against XML, RDF and OWL documents.

Now, we would like to show an example of query in order to give us an idea about how the proposed extension of XQuery is suitable for OWL querying and reasoning. The query we like to show is *"Retrieve the authors of manuscripts"*. It can be expressed in our proposed extension of XQuery as follows:

```
namespace mns : "http://www.ontologies.org/manuscripts/#" in
< list > {
for  ($Author,$Property,$Manuscript) in owldocument(''ex.owl'') return
for  ($Manuscript,$Property2,$Type) in owldocument(''ex.owl'')
where rdfs:subPropertyOf($Property,mns:author_of)
and $Property2=rdf:typeOf and rdfs:subClassOf($Type,mns:manuscript) return
<author>{ $Author } </author >
} </ list >
```

In the example we can see the following elements:

- OWL triples are traversed by means of the new **for** expression of XQuery. Each triple is described by means of three variables (prefixed with '$' as usual in XQuery). Given that we have to query two properties, that is, *rdf:typeOf* and *author_of*, we have to combine two **for** expressions.
- The **where** expression has to check whether the first property, that is, *$Property*, is a subproperty of *author_of*, and the second property is *rdf:typeOf*. Let us remark that we could write *$Property=mns:author_of* instead of *rdfs:subPropertyOf($Property,mns:author_of)* but in such a case only triples "*author_of*" would be considered, and not subproperties of "*author_of*".
- The type of the manuscripts should be "*Manuscript*", as it is checked by means of *rdfs:subClassOf($Type,mns:manuscript)*.
- Finally, the output of the query is shown by means of XML in which each element (i.e. the author) is labeled by means of "*author*".

In this case the answer would be:

```
<list>
<author>Abiteboul</author>
<author>Suciu</author>
<author>Buneman</author>
<author>Buneman</author>
<author>Simeon</author>
</list>
```

Let us remark that our proposed query language is able to reason with OWL, that is, it uses that *Book* and *Paper* are subclasses of *Manuscript* and the relationship *author_of* is equivalent to *writes*.

4 Prolog Implementation

In this section, we will propose the implementation of the extension of XQuery in Prolog. Firstly, we will describe the encoding of the subset of OWL into Prolog. Secondy, we will show the encoding of XQuery in Prolog. The encoding of the extension of XQuery will use the encoding of XPath studied in [5], and the encoding of XQuery studied in [4].

```
triple(man,rdfs:subClassOf,person).
triple(woman,rdfs:subClassOf,person).
triple(inter([person,exists(author_of,manuscript)]),rdfs:subClassOf,writer).
triple(union([paper,book]),rdfs:subClassOf,manuscript).
triple(inter([book,hasvalue(topic,''XML'')]),rdfs:subClassOf,xmlbook).
triple(inter([manuscript,exists(reviewed_by,person)]),rdfs:subClassOf,reviewed).
triple(manuscript,rdfs:subClassOf,forall(rating,score)).
triple(manuscript,rdfs:subClassOf,forall(topic,topic)).
triple(author_of,owl:equivalentProperty,writes).
triple(authored_by,owl:equivalentProperty,inv(author_of)).
triple(average_rating,rdfs:subPropertyOf,rating).
triple(owl:thing,rdfs:subClassOf, forall(author_of, manuscript)).
triple(owl:thing,rdfs:subClassOf, forall(inv(author_of), person)).
triple(owl:thing, rdfs:subClassOf, forall(reviewed_by,person)).
triple(owl:thing, rdfs:subClassOf, forall(inv(reviewed_by),manuscript)).
triple(``Abiteboul'',rdf:type,man).
triple(``Buneman'',rdf:type,man).
triple(``Suciu'',rdf:type,man).
triple(``Data on the Web'',rdf:type,book).
triple(``XML in Scottland'',rdf:type,book).
triple(``Growing XQuery'',rdf:type,paper).
triple(``Anonymous'',rdf:type,person).
triple(``Abiteboul'',author_of,``Data on the Web'').
triple(``Data on the Web'',authored_by,``Buneman'').
triple(``Suciu'',author_of,``Data on the Web'').
triple(``Buneman'',author_of,``XML in Scottland'').
triple(``Simeon'',writes,``Growing XQuery'').
triple(``Data on the Web'',reviewed_by,``Anonymous'').
triple(``Growing XQuery'',reviewed_by,``Almendros'').
triple(``Data on the Web'',average_rating,``good'').
triple(``XML in Scottland'',rating,``excellent'').
triple(``Growing XQuery'',rating,``good'').
triple(``Data on the Web'',topic,``XML'').
triple(``Data on the Web'',topic,``Web'').
triple(``XML in Scottland'',topic,``XML'').
```

Fig. 5. Representation of Ontology instances

4.1 Ontology Encoding

The encoding of the ontologies defined in Section 2 consists of Prolog facts and rules. Facts are used for representing a given ontology instance. Rules are used for representing the ontology reasoning.

