Querying the Semantic Web of Data using SPARQL, RDF and XML

Olivier Corby, Leila Kefi-Khelif, Hacène Cherfi, Fabien Gandon, Khaled Khelif

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Querying the Semantic Web of Data using SPARQL, RDF and XML

Olivier Corby, Leila Kefi-Khelif, Hacène Cherfi, Fabien Gandon and Khaled Khelif

N° 6847

February 2009

Thème COG

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6847--FR+ENG

Querying the Semantic Web of Data using SPARQL, RDF and XML

Olivier Corby, Leila Kefi-Khelif, Hacène Cherfi, Fabien Gandon and Khaled Khelif

Thème COG — Systèmes cognitifs
Équipe-Projet Edelweiss

Rapport de recherche n° 6847 — February 2009 — 22 pages

Abstract: The Semantic Web relies on two layers: XML and RDF. XML documents are merely trees representing structured data or documents and accessed using XPath. RDF is used for a Web of data and to provide metadata about data or documents; it is organized as graphs made of collections of elementary triples and can be queried using SPARQL. Based on these two paradigms, there exist tools and platforms that produce and process both XML and RDF. When doing information integration and mash-up applications, there are scenarios where we need to query, compare and integrate data coming from both worlds. In this report we present a seamless way of mixing both paradigms in SPARQL. Generic extensions to SPARQL are explained, and then we provide use cases and an application in semantic annotation of textual documents using NLP techniques.

Key-words: Semantic Web, Web of Data, SPARQL, RDF, XPath, XML

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Interroger le Web sémantique de données avec SPARQL, RDF et XML

Résumé : Le Web sémantique repose sur deux niveaux : XML et RDF. Les documents XML peuvent être vus comme des arbres qui représentent des données et des documents structurés pouvant être accédés par XPath. RDF est utilisé sur le Web sémantique pour exprimer des métadonnées sur des documents et des données, RDF est organisé en graphes formés à partir de collections de triplets élémentaires qui peuvent être interrogés avec SPARQL. Des outils et des plate-formes basés sur ces deux paradigmes produisent et consomment des énoncés en RDF et en XML. Ainsi, dans les applications qui intègrent ces deux formalismes, il peut être nécessaire d'interroger et de comparer des données RDF et XML. Dans ce rapport nous présentons une manière d'intégrer aisément les deux paradigmes dans SPARQL. Des extensions génériques de SPARQL sont proposées et nous décrivons des cas d'utilisation ainsi qu'une application pour la production d'annotations sémantiques à partir de texte.

Mots-clés : Web Sémantique, Web de données, SPARQL, RDF, XPath, XML

1 Introduction

The Semantic Web relies on two layers: XML (Extensible Markup Language) and RDF (Resource Description Framework). XML [BPSM98] is a meta language that enables us to describe structured documents and structured data as (abstract syntax) trees. The XML tree model is standardized as well as its API and access languages (e.g. DOM, XPath). XPath [CD99] enables us to query an XML document by means of paths in its tree structure. It is designed to be used by other languages such as XSLT or XQuery but also by any framework that has to point or access to an XML tree. XSLT [Cla99] is the Extensible Stylesheet Transformation Language. It enables to transform an XML document into another one following transformation rules.

RDF [MMM04] is used for a Web of data and to provide metadata about data or documents. It is a triple language with a graph model and an XML syntax. SPARQL [PS08] is a query language that enables us to query RDF graphs. SPARQL borrows some statements from XPath such as regular expressions syntax, casting functions, etc.

Nowadays, there are tools and platforms that *produce and consume XML and RDF*. Hence, there are scenarios, for instance in data integration and mash-up applications, where we need to query both formalisms, compare and integrate values coming from both worlds. This work is about integrating these two paradigms within SPARQL. Furthermore, we show that our approach enables to integrate other languages such as XSLT, SQL and nested SPARQL.

We have identified several use cases for mixing RDF and XML processing. The generic case can be summarized as *integrating RDF graph-oriented and XML tree-oriented data processing*.

1.1 Use cases

RDF enables to embed *structured data* as `rdf:XMLLiteral` datatype values within RDF triples. However, it must be noted that there is no mean to query the content of such nested structured data using SPARQL (except using string operations). Use cases can come from including fraction of XML documents that are annotated. It can also come from legacy XML format such as STEP/Express [BLP00] in product engineering. We do not need to translate everything into RDF when data are available in XML.

However, we would like to access the content of such XML data, compare values coming from XML with values coming from RDF. For example, given a resource found in RDF, we may want to retrieve additional features in the XML description. We may also want to compare XML Literal values.

Another use case may be to *construct* RDF triples with values coming from such XML documents using the SPARQL construct clause.

Applications may store XPath expressions in RDF metadata, pointing to part of the XML document being annotated. We may want to access the content of the part beeing annotated to perform additional search or to show it in a user interface.

1.2 Challenge

In order to process XML documents within SPARQL, some authors propose to translate XML into RDF. Others propose to integrate XQuery and SPARQL to query both models, see section 7 on related work below. Our position is to investigate whether it is possible to query XML data within the SPARQL framework with a minimum of changes to the language.

