XML/RDF Query By Example

http://dyomedea.com/papers/2005-extreme/

Extreme Markup Languages 2005

By:

Eric van der Vlist, Dyomedea (vdv@dyomedea.com)

Context

The French Institut national de la statistique et des études économiques (INSEE) needs a XML vocabulary to expose the consolidated content of several LDAP databases.

LDAP can be seen as the combination of a graph of classes, objects and a hierarchy similar to traditional file system directories. This combination can be compared to what we would obtain by attaching RDF triples to each directory of a file system.

LDAP as a tree

A first approach to serializing LDAP as XML is to focus on its hierarchical dimension and to map this hierarchy into an XML tree of elements and attributes. This could lead to something such as:

```
<?xml version="1.0" encoding="utf-8"?>
<annuaire>
 <fr>
  <insee>
   <Personnes>
    <inseePerson dn="uid=R3D2,ou=Personnes,o=insee,c=fr">
     <telephoneNumber>0123456789</telephoneNumber>
     <cn>Laurent Dupondt</cn>
     <inseeFonction dn="uid=GS10,ou=Fonctions,o=insee,c=fr"/>
     <inseeTimbre>SED</inseeTimbre>
     <uid>R3D2</uid>
     <inseeNomGIP>Dupondt</inseeNomGIP>
     <inseeDomaineNT>DR40A</inseeDomaineNT>
     <sn>Dupondt</sn>
     <inseeGroupeDefaut>MVS:SE40</inseeGroupeDefaut>
     <employeeType>Interne</employeeType>
     <inseePrenomGIP>LAURENT</inseePrenomGIP>
     <inseeServeurExchange>S40X01</inseeServeurExchange>
     <roomNumber>113</roomNumber>
     <inseeGrade>Attaché</inseeGrade>
     <personalTitle>M</personalTitle>
     <mail>laurendt.dupondt@pas-de-pourriel.fr</mail>
     <givenName>Laurent</givenName>
```

```
     <inseeUnite dn="ou=DR54-SED,ou=Unit%C3%A9s,o=insee,c=fr"/>
     <objectClass>top</objectClass>
     <objectClass>person</objectClass>
     <objectClass>organizationalPerson</objectClass>
     <objectClass>inetOrgPerson</objectClass>
     <objectClass>InseePerson</objectClass>
     <employeeNumber>12345</employeeNumber>
     <ou>DR54-SED</ou>
    </inseePerson>
   </Personnes>
  </insee>
 </fr>
</annuaire>
```

In this example, we have mapped the root of the LDAP structure into an "annuaire" document element and each branch and object of the LDAP directory into an XML element.

This approach gives the primary role to the hierarchical facet of LDAP and does not natively expose its graph facet that needs to be implemented using some kind of links such as ID/IDREF, XLink or application specific links: in this example, we had chosen to use the LDAP "distinguished names" (dn) as identifiers.

LDAP as a graph

This focus on the hierarchical dimension happens to be very far from the most typical use of LDAP repositories at the INSEE: the structure adopted by the INSEE is on the contrary rather flat with many objects under each node. For instance, all the persons are stored under the same node (ou=Personnes,o=insee,c=fr). To take this pattern into account, the natural thing was thus to give the primary role to the graph dimension.

That being said, there is a standard vocabulary for describing graphs in XML and this vocabulary is called RDF.

While it sounded very natural to use RDF to serialize LDAP in XML, I had to take into account the fact that the initial request was to define a XML vocabulary and that I had to maintain this vocabulary acceptable for XML heads.

The context was thus quite similar to the one in which we were when we've published the RSS 1.0 specification: defining a RDF vocabulary that is acceptable by both XML and RDF heads.

I think that being acceptable by RDF heads is the easiest part since this basically means that the vocabulary has to be conform to the RDF/XML Syntax Specification: there is a clear specification to comply with.

Being acceptable by XML heads means that the XML vocabulary has to "look" natural to people who do not know RDF and that the so called "RDF tax" has to be minimized.

Concretely, that involves choosing between the many options available to express the same set of triples in RDF/XML the one should be the most straightforward and least surprising to someone who knows XML but do not know RDF.

We ended up with a vocabulary that looks like:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://xml.insee.intra/schema/annuaire/"
xmlns:a="http://xml.insee.intra/schema/annuaire/"
 xmlns:l="http://xml.insee.intra/schema/ldap/">
 <inseePerson rdf:about="http://xml.insee.fr/ldap/uid=R2D2,ou=Personnes,o=insee,c=fr">
  <l:dn>uid=R2D2,ou=Personnes,o=insee,c=fr</l:dn>
  <l:parent rdf:resource="http://xml.insee.fr/ldap/ou=Personnes,o=insee,c=fr"/>
  <l:ancestorOrSelf
rdf:resource="http://xml.insee.fr/ldap/uid=R2D2,ou=Personnes,o=insee,c=fr"/>
  <l:ancestorOrSelf rdf:resource="http://xml.insee.fr/ldap/ou=Personnes,o=insee,c=fr"/>
  <l:ancestorOrSelf rdf:resource="http://xml.insee.fr/ldap/o=insee,c=fr"/>
  <l:ancestorOrSelf rdf:resource