1. Ontology Instance Encoding: The encoding introduces Prolog facts about a predicate called *triple*. There is one fact for each element of the ontology instance. The encoding of DL formulas is as follows. Class and property names are represented by means of Prolog atoms. Quantified formulas are represented by means of Prolog terms, that is, $\forall P.C$, $\exists P.C$ and $\exists P.\{o\}$ are represented by means of Prolog term $forall(p, c)$, $exists(p, c)$ and $hasvalue(p, o)$, respectively. Unions (i.e. $C_1 \sqcup .... \sqcup C_n$) and intersections (i.e. $C_1 \sqcap ... \sqcap C_n$) are also represented as $union([C_1, \ldots, C_n])$ and $inter([C_1, \ldots, C_n])$, respectively. Inverse and transitivity properties (i.e. P^- and P^+) are represented as Prolog terms: $inv(P)$ and $trans(P)$. Finally OWL relationships: *rdfs:subClassOf*, *rdf:type*, etc are represented as atoms in Prolog. In the running example, we will have the facts of Figure 5.

2. OWL Reasoning Encoding: Now, the second element of the encoding consists of Prolog rules defining how to reason about OWL properties. Such rules express the semantic information deduced from a given ontology instance. Such

$fol^t(C \sqsubseteq D) = \forall x.fol^t_x(C) \rightarrow fol^t_x(D)$	
$fol^t(E \equiv F) = \forall x.fol^t_x(E) \leftrightarrow fol^t_x(F)$	$fol^t_x(A) = triple(x, rdf : type, A)$
$fol^t(P \sqsubseteq Q) = \forall x, y.triple(x, p, y) \rightarrow triple(x, q, y)$	$fol^t_x(C \sqcap D) = fol^t_x(C) \wedge fol^t_x(D)$
$fol^t(P \equiv Q) = \forall x, y.triple(x, p, y) \leftrightarrow triple(x, q, y)$	$fol^t_x(C \sqcup D) = fol^t_x(C) \vee fol^t_x(D)$
$fol^t(P \equiv Q^-) = \forall x, y.triple(x, p, y) \leftrightarrow triple(y, q, x)$	$fol^t_x(\exists P.C) = \exists y.triple(x, p, y) \wedge fol^t_y(C)$
$fol^t(P \equiv P^-) = \forall x, y.triple(x, p, y) \leftrightarrow triple(y, p, x)$	$fol^t_x(\forall P.C) = \forall y.triple(x, p, y) \rightarrow fol^t_y(C)$
$fol^t(P^+ \sqsubseteq P) = \forall x, y, z.triple(x, p, y) \wedge triple(y, p, z) \rightarrow triple(x, p, z)$	$fol^t_x(\forall P^-.C) = \forall y.triple(y, p, x) \rightarrow fol^t_y(C)$
$fol^t(\top \sqsubseteq \forall P.C) = \forall x.fol^t_x(\forall P.C)$	$fol^t_x(\exists P.\{o\}) = \exists y.triple(x, p, y) \wedge y = o$
$fol^t(\top \sqsubseteq \forall P^-.C) = \forall x.fol^t_x(\forall P^-.C)$	

Fig. 6. Triple-based encoding of DL in FOL

rules infer new relationships between the data in the form of triples. Therefore new Prolog terms make true the predicate *triple*. For instance, new triples for *rdf:type* are defined from the *rdfs:subClassOf* and previously inferred *rdf:type* relationships. In order to define the encoding, we have to follow the encoding of DL into FOL, which is described in Figure 6. Such encoding is based on a representation by means of triples of the DL formulas. In such encoding, class and property names are considered as constants in FOL. The encoding of OWL reasoning by means of rules is shown in Table 1.