Surprisingly, the answer is yes. SPARQL enables to query and compare XML data with RDF data with two extensions: a new datatype and a new function. The new function matches an XPath expression on an XML Literal. As XPath evaluation returns a node-set, we need a new datatype that implements the node-set within SPARQL filter evaluation framework. We call this new datatype a sequence. In this paper we show that these two notions enable to query XML data.

Interestingly, the generic sequence datatype enables us to implement two more extensions following the same design pattern: nested SPARQL queries and nested SQL queries. Hence we are able to process legacy data coming from XML documents and relational database within SPARQL.

To our knowledge, we are the first to propose and implement this solution within SPARQL and it is the simplest solution that have been proposed so far. Related works are presented at the end of the paper. We show that this extension fits perfectly within SPARQL framework and that it does not change the semantics of graph pattern matching.

2 SPARQL Extension

In this section we first present two generic extensions to SPARQL that are of particular interest for XML integration within RDF and that will be used in this paper.

2.1 Select expression

Our SPARQL engine [CFZ07] enables to return evaluable expressions in the select clause. The syntax is shown below where `Constraint` is an evaluable expression and `Var` is a variable as defined in the SPARQL grammar[1]:

```
'select' ( Constraint 'as' Var ) *
```

Example:

```
select *  size(?doc) as ?size
where {
  ?x :author ?doc
}
```

Such a variable, e.g. `size` , is a name used to identify the result returned by the query. Mutually recursive definitions of variables are forbidden. In addition and as syntactic sugar, such a variable from the select clause can be used in the filters of the query, as well as in the `order by` clause. The variable is statically

[1] `http://www.w3.org/TR/rdf-sparql-query#rConstraint`

rewritten into the expression defined by the select clause. The rewriting takes place in the select clauses, in the filters and in the order by (i.e. not in the triples). Hence, this extension is syntactic and it does not modify the semantics of SPARQL. The example below shows such a variable, `size` (2), that is reused in a filter (5):

```
(1) select *
(2)   size(?doc) as ?size
(3) where {
(4)   ?x :author ?doc
(5)   filter(?size > 100000)
(6) }
```

The query above is rewritten as shown below (5).

```
(1) select *
(2)   size(?doc) as ?size
(3) where {
(4)   ?x :author ?doc
(5)   filter(size(?doc) > 100000)
(6) }
```

2.2 Construct with select

In addition to evaluable expressions in the `select` clause, our SPARQL implementation allows us to use a `construct` clause together with a `select` clause. Hence, it is possible to construct triples with resources computed by a function call. In the example below, we construct a triple (2) with a value that is computed by a function call (5).

```
(1) construct {
(2)   ?doc :size ?size
(3) }
(4) select
(5)   size(?doc) as ?size
(6) where {
(7)   ?x :author ?doc
(8)   filter(?size > 100000)
(9) }
```

3 XPath in SPARQL

The SPARQL query language enables to query RDF metadata by performing graph match and filter test. In the example below, we search for a resource where the *phrase* property (3) contains the *proteins* string (4).

```
(1) select * where {
(2)   ?x :author ?doc
(3)   ?doc :phrase ?phrase
(4)   filter(regex(?phrase, 'proteins'))
(5) }
```

In the example above, we suppose that all information is available in RDF. But it may happen that some information be accessible in XML document. For example, it may happen that RDF annotates the URI of the document and that additional informations be within the document. How can we access to the content of the XML document once we have its URI? In SPARQL, it is not possible to perform such a simple processing.

We propose to solve this problem with a generic solution. We introduce a new function `xpath` in SPARQL with two arguments. The first argument is the URL of an XML document (or an `rdf:XMLLiteral` datatype value as we shall see later). The second argument is an XPath expression. The purpose of this function is to evaluate the XPath expression against the content of the XML document (or XML Literal).

Throughout this paper, we use an example in text mining where we manipulate the output of an NLP tool that generates an abstract syntax tree in XML. We show examples of query that explore this XML document.

```
<phrase>
  <subject>
    <label>proteins</label>
    <mod><label>Wnt</label></mod>
  </subject>
  <verb>
    <label>play</label>
  </verb>
  <object>
    <label>roles</label>
    <mod>
         <label>important</label>
    </mod>
    <mod>
         <label>in</label>
         <object>
           <label>development</label>
           <mod>
             <label>cartilage</label>
           </mod>
         </object>
    </mod>
  </object>
</phrase>
```

In the example below, we retrieve the URI of documents and we test the content of the XML documents. We test whether the word *proteins* is present somewhere in the XML tree. As we can see, words are stored within `label` tags, hence we write an XPath expression that walks through all `label` tag text value in the XML tree: `/phrase//label/text()`. The result of the `xpath()` is eventually coerced to a Boolean value indicating whether it succeeded. The query (1) returns resources that (2) are author of documents that (3) contain an occurrence of the *proteins* string within their XML markup.

```
(1) select * where {
(2)   ?x :author ?doc
(3)  filter(xpath(?doc, "/phrase//label/text()") = 'proteins')
(4)  }
```

Using the extension presented above (function in select), we can return as result of the query some information from the XML document as shown below. The query returns as result, line (2), the verb of the phrase.

```
(1) select *
(2)   xpath(?doc, "/phrase/verb/label/text()") as ?res
(3) where {
(4)   ?x :author ?doc
(5)   filter(xpath(?doc, "/phrase//label/text()") = 'proteins')
(6) }
```

Surprisingly, we see that (almost) all that is needed is a function, `xpath()`, to query XML documents within SPARQL. In fact, something else is needed because some XPath expressions may return more than one result, i.e. a node-set. This is solved by adding a new datatype, within SPARQL datatype framework, that implements node-sets, this datatype is called a sequence.