The rules from **(Eq1)** to **(Eq4)** handle inference about equivalence. **(Eq1)** infers equivalence by reflexivity, **(Eq2)** infers equivalence by transitivity, and **(Eq3)** infers equivalence by symmetry. **(Eq4)** infers equivalence from the subclass relationship. The rules from **(Sub1)** to **(Sub13)** handle inference about subclasses. Cases from **(Sub3)** to **(Sub7)** define new subclass relationships from union and intersection operators. Cases from **(Sub8)** to **(Sub13)** define new subclass relationships for complex formulas. The rules **(Type1)** to **(Type7)** infer type relationships from subclass and equivalence relationships. The most relevant ones are the cases from **(Type2)** to **(Type5)** defining the meaning of complex formulas with regard to individuals. Finally, the rules **(Prop1)** to **(Prop11)** infer relationships about roles. The most relevant ones are the case **(Prop8)** and **(Prop9)** about the inverse of a property and the case **(Prop10)** about the transitivity relationship.

Our rules are able to infer new information from a given ontology. For instance, `triple(‘‘Data on the Web’’,rdf:type,reviewed)`, using the following **TBox** and **ABox** information:

```
triple(‘‘Data on the Web’’,rdf:type,book).
triple(book,rdfs:subClassOf,manuscript).
triple(‘‘Anonymous’’,rdf:type,person).
triple(‘‘Data on the Web’’,reviewed_by,‘Anonymous’’).
triple(inter([manuscript,exists(reviewed_by,person)]),rdfs:subClassOf, reviewed).
```

In order to avoid the looping of the rules, a bottom-up interpreter in Prolog has been implemented (for instance, the rule (**Sub2**) of Figure 1 loops in Prolog). Fortunately, the rules of Figure 1 satisfy a nice property. Applying the rules in a bottom-up fashion they compute a finite set of OWL relationships, that is, a finite set of triples. Therefore, the reasoning in the selected fragment of OWL is finite. It does not hold in general in description logic. Therefore, the implemented RDF(S)/OWL reasoner actually can be used for the pre-processing

Table 1. OWL reasoning rules

Rule Name	Prolog Rule
(**Eq1**)	triple(E, owl : equivalentClass, E) : −class(E).
(**Eq2**)	triple(E, owl : equivalentClass, G) : −triple(E, owl : equivalentClass, F), triple(F, owl : equivalentClass, G).
(**Eq3**)	triple(E, owl : equivalentClass, F) : −triple(F, owl : equivalentClass, E).
(**Eq4**)	triple(E, owl : equivalentClass, F) : −triple(E, rdfs : subClassOf, F), triple(F, rdfs : subClassOf, E).
(**Sub1**)	triple(E, rdfs : subClassOf, F) : −triple(E, owl : equivalentClass, F).
(**Sub2**)	triple(C, rdfs : subClassOf, E) : −triple(C, rdfs : subClassOf, D), triple(D, rdfs : subClassOf, E).
(**Sub3**)	triple(D, rdfs : subClassOf, E) : −triple(union(U), rdfs : subClassOf, E), member(D, U).
(**Sub4**)	triple(E, rdfs : subClassOf, C) : −triple(E, rdfs : subClassOf, inter(I)), member(C, I).
(**Sub5**)	triple(inter(I2), rdfs : subClassOf, D) : −triple(inter(I1), rdfs : subClassOf, D), member(C, I1), triple(E, rdfs : subClassOf, C), replace(I1, C, E, I2).
(**Sub6**)	triple(union(U2), rdfs : subClassOf, D) : −triple(union(U1), rdfs : subClassOf, D), member(C, U1), triple(E, rdfs : subClassOf, C), replace(U1, C, E, U2).
(**Sub7**)	triple(C, rdfs : subClassOf, inter(I2)) : −triple(C, rdfs : subClassOf, inter(I1)), member(D, I1), triple(D, rdfs : subClassOf, E), replace(I1, D, E, I2).
(**Sub8**)	triple(hasvalue(Q, O), rdfs : subClassOf, D) : −triple(Q, owl : subPropertyOf, P), triple(hasvalue(P, O), rdfs : subClassOf, D).
(**Sub9**)	triple(exists(Q, C), rdfs : subClassOf, D) : −triple(Q, owl : subPropertyOf, P), triple(exists(P, C), rdfs : subClassOf, D).
(**Sub10**)	triple(exists(P, E), rdfs : subClassOf, D) : −triple(E, rdfs : subClassOf, C), triple(exists(P, C), rdfs : subClassOf, D).
(**Sub11**)	triple(C, rdfs : subClassOf, hasvalue(Q, O)) : −triple(P, owl : subPropertyOf, Q), triple(C, rdfs : subClassOf, hasvalue(P, O)).
(**Sub12**)	triple(C, rdfs : subClassOf, forall(Q, D)) : −triple(Q, owl : subPropertyOf, P), triple(C, rdfs : subClassOf, forall(P, D)).
(**Sub13**)	triple(C, rdfs : subClassOf, forall(P, E)) : −triple(D, rdfs : subClassOf, E), triple(C, rdfs : subClassOf, forall(P, D)).
(**Type1**)	triple(A, rdf : type, D) : −triple(C, rdfs : subClassOf, D), triple(A, rdf : type, C).
(**Type2**)	triple(A, rdf : type, D) : −triple(inter(E), rdf : type, D), triple_cond(A, E).
(**Type3**)	triple(A, rdf : type, D) : −triple(exists(P, C), rdfs : subClassOf, D), triple(B, rdf : type, C), triple(A, P, B).
(**Type4**)	triple(A, rdf : type, D) : −triple(hasvalue(P, O), rdfs : subClassOf, D), triple(A, P, O), individual(O).
(**Type5**)	triple(B, rdf : type, D) : −triple(forall(P, C), rdfs : subClassOf, D), triple(A, P, B), triple(A, rdf : type, C).
(**Type6**)	triple(A, rdf : type, D) : −triple(forall(inv(P), C), rdfs : subClassOf, D), triple(A, P, B), triple(A, rdf : type, C).
(**Type7**)	triple(A, rdf : type, owl : thing) : −individual(A).
(**Prop1**)	triple(P, owl : equivalentProperty, P) : −property(P).
(**Prop2**)	triple(P, owl : equivalentProperty, R) : −triple(P, owl : equivalentProperty, Q), triple(Q, owl : equivalentProperty, R).
(**Prop3**)	triple(P, owl : equivalentProperty, Q) : −triple(Q, owl : equivalentProperty, P).
(**Prop4**)	triple(P, owl : equivalentProperty, Q) : −triple(P, rdfs : subPropertyOf, Q), triple(Q, rdfs : subPropertyOf, P).
(**Prop5**)	triple(P, rdfs : subPropertyOf, Q) : −triple(P, owl : equivalentProperty, Q).
(**Prop6**)	triple(P, rdfs : subPropertyOf, R) : −triple(P, rdfs : subPropertyOf, Q), triple(Q, rdfs : subPropertyOf, R).
(**Prop7**)	triple(A, Q, B) : −triple(P, rdfs : subPropertyOf, Q), triple(A, P, B).
(**Prop8**)	triple(B, Q, A) : −triple(P, rdfs : subPropertyOf, inv(Q)),triple(A, P, B).
(**Prop9**)	triple(B, P, A) : −triple(inv(Q), rdfs : subPropertyOf, P),triple(A, Q, B).
(**Prop10**)	triple(A, P, C) : −triple(trans(P), rdfs : subPropertyOf, P), triple(A, P, B), triple(B, P, C).
(**Prop11**)	triple(A, P, O) : −triple(C, rdfs : subClassOf, hasvalue(P, O)), triple(A, rdf : type, C).

```
xquery:= namespace name : resource in xquery | flwr | value
        | < tag att ='{'vexpr'}', ..., att ='{'vexpr'}'> xquery, ..., xquery < /tag >.
owldoc := owldocument(doc).          vexpr:= $var | value.
flwr:= for ($var,$var,$var) in owldoc [where constraint] return xqvar.
xqvar:= '{' vexpr '}' | < tag att ='{'vexpr'}', ..., att ='{'vexpr'}'> xqvar, ..., xqvar < /tag >
        | flwr | value.
constraint := Op(vexpr, ..., vexpr) | constraint 'or' constraint | constraint 'and' constraint.
```

Fig. 7. A Subset of the Core XQuery

OWL documents in order to be used in the solving of queries. It improves the efficience of the proposed language.

Now, we would like to show how the extended XQuery language can be encoded in Prolog, in such a way that the queries formulated in our proposed query language can be executed under Prolog. In order to make the paper self-contained we will restrict to a fragment of the extended XQuery for querying and reasoning with OWL triples and for generating XML documents as output. Now, the fragment to be encoded will consist in the grammar of the Figure 7, where *Op* can be one of the OWL built-in boolean operators. The reader can check that such fragment has been used in the example of the previous section. Now, a crucial point is the encoding of XML documents in Prolog. Such encoding will allow the encoding of the output of the query.