Definition: a sequence is an ordered collection of datatype values or of sequences.

As the definition is recursive, we can have sequences of sequences. We have extended the SPARQL filter evaluator in the spirit of XPath, in order to process sequence of values, see section 5.1. When evaluating an expression, if one value to be compared is a sequence of values, then the comparison will be true if and only if there is a value in the sequence such that the result of performing the comparison on the two values is true, using SPARQL standard evaluation rules.

It must be noted that the sequence datatype has a special status as it can appear only within filters, as a result of a function call, e.g. `xpath()`, (or within a `select` expression). It cannot appear within a triple, hence SPARQL graph match semantics is not impacted by this extension.

Hence, we can now express our claim: *an `xpath()` function and a `sequence` datatype suffice to query XML within SPARQL using XPath.*

3.1 Sharing namespaces and variables

Namespace declarations are propagated from the SPARQL query to the XPath processor. SPARQL query variables are also available in the XPath evaluation. In the query below, `key` is the same variable in SPARQL and XPath. The query retrieves documents where (6) one of the labels in the content has the same value as (5) the `keyword` RDF property of the document. The SPARQL variable value is casted as a string within the XPath processor.

```
(1) prefix ns: <http://example.org/namespace#>
(2) select *
(3) where {
(4)   ?x :author ?doc
(5)   ?doc :keyword ?key
```

```
(6)   filter(xpath(?doc, "/ns:phrase//*[ns:label = $key]"))
(7) }
```

We now present some use cases and some extensions in the same spirit.

3.2 XPath expression in RDF metadata

RDF metadata may contain XPath expressions that point to a targeted subpart of the document that is annotated (e.g. first chapter, second section, third paragraph). A convention would be needed to identify such embedded expressions, e.g. a specific property. A SPARQL query can retrieve such a metadata with its associated XPath expression and it is then possible to evaluate the XPath expression on the fly as shown below. First we show an RDF metadata containing an XPath expression, line (5).

```
(1) <Person>
(2)   <author>
(3)     <Document rdf:about='http://example.org#doc'>
(4)       <title>...</title>
(5)       <expression>/phrase/subject/label</expression>
(6)     </Document>
(7)   </author>
(8) </Person>
```

In the query below, we evaluate the XPath expression, (3) associated with a metadata, by means (4) of the variable `exp`. In addition, we use the result of the evaluation of the path in a filter and we test if it contains an occurrence of a string using a regular expression.

```
(1) select * where {
(2)   ?x :author ?doc
(3)   ?doc :expression ?exp
(4)   filter(regex(xpath(?doc, ?exp), 'protein'))
(5) }
```

3.3 Query Embedded XMLLiteral

Until now, we have shown examples where we test an XPath expression on an XML document given its URL. But RDF enables us to embed XML fragments[2] into the RDF graph by means of `rdf:XMLLiteral` datatype values and `Literal parseType` syntax as shown below.

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<Person>
<author rdf:parseType='Literal'>
<phrase>
  <subject>
    <label>proteins</label>
    <mod><label>Wnt</label></mod>
```

[2] `http://www.w3.org/TR/REC-rdf-syntax#xmlliterals`

```
  </subject>
  ...
</phrase>
</author>
</Person>
</rdf:RDF>
```

Hence, it is natural to query such an XML fragment that is embedded in the RDF graph. In itself, it is a sufficient justification to use XPath in SPARQL. In the query below, `?doc` is an `rdf:XMLLiteral` datatype value, line (3).

```
(1) select * where {
(2)   ?x :author ?doc
(3)   filter(xpath(?doc, "/phrase//*[label = 'proteins']"))
(4) }
```

This may be useful for querying an extract of an annotated document that was included in the metadata. It may also be useful to query additional *structured* data that does not fit well in the RDF triple formalism and that is embedded as XML markup, e.g. STEP/Express, MathML, etc.

3.4 Query RDFa

XPath can be used to query RDFa [AB08] statements embedded in XHTML documents. It could also be used to query microformats such as calendar or vcard.

```
select * where {
  ?x :author ?doc
  filter(xpath(?doc, "//*[@about and @rel and @href]"))
}
```

3.5 Select expression

Using evaluable expressions in the select clause, as shown above, enables us to identify and isolate part of an XML DOM tree using XPath. The result of the XPath evaluation is stored in a transient value of datatype `xsd:string` in case of an XML text node and of datatype `rdf:XMLLiteral` in case of an XML element. In this case, the DOM sub-tree is stored in the XML Literal. It is then possible to evaluate further XPath expressions against this XML Literal. Hence we can emulate the access to an object in an XML document and then access to the object's properties. This is particularly useful when there are other XML formats than RDF in a Semantic Web application, e.g. XML STEP/Express in product engineering. In the example below, we isolate an object, `phrase` (2), and access its properties: `subject` (3) and `object` (4).