4.2 Encoding of XML Documents

In this section we will show an example of encoding of XML documents. A formal and complete definition can be found in [5]. Let us consider the following document called "b.xml":

```
<books>
<book year="2003">
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <title>Data on the Web</title>
<review>A <em>fine</em> book.</review>
</book>
<book year="2002">
    <author>Buneman</author>
    <title>XML in Scottland</title>
    <review><em>The <em>best</em> ever!</em></review>
</book>
</books>
```

Now, it can be represented by means of a Prolog program as in Figure 8. In our XML encoding we can distinguish the so-called *schema rules* which define the structure of the XML document, and *facts* which store the leaves of the XML tree (and therefore the values of the XML document). Each tag is translated into a predicate name: *books*, *book*, etc. Each predicate has four arguments. The first one, used for representing the XML document structure, is encapsulated into a function symbol with the same name as the tag adding the suffix *type*. Therefore, we have *bookstype*, *booktype*, etc. The second argument is used for numbering each node (a list of natural numbers identifying each node); the third argument of the predicates is used for numbering each type (a natural number identifying

Rules (Schema):

```
books(bookstype(Book, []), NBooks,1,Doc) :-
       book(Book, [NBook|NBooks],2,Doc).
book(booktype(Author, Title, Review,
        [year=Year]),NBook ,2,Doc) :-
       author(Author, [NAu|NBook],3,Doc),
       title(Title, [NTitle|NBook],3,Doc),
       review(Review, [NRe|NBook],3,Doc),
       year(Year, NBook,3,Doc).
review(reviewtype(Un,Em,[]),NReview,3,Doc):-
    unlabeled(Un,[NUn|NReview],4,Doc),
    em(Em,[NEm|NReview],4,Doc).
review(reviewtype(Em,[]),NReview,3,Doc):-
    em(Em,[NEm|NReview],5,Doc).
em(emtype(Unlabeled,Em,[]),NEms,5,Doc) :-
    unlabeled(Unlabeled,[NUn|NEms],6,Doc),
    em(Em, [NEm|NEms],6,Doc).
```

Facts (Document):

```
year('2003', [1, 1], 3,''b.xml").
author('Abiteboul', [1, 1, 1], 3,''b.xml").
author('Buneman', [2,1, 1], 3,''b.xml").
author('Suciu', [3,1,1], 3,''b.xml").
title('Data on the Web', [4, 1, 1], 3,''b.xml").
unlabeled('A', [1, 5, 1, 1], 4,''b.xml").
em('fine', [2, 5, 1, 1], 4,''b.xml").
unlabeled('book.', [3, 5, 1, 1], 4,''b.xml").
year('2002', [2, 1], 3,''b.xml").
author('Buneman', [1, 2, 1], 3,''b.xml").
title('XML in Scottland', [2, 2, 1], 3,''b.xml").
unlabeled('The', [1, 1, 3, 2, 1], 6,''b.xml").
em('best', [2, 1, 3, 2, 1], 6,''b.xml").
unlabeled('ever!', [3, 1, 3, 2, 1], 6,''b.xml").
```

Fig. 8. Encoding of XML documents

each type); and the last argument represents the document name. The key element of our encoding is to be able to recover the original XML document from the set of rules and facts. The encoding has the following peculiarities. In order to specify the order of an XML document in a fact based representation, each fact is numbered (from left to right and by levels in the XML tree). In addition, the hierarchical structure of the XML records is described by means of the identifier of each fact: the number of the children is a suffix of the number of the parent. The type number makes possible to map schema rules with facts.

4.3 Encoding of XQuery

Now, we will show how to encode the proposed extension of XQuery. We can summarize the encoding as follows. The encoding combines the schema rules of the output document with rules which call the *triple* predicate. A new predicate is defined called *join*. The join predicate calls to *triple* predicate in order to carry out the join of the triples. Now, let us see examples of queries in our proposal and the corresponding encoding.

Query 1: Let us suppose the query *"Retrieve the authors of manuscripts"* which is defined in XQuery as follows:

```
< list > {
for ($Author,$Property,$Manuscript) in owldocument(''ex.owl'') return
for ($Manuscript,$Property2,$Type) in owldocument(''ex.owl'')
where rdfs:subPropertyOf($Property,author_of)
and $Property2=rdf:typeOf and rdfs:subClassOf($Type,manuscript) return
<author>{ $Author } </author >
} </ list >
```

Now, the encoding is as follows:

```
(1) list(listtype(Author,[]),NList,1,Doc):-author(Author,[NAuthor|NList],2,Doc).
(2) author(authortype(Author,[]),NAuthor,2,''result.xml''):-join(Author,NAuthor).
(3) join(Author,[NAuthor,[1]]):-
                          triple(Author,Property,Manuscript,NAuthor,''ex.owl''),
                          triple(Manuscript,Property2,Type,_,''ex.owl''),
                          triple(Property,rdfs:subPropertyOf,author_of,_,''ex.owl''),
                          Property2=rdf:typeOf,
                          triple(Type,rdfs:subClassOf,manuscript,_,''ex.owl'').
```

The encoding takes into account the following elements.

- The **return** expression generates an XML document, and therefore the encoding includes the schema rules of the output document. In the previous example, this is the case of rule **(1)**, describing that the *list* tag includes elements labeled as *author*.
- The rules **(2)** and **(3)** are the main rules of the encoding, in which the elements of the output document are computed by means of the so-called *join* predicate.
- The *join* predicate is the responsible of the encoding of the **for** and **where** constructs. Each **for** associated to an OWL triple is encoded by means of a call to the *triple* predicate with variables.
- Now, the **where** expression is encoded as follows. In the case of binary operators like "=", ">", ">=", etc, they are encoded by means of calls to the corresponding Prolog operator. In the case of built-in boolean operators of the kind *rdf:typeOf*, *rdfs:subClassOf*, etc, a call to the *triple* predicate is achieved.
- Finally, we have to incorporate to the *triple* predicate two new arguments in facts and rules. The first new argument is a list of natural numbers identifying the triple. Each element of the **TBox** and the **ABox** is identified by means of [1], [2], etc. The triples inferred from the **TBox** and the **ABox** can be identified by appending the identifiers of the triples used for the inference. Therefore, in general, each triple can be identified as a list of natural numbers (for instance, $[1, 4, 5]$). Triple identifiers are required for representing the order of the elements of the output XML document, according to our encoding. The second new argument is the name of the document which stores the triple.

Now, the Prolog goal `?-author(Author,Node,Type,Doc)` has the following answers:

```
Author=authortype("Abiteboul",[]),Node=[...],Type=2,Doc=‘‘result.xml’’
Author=authortype("Suciu",[]),Node=[...],Type=2,Doc=‘‘result.xml’’
Author=authortype("Buneman",[]),Node=[...],Type=2,Doc=‘‘result.xml’’
Author=authortype("Buneman",[]),Node=[...],Type=2,Doc=‘‘result.xml’’
Author=authortype("Simeon",[]),Node=[...],Type=2,Doc=‘‘result.xml’’
```

From the schema rule (rule (**1**)), and these computed answers, we can rebuild the output XML document:

```
<list>
<author>Abiteboul</author>
<author>Suciu</author>
<author>Buneman</author>
<author>Buneman</author>
<author>Simeon</author>
</list>
```

Query 2: The second query we would like to show is *"Retrieve the reviewed manuscripts and their writers"*. In this case, the query can be expressed in XQuery as follows:

```
< list > {
for ($Writer,$Property,$Manuscript) in owldocument(‘‘ex.owl’’) return
for ($Manuscript,$Property2,$Type) in owldocument(‘‘ex.owl’’)
where rdfs:subPropertyOf($Property,writes)
and $Property2=rdf:typeOf and rdfs:subClassOf($Type,reviewed) return
<item>
<manuscript> { $Manuscript } </ manuscript >
<writer>{ $Writer } </writer >
</item>
} </ list >
```

In this case, the properties to be checked are "*writes*" and "*rdf:typeOf*". The boolean operator *rdfs : subPropertyOf ($Property, writes)* is also true for triples in which the *$Property* is "*author_of*" given that both are equivalent properties in the given ontology. A more concise way to express the previous query is:

```
< list > {
for ($Writer,$Property,$Manuscript) in owldocument(‘‘ex.owl’’)
where rdfs:subPropertyOf($Property,writes) and rdf:type(Manuscript,reviewed) return
<item>
<manuscript> { $Manuscript } </ manuscript >
<writer>{ $Writer } </writer >
</item>
} </ list >
```

It uses the built-in boolean operator *rdf:type* whose semantics takes into account the inclusion and equivalence relationships of the input ontology. In this case the answer would be:

```
<list>
<item><manuscript>Data on the Web</manuscript>
<writer>Abiteboul</writer>< /item>
<item><manuscript>Data on the Web</manuscript>
<writer>Suciu</writer>< /item>
<item><manuscript>Data on the Web</manuscript>
<writer>Buneman</writer>< /item>
<item><manuscript>XML in Scottland</manuscript>
<writer>Buneman</writer>< /item>
<item><manuscript>Growing XQuery</manuscript>
<writer>Simeon</writer>< /item>
</list>
```