```
(1) select *
(2)   xpath(?doc,    "/phrase") as ?phrase
(3)   xpath(?phrase, "subject") as ?subject
(4)   xpath(?phrase, "object")  as ?object
(5) where {
(6)   ?x :author ?doc
(7) }
```

3.6 Construct using XPath

In addition to evaluable expressions in the `select` clause, our SPARQL implementation allows us to use a `construct` clause together with a `select` clause. It is possible to construct triples with resources computed by a function call, and hence by an XPath. We take into account sequences of values returned by XPath and then may create several triples accordingly. In the example below, we construct a triple (2) with a value computed by an xpath function call (5), hence with a value coming from an XML document.

```
(1) construct {
(2)   ?doc :title ?text
(3) }
(4) select *
(5)   xpath(?doc, "/phrase/subject/label") as ?text
(6) where {
(7)   ?x :author ?doc
(8) }
```

If the XPath expression returns a sequence of datatype values, we construct one triple for each value in the sequence. If a triple is constructed with several arguments that are sequences (e.g. subject and object) we construct one triple for each tuple of values obtained by enumerating the sequences.

3.7 XPointer

In addition to embedding XPath expressions in RDF, we interpret XPointers embedded in RDF using an `xpointer()` function. We can retrieve the portion of the document using an xpointer function call as shown below. The `xpointer` function (11) extracts the XPath expression from the URI, here `/phrase/subject/label` (05), and interprets the path with the usual `xpath` function.

```
(01) <Person>
(02)   <author>
(04)     <Statement rdf:about=
(05)       'http://example.org/doc#xpointer(/phrase/subject/label)'>
(06)     ...
(07)     </Statement>
(08)   </author>
(09) </Person>

(10) select *
(11)   xpointer(?doc) as ?elem
(12) where {
(13)  ?x :author ?doc
(14) }
```

4 Processing other languages

In this section we show that the principles presented above enable to integrate smoothly other languages.

4.1 SPARQL and XSLT

Using the same design pattern as XPath, we provide an `xslt` function (2) to process XML documents with XSLT stylesheets. It can be used as preprocessing for further XPath interrogation (3) as shown in the example below.

```
(1) select *
(2)   xslt(?doc, "style.xsl") as ?tree
(3)   xpath(?tree, "//title/text()") as ?title
(4) where {
(5)   ?x :author ?doc
(6) }
```

A generic use case consists in using an RDFa extractor stylesheet[3] to extract RDF triples from an XHTML document *within* SPARQL. An even more generic use case consists in getting the XLST transformation by means of a GRDDL [Gan07] declaration as shown below. First we show a document with a GRDDL profile (02) and a link to a transformation (04, 05). Then we show a query that retrieves the transformation (09) and applies it (10):

```
(01) <html xmlns="http://www.w3.org/1999/xhtml">
(02) <head profile="http://www.w3.org/2003/g/data-view">
(03) <title>Commentarii de Bello Gallico</title>
(04) <link rel="transformation" href=
(05)   "http://ns.inria.fr/grddl/rdfa/2008/05/05/RDFa2RDFXML.xsl"/>
(06) </head>
     ...
(07) </html>

(08) select *
(09)   xpath(?doc, "//link[@rel='transformation']/@href") as ?xsl
(10)   xslt(?doc, ?xsl) as ?rdf
(11) where {
(12) ?x :author ?doc
(13) filter(xpath(?doc,
(14)   "/html/head[@profile = 'http://www.w3.org/2003/g/data-view']"))
(15) }
```

4.2 XSLT and SPARQL

In addition, our `xslt` function that runs an XSLT stylesheet enables this stylesheet to evaluate a SPARQL query, and integrate the result of the query as a DOM tree within the stylesheet processing. Hence, it is possible to use XPath and XSLT within SPARQL and use SPARQL within XSLT. An example of such transformation is shown below, with a call to the `server:sparql` function (6) inside `xslt`:

```
(1) <xsl:param name='engine' />

(2) <xsl:variable name='query'>
```

[3] `http://ns.inria.fr/grddl/rdfa`

```
(3) select * where {?x ?p ?y}
(4) </xsl:variable>

(5) <xsl:variable name='res'
(6)   select='server:sparql($engine, $query)'/>

(7) <xsl:apply-templates select='$res' />
```

4.3 Nested SPARQL

We have designed a function that enables us to compute a nested query inside a SPARQL query. The result is a sequence of values, like for XPath. The result is cached in order to optimize query processing, i.e. it is computed once. In the query below, we compute dynamically a collection of graph URI using the nested `sparql` function (2) that is bound to variable `s`:

```
(1) select *
(2)   sparql("select ?s where { PAT1 }") as ?s
(3) where {
(4)   graph ?g { PAT2 } . filter(?g = ?s)
(5) }
```

A use case for nested queries consists in storing SPARQL queries into RDF metadata, retrieving them using a pattern and computing the queries on the fly. Embedded queries may represent validity conditions for metadata or predefined queries, stored in RDF, that are part of a Semantic Web application. Note that SPARQL queries may also be stored in XML documents and retrieved using XPath. The query below shows the retrieval of a query (2) from RDF and its execution (3).