Now, the goal is $? - \texttt{item}(\texttt{Item}, \texttt{Node}, \texttt{Type}, \texttt{Doc})$ and the encoding is as follows:

```
list(listtype(Item,[]),NL,1,Doc):- item(Item,[NItem|NL],2,Doc).
item(itemtype(manuscripttype(Manuscript,[]),writertype(Writer,[]),[]),NItem,2,‘‘result.xml’’):-
                              join(Manuscript,Writer,NItem).
join(Manuscript,Writer,[NM,[1]]):-
                              triple(Writer,Property,Manuscript,NM,‘‘ex.owl’’),
                              triple(Manuscript,Property2,Type,_,‘‘ex.owl’’),
                              triple(Property,rdfs:subPropertyOf,writes,_,‘‘ex.owl’’),
                              Property2=rdf:typeOf,
                              triple(Type,rdfs:subClassOf,reviewed,_,‘‘ex.owl’’).
```

Query 3: Finally, we would like to show the query *"Retrieve the equivalent properties of the ontology"*, in which we can extract from the input ontology some properties and we can represent the output of the query as an ontology:

```
< owl:Ontology > {
for ($Object1,$Property,$Object2) in owldocument(''ex.owl'')
where $Property=owl:equivalentProperty return
<owl:ObjectProperty rdf:about=''{$Object1}''/>
<owl:equivalentProperty rdf:resource=''{$Object2}''/>
</owl:ObjectProperty>
} </ owl:Ontology >
```

Now, the goal is $?-$ `owlObjectProperty(OwlObjectProperty,Node,Type,Doc)`, and the query is encoded as follows:

```
owlOntology(owlOntologytype(OwlObjectProp,[]),Nowl,1,Doc):-
      owlObjectProperty(OwlObjectProp,[Nowlo|Nowl],2,Doc).
owlObjectProperty(owlObjectPropertytype(owlequivalentPropertytype('''',[rdf:resource=Object2]),
       [rdf:about=Object1]),Nowlp,2,''result.xml''):-join(Object1,Object2,Nowlp).
join(Object1,Object2,[NTriple,[1]]):-
       triple(Object1,Property,Object2,NTriple,''ex.owl''),
       Property=owl:equivalentProperty.
```

5 Conclusions and Future Work

In this paper we have studied an extension of XQuery for the querying and reasoning with OWL style ontologies. Such extension combines RDF(S)/OWL and XML documents as input/output documents. By means of built-in boolean operators XQuery can be equipped with inference mechanism for OWL properties. We have also studied how to implement/encode such language in Prolog. We have developed an implementation of the XQuery language and a RDF(S) / OWL reasoner which can be downloaded from our Web site: `http://indalog.ual.es/XQuery` and `http://indalog.ual.es/OProlog`. More details about the implementation of the RDF(S)/OWL reasoner and the XQuery implementation can be found in [2,3]. As future work, we would like to extend our work with the handling of more complex ontologies in the line of [18,27]. In addition, we are now developing an implementation of the proposed extension as a XQuery library, using the available extension mechanisms of XQuery. We believe that it can be significant for the widespread acceptance of the approach.

Acknowledgements

The author would like to thank the anonymous referees for their insightful comments and suggestions. This work has been partially supported by the Spanish MICINN under grant TIN2008-06622-C03-03.

References

1. Almendros-Jiménez, J.M.: An RDF Query Language based on Logic Programming. Electronic Notes in Theoretical Computer Science 200(3), 67–85 (2008)
2. Almendros-Jiménez, J.M.: An encoding of xQuery in prolog. In: Bellahsène, Z., Hunt, E., Rys, M., Unland, R. (eds.) XSym 2009. LNCS, vol. 5679, pp. 145–155. Springer, Heidelberg (2009)

3. Almendros-Jiménez, J.M.: A Query Language for OWL based on Logic Programming. In: 5th Int'l Workshop on Automated Specification and Verification of Web Systems, WWv 2009, pp. 69–84 (2009)
4. Almendros-Jiménez, J.M., Becerra-Terón, A., Enciso-Baños, F.J.: Integrating XQuery and Logic Programming. In: Seipel, D., Hanus, M., Wolf, A. (eds.) INAP 2007. LNCS (LNAI), vol. 5437, pp. 117–135. Springer, Heidelberg (2009)
5. Almendros-Jiménez, J.M., Becerra-Terón, A., Enciso-Baños, F.J.: Querying XML documents in logic programming. TPLP 8(3), 323–361 (2008)
6. Baader, F., Horrocks, I., Sattler, U.: Description logics, pp. 3–28. Springer, Heidelberg (2004)
7. Bailey, J., Bry, F., Furche, T., Schaffert, S.: Web and Semantic Web Query Languages: A Survey. In: Eisinger, N., Małuszyński, J. (eds.) Reasoning Web. LNCS, vol. 3564, pp. 35–133. Springer, Heidelberg (2005)
8. Berners-Lee, T.: N3QL-RDF Data Query Language. Technical report, Online only (2004)
9. Boncz, P., Grust, T., van Keulen, M., Manegold, S., Rittinger, J., Teubner, J.: MonetDB/XQuery: a fast XQuery processor powered by a relational engine. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 479–490. ACM, New York (2006)
10. Boncz, P.A., Grust, T., van Keulen, M., Manegold, S., Rittinger, J., Teubner, J.: Pathfinder: XQuery - The Relational Way. In: Proc. of the International Conference on Very Large Databases, pp. 1322–1325. ACM Press, New York (2005)
11. Chamberlin, D., Draper, D., Fernández, M., Kay, M., Robie, J., Rys, M., Simeon, J., Tivy, J., Wadler, P.: XQuery from the Experts. Addison Wesley, Boston (2004)
12. Davis, I.: RDF Template Language 1.0. Technical report, Online only (September 2003)
13. de Bruijn, J., Lara, R., Polleres, A., Fensel, D.: OWL DL vs. OWL Flight: conceptual modeling and reasoning for the semantic Web. In: WWW 2005: Proceedings of the 14th International Conference on World Wide Web, pp. 623–632. ACM Press, New York (2005)
14. de Laborda, C.P., Conrad, S.: Bringing Relational Data into the Semantic Web using SPARQL and Relational OWL. In: Procs. of ICDEW 2006. IEEE Computer Society, Los Alamitos (2006)
15. Furche, T., Bry, F., Bolzer, O.: Marriages of Convenience: Triples and Graphs, RDF and XML in Web Querying. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 72–84. Springer, Heidelberg (2005)
16. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logic. In: Proc. of the International Conference on World Wide Web, USA, pp. 48–57. ACM Press, New York (2003)
17. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF query languages. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 502–517. Springer, Heidelberg (2004)
18. Hustadt, U., Motik, B., Sattler, U.: Reasoning in Description Logics by a Reduction to Disjunctive Datalog. J. Autom. Reasoning 39(3), 351–384 (2007)
19. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: a declarative query language for RDF. In: WWW 2002: Proceedings of the 11th International Conference on World Wide Web, pp. 592–603. ACM Press, New York (2002)
20. Katz, H.: XsRQL: an XQuery-style Query Language for RDF. Technical report, Online only (2004)

21. Marian, A., Simeon, J.: Projecting XML Documents. In: Proc. of International Conference on Very Large Databases, Burlington, USA, pp. 213–224. Morgan Kaufmann, San Francisco (2003)
22. Miller, L., Seaborne, A., Reggiori, A.: Three Implementations of SquishQL, a Simple RDF Query Language. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 423–435. Springer, Heidelberg (2002)
23. Robie, J., Garshol, L.M., Newcomb, S., Biezunski, M., Fuchs, M., Miller, L., Brickley, D., Christophides, V., Karvounarakis, G.: The Syntactic Web: Syntax and Semantics on the Web. Markup Languages: Theory & Practice 4(3), 411–440 (2002)
24. Schaffert, S., Bry, F.: A Gentle Introduction to Xcerpt, a Rule-based Query and Transformation Language for XML. In: Proc. of International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, Aachen, Germany. CEUR Workshop Proceedings, vol. 60, pages 22 (2002)
25. Sintek, M., Decker, S.: *TRIPLE*–A Query, Inference, and Transformation Language for the Semantic Web. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 364–378. Springer, Heidelberg (2002)
26. Souzis, A.: RxPath: a mapping of RDF to the XPath Data Model. In: Extreme Markup Languages (2006)
27. Volz, R.: Web Ontology Reasoning with Logic Databases. PhD thesis, Universität Fridericiana zu Karlsruhe (2004)
28. W3C. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report (2004), `www.w3.org`
29. W3C. Resource Description Framework (RDF). Technical report (2004), `www.w3.org`
30. W3C. XML Path Language (XPath) 2.0. Technical report (2007), `www.w3.org`
31. W3C. XML Query Working Group and XSL Working Group, XQuery 1.0: An XML Query Language. Technical report (2007), `www.w3.org`
32. W3C. OWL 2 Web Ontology Language. Technical report (2008), `www.w3.org`
33. Walsh, N.: RDF Twig: Accessing RDF Graphs in XSLT. In: Proceedings of Extreme Markup Languages (2003)