```
(1) select * where {
(2)   ?x :query ?query
(3)   filter(sparql(?query))
(4) }
```

4.4 Nested SQL

We have seen that we can retrieve information from XML documents. We show now that, using the same design pattern, we can retrieve data from relational database using SQL. We have designed an extension that enables to compute a nested SQL query within a SPARQL query. This is done by means of a `sql()` function that returns a sequence of sequence results, one sequence for each variable of the SQL select clause. We have designed a syntactic extension to SPARQL select clause that enables to bind the result sequence elements to a list of variables. In the example below (6), variable *gene* is bound to the sequence of values of the first variable of the SQL query and variable *value* is bound to the sequence of values of the second variable.

```
(1) prefix db: &lt;jdbc:derby://localhost:1527/&gt;
(2) select  *
(3)   sql(db:DBTest,
```

```
(4)     "SELECT distinct  gene, value FROM EXPVALUES
(5)      WHERE value < 100 AND gene < 'g10'")
(6)   as (?gene,  ?value)
(7) where {
(8)   ?g :value ?val
(9)   filter(?val = ?value)
(10) }
```

In addition, it is possible to construct RDF triples with data coming from a relational database by using `construct` (02) with `sql` (06) as shown below.

```
(01) prefix db: &lt;jdbc:derby://localhost:1527/&gt;
(02) construct {
(03)   ?gene :reference ?value
(04) }
(05) select  *
(06)   sql(db:DBTest,
(07)     "SELECT distinct  gene, value FROM EXPVALUES
(08)      WHERE value < 100 AND gene < 'g10'")
(09)   as (?gene,  ?value)
(10) where {
(11)   ?g :value ?val
(12)   filter(?val = ?value)
(13) }
```

5 Implementation

We present the implementation of this work within the Corese Semantic Web Factory.

5.1 SPARQL Filter evaluator

The XPath function returns a sequence of values corresponding to the list of nodes found in the XML document. It is then natural to compare SPARQL expressions with XPath expressions in filters. But as XPath expressions return sequences of values, it is necessary to define new evaluation rules for filters.

To handle sequences of values, we have introduced a new datatype, in addition to standard SPARQL datatypes, which is a sequence of datatype values. Currently, our xpath implementation returns sequences of `xsd:string` as well as `rdf:XMLLiteral`. However, we may extend xpath to return other datatypes such as integers, dates, etc.

The SPARQL filter evaluator has been extended in the spirit of XPath. When evaluating an expression, if both values to be compared are sequences of values, then the comparison will be true if and only if there is a value in the first sequence and a value in the second sequence such that the result of performing the comparison on the two values is true using SPARQL standard evaluation rules. If one value to be compared is a sequence and the other is a single value, then the comparison will be true if and only if there is a value in the sequence such that the result of performing the comparison on the two values is true.

The same rule applies with Boolean functions that are sequence aware. A function applied on a sequence of values evaluates to true if and only if there is a value in the sequence such that the function evaluates to true when applied on that value.

When applying a sequence aware function that is not Boolean to a sequence of values, the function is applied to every elements of the sequence and the result is a new sequence built with the result values in the same order. Sequence aware operations currently are:

- Operators: `= != < <= >= >`
- Boolean functions: `regex, sameTerm`
- Functions: `uri, cast (e.g. xsd:integer)`

The values returned by the `xpath` function currently are of `xsd:string` or `rdf:XMLLiteral` datatypes and we can cast them using sequence aware cast functions, e.g. `xsd:integer`, `xsd:date`.

It must be noted that this extension does not change the semantics of SPARQL graph match because it is realized by means of a new datatype (i.e. sequence of datatype values) that appears only within filter evaluation. The sequence datatype does not appear within triples in the body of the query.

5.2 Corese Implementation

The implementation is done within the Corese[4] Semantic Factory [CFZ07, Cor08]. Corese is an implementation of RDF, RDFS, part of OWL Lite, SPARQL and RDF Rules, based on the Conceptual Graphs paradigm [Sow84, CM92]. It may be used to build Semantic Web Servers using the Sewese/Semtags factory[5]. These factories have been used to design and develop the SweetWiki[6] Semantic Wiki and more than 20 applications.

We use the Java 1.6 XPath[7] (XPath 1.0) and XSLT[8] packages as well as the Apache Derby database[9] that is available with Java 1.6.

The parsing of XML documents as DOM and the evaluation of XPath are time consuming. Hence, we have designed a cache in order to store the DOM tree of parsed documents during SPARQL query evaluation. In addition, we cache the evaluation of XPath expressions in such a way that two successive evaluations of the same expression on the same document return the same result without recomputing it. A similar cache is available for nested SPARQL and SQL queries.

All that is presented in this paper is running within Corese and we now present an application that uses these extensions.

[4] `http://www.inria.fr/sophia/edelweiss/software/corese`
[5] `http://www.inria.fr/sophia/edelweiss/wiki/wakka.php?wiki=Sewese`
[6] `http://www.inria.fr/sophia/edelweiss/wiki/wakka.php?wiki=SweetWiki`
[7] `javax.xml.xpath`
[8] `javax.xml.transform`
[9] `http://db.apache.org/derby`

6 Application

In this section we present an application that is carried out in the framework of the BioMarker[10] collaborative project with biologists. The aim of this project is to support biologists in the validation of their experiments using Semantic Web technologies and especially semantic annotations generated from texts. In fact, a major need of biologists is to access knowledge described in natural language texts. This knowledge is essential for checking, validating and enriching new research work. Mining this literature is one way to detect relevant information and generate semantic annotations on documents in order to facilitate their search. Our aim is to generate a structured annotation based on UMLS (Unified Medical Language System) [HL93] that describes the semantic content of a text, analyzing the content of the textual document (i.e. a scientific paper). This annotation describes interactions between genes/proteins and other UMLS concepts.
The generation of such annotations is decomposed into two steps described below. We will focus on the second step which uses our extensions.

6.1 Relations and concepts detection

We used MeatAnnot [KDKB07] to detect instances of UMLS concepts and relations in a text. MeatAnnot uses NLP (Natural Language Processing) tools GATE [CMBT02], TreeTagger [Sch94] and developed extensions dedicated to extraction of semantic relations and concepts describing the biomedical domain. These extensions use (i) JAPE [CMBT02], a language based on regular expressions, to write grammars allowing to extract all instances of each relation (such as play_role, expressed_in, disrupt, etc.) and (ii) the UMLS Knowledge Server to detect instances of concepts (biological terms such as gene names, diseases, etc. and information about these terms).

6.2 Annotation generation

The purpose here is to link each detected relationship with the appropriate instances of detected UMLS concepts and to generate an RDF annotation. We used the result of the RASP [BCW06] NLP tool applied to each sentence of the document, combined to the instances of UMLS relationships and concepts detected on these sentences. RASP is a domain-independent, robust parsing system for English. We build XML trees which are abstract syntax parse trees coming from transformations of the RASP/MeatAnnot results. Each XML tree is associated with a verbal group of the sentence.

For each relationship defined in our ontology, we add information about its grammatical characteristics in texts (subjects, verbs, adjectives, etc.).

The example shown below is an RDF document that embeds XML fragments. It is obtained by an XSL transformation that simplifies, restructures and renames a RASP result and integrates instances and properties identified by the MeatAnnot NLP tool. The RDF/XML document is the result obtained after processing the sentence shown below:

Wnt proteins play important roles in cartilage development.

[10] `http://www.inria.fr/sophia/edelweiss/wiki/wakka.php?wiki=Projects`

```
<rt:Document>
<rt:has_phrase rdf:parseType="Resource">
<rt:label>Wnt proteins play important roles
in cartilage development.
</rt:label>

<!-- Linguistic analysis by RASP -->
<rt:tags rdf:parseType="Literal">
<phrase>
  <subject>
    <label>proteins</label>
    <mod><label>Wnt</label></mod>
  </subject>
  <verb>
    <label>play</label>
  </verb>
  <object>
    <label>roles</label>
    <mod>
         <label>important</label>
    </mod>
    <mod>
         <label>in</label>
         <object>
           <label>development</label>
           <mod>
             <label>cartilage</label>
           </mod>
         </object>
    </mod>
  </object>
</phrase>
</rt:tags>
</rt:has_phrase>
</rt:Document>
```

The query below, when applied to the RDF/XML document presented above, detects which instances of UMLS concepts are linked by a relationship and it generates an annotation describing this interaction. In this case, two points can justify mixing SPARQL/XPath and RDF/XML:

1. consulting the ontology to select the property according to its description (the property play_role according to the verb play).
2. checking that instances (the subject-concept sc) exist with the right type according to the property signature (the domain of the relationship in this example).

```
prefix rt:  <http://ns.inria.fr/edelweiss/rasptag#>
prefix fun: <function://corese.functions.Fun>
```

```
construct {
  ?sc ?pr ?oc
}
where {
   ?doc rt:has_phrase ?ph
   ?ph  rt:tags ?tree

   <!-- subject-concept detection-->
   ?sc rdf:type ?sctype
   ?sc rt:label ?sl

filter (fun:match(?sl,
        xpath(?tree, "/phrase/subject//label/text()")))

   <!-- object-concept detection-->
   ?oc rt:label ?ol

filter (fun:match(?ol,
        xpath(?tree, "/phrase/object//label/text()")))

   <!-- relationship detection-->
   ?pr rdfs:domain ?sctype
   ?pr rt:is_defined_by [ rt:verb ?vl ]

filter (fun:match(?vl,
        xpath(?tree, "/phrase/verb//label/text()")))
}
```

The function `fun:match(word, listOfWords)` returns true if all the terms composing the string `word` exist in the sequence of values `listOfWords`. The generated annotation is:

```
<umls:Organ_or_Tissue_Function rdf:about=
   'http://ns.inria.fr/bio/cart_dev'/>

<umls:Gene_or_Protein rdf:about=
  'http://ns.inria.fr/bio/Wnt_protein'>
   <umls:play_role rdf:resource=
      'http://ns.inria.fr/bio/cart_dev'/>
</umls:Gene_or_Protein>
```

In the example above, we presented the use of the extensions on a single sentence; the same treatment can be performed on whole documents in order to generate rich annotations used to improve the information retrieval task. We can also process more complex sentences containing, for example, (i) linguistic variations (*'cartilage development' vs 'development of cartilage'*), (ii) conjunctions (and/or) and (iii) passive forms (*cartilage development is affected by Wnt proteins*). The interest of our approach is to perform complex treatment using a single/simple query.

7 Related work

In addition to XSLT [Cla99] and XQuery [W3C07] that would enable querying RDF and XML altogether, we have identified related work shown below.

The closest work that we have found in the literature is SPAT [Pru07], SPARQL Annotations. SPAT allows associations between RDF terms and information in an XML document by adding XPaths to SPARQL triple patterns. SPAT statements can be seen as patterns to generate RDF annotations in the context of an XML document and conversely. It is not clear whether SPAT can be used in a query when there is no XML document in context.

SparqPlug: Generating Linked Data from Legacy HTML, SPARQL and the DOM [CHM08]. This work aims at treating XML DOM as a data source to be queried using SPARQL. Authors propose a scheme to systematically translate DOM tree to RDF graph using a predefined set of predicates such as subNodes, hasAttributes, etc. Then they generate a SPARQL query to extract target data from the generated RDF graph. This is another way of integrating XML within RDF.

Translating XPath Queries into SPARQL Queries: This work is similar to SparqPlug described above [DFG+07].

XSPARQL [AKKP07] is a merge of SPARQL statements into XQuery. A special *for* binding statement is introduced to bind results coming from a SPARQL nested query to XQuery variables. The result of an XSPARQL query can also be a construct clause. The implementation relies on an XQuery interpreter and a SPARQL interpreter. As far as we understand, it enables to query RDF using SPARQL within XQuery but not to query XML within SPARQL.

Embedding SPARQL into XQuery/XSLT [GGL+08]: SPARQL is integrated as is in XQuery with a binding statement to retrieve SPARQL results and binds them to variables of the host language. It is similar to XSPARQL, although it seems not to process construct. The integration in XSLT is similar and it is possible to bind the results to XSLT variables. In our approach, the result of a SPARQL query in XSLT is returned as SPARQL Query Results XML Format.

XSLT+SPARQL: Scripting the Semantic Web with SPARQL embedded into XSLT Stylesheets [BLH08]. This work is very similar to our integration of SPARQL into XSLT using external function call, the result of the query is also processed as XML Result Format.

In conclusion, none of these works enable to process XML directly within SPARQL, although they propose functionalities that are related to our vision. It is is clear that integrating XML and RDF processing is crucial and up to date.

8 Conclusion and future work

We have presented extensions of SPARQL for the Semantic Web of Data implemented within the Corese Semantic Factory. We can use XPath and XSLT within SPARQL, SPARQL within XSLT. We have proposed extensions to SPARQL such as evaluable expressions in the `select` clause, `construct` with such `select` and sequence of values by means of a new datatype. In addition, using the same design pattern, we are able to integrate nested SPARQL queries

and nested SQL queries. We plan to integrate XQuery using another external function.

We are testing these principles in text mining and NLP applications to generate semantic metadata from text in the context of the BioMarker project. The goal is to process XML abstract syntax parse trees from NLP tools such as GATE and RASP and to generate semantic annotations.

In conclusion, this work is a powerful step towards the seamless integration of RDF and XML processing on the Semantic Web of Data. All that is presented in this paper is implemented within the Corese Semantic Factory.

Acknowledgements

Special thanks to Adil El Ghali for the idea of integrating XSLT into SPARQL and to Martine Collard for discussions that led to the integration of SQL.

NLP experiments in BioMarker are funded by the French Ministry of Industry, the PASS cluster (Pôle Parfums, Arôme, Senteurs, Saveurs) and Région PACA .

References

[AB08] Ben Adida and Mark Birbeck. RDFa Primer - Embedding Structured Data in Web Pages. Working Group Note, W3C, march 2008. http://www.w3.org/TR/xhtml-rdfa-primer.

[AKKP07] Waseem Akhtar, Jacek Kopecký, Thomas Krennwallner, and Axel Polleres. XSPARQL: Traveling Between the XML and RDF Worlds and Avoiding the XSLT Pilgrimage. Technical report, Deri, December 2007.

[BCW06] E. Briscoe, J. Carroll, and R. Watson. The Second Release of the RASP System. In *Proc. COLING/ACL*, Sydney, Australia, 2006.

[BLH08] Diego Berrueta, Jose Emilio Labra, and Ivan Herman. XSLT+SPARQL: Scripting the Semantic Web with SPARQL embedded into XSLT Stylesheets. In *Proc. 4th Workshop on Scripting for the Semantic Web, ESWC*, Tenerife, Spain, June 2008.

[BLP00] Peter Bergström, Robin La Fontaine, and David Price. Implementation methods: XML representation of EXPRESS schemas and data. Technical Report 10303-0028, ISO, 2000.

[BPSM98] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. Recommendation, W3C, 1998. http://www.w3.org/TR/REC-xml.

[CD99] James Clark and Steve DeRose. XML Path Language (XPath). Recommendation, W3C, 1999. http://www.w3.org/TR/xpath.

[CFZ07] Olivier Corby and Catherine Faron-Zucker. Implementation of SPARQL Query Language based on Graph Homomorphism. In

Proc. of the 15th Int. Conference on Conceptual Structures (ICCS), pages 472–475, Sheffield, UK, July 2007.

[CHM08] Peter Coetzee, Tom Heath, and Enrico Motta. SparqPlug: Generating Linked Data from Legacy HTML, SPARQL and the DOM. In *Proc. WWW'2008 Workshop about Linked Data on the Web (LDOW)*, 2008.

[Cla99] James Clark. XSL Transformations (XSLT). Recommendation, W3C, 1999. http://www.w3.org/TR/xslt.

[CM92] Michel Chein and Marie-Laure Mugnier. Conceptual Graphs: Fundamental Notions. *Revue d'Intelligence Artificielle*, 6(4):365–406, April 1992.

[CMBT02] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *ACL'02*, 2002.

[Cor08] Olivier Corby. Web, Graphs & Semantics. In *Proc. of the 16th International Conference on Conceptual Structures (ICCS'2008)*, Toulouse, July 2008.

[DFG+07] M. Droop, M. Flarer, J. Groppe, S. Groppe, V. Linnemann, J. Pinggera, F. Santner, M. Schier, F. Schöpf, H. Staffler, and S. Zugal. Translating XPath Queries into SPARQL Queries. In *OTM Workshops*, pages 9–10. Springer Verlag LNCS, 2007.

[Gan07] Fabien Gandon. GRDDL Use Cases: Scenarios of extracting RDF data from XML documents. WG Note, W3C, April 2007. http://www.w3.org/TR/grddl-scenarios.

[GGL+08] Sven Groppe, Jinghua Groppe, Volker Linnemann, Dirk Kukulenz, Nils Hoeller, and Christoph Reinke. Embedding SPARQL into XQuery/XSLT. In *Proc. ACM Symposium on Applied Computing*, Fortaleza, Ceará, Brazil, March 2008.

[HL93] B. Humphreys and D. Lindberg. The UMLS Project: Making the Conceptual Connection Between Users and the Information they Need. In *Bulletin of the Medical Library Association*, page 81(2):170, 1993.

[KDKB07] Khaled Khelif, Rose Dieng-Kuntz, and Pascal Barbry. An Ontology-Based Approach to Support Text Mining and Information Retrieval in the Biological Domain. *Journal of Universal Computer Science (JUCS), Special Issue on Ontologies and their Applications*, 2007.

[MMM04] Frank Manola, Eric Miller, and Brian McBride. RDF Primer. Recommendation, W3C, 2004. http://www.w3.org/TR/rdf-primer.

[Pru07] Eric Prud'hommeaux. SPAT - SPARQL Annotations. Work in Progress, W3C, February 2007. http://www.w3.org/2007/01/SPAT.

[PS08] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Recommendation, W3C, 2008. http://www.w3.org/TR/rdf-sparql-query.

[Sch94] H. Schmid. Probabilistic Part-of-Speech Tagging using Decision Trees. In *Proc. Int. Conference on New Methods in Language Processing*, Manchester, 1994.

[Sow84] John Sowa. *Conceptual Structures - Information Processing in Mind and Machine*. Addison Wesley, 1984.

[W3C07] W3C. XQuery 1.0: An XML Query Language. Recommendation, W3C, 2007. http://www.w3.org/TR/xquery.

Contents

1 Introduction **3**
1.1 Use cases . . . 3
1.2 Challenge . . . 4

2 SPARQL Extension **4**
2.1 Select expression . . . 4
2.2 Construct with select . . . 5

3 XPath in SPARQL **5**
3.1 Sharing namespaces and variables . . . 7
3.2 XPath expression in RDF metadata . . . 8
3.3 Query Embedded XMLLiteral . . . 8
3.4 Query RDFa . . . 9
3.5 Select expression . . . 9
3.6 Construct using XPath . . . 10
3.7 XPointer . . . 10

4 Processing other languages **10**
4.1 SPARQL and XSLT . . . 11
4.2 XSLT and SPARQL . . . 11
4.3 Nested SPARQL . . . 12
4.4 Nested SQL . . . 12

5 Implementation **13**
5.1 SPARQL Filter evaluator . . . 13
5.2 Corese Implementation . . . 14

6 Application **15**
6.1 Relations and concepts detection . . . 15
6.2 Annotation generation . . . 15

7 Related work **18**

8 Conclusion and future work **18**

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Futurs : Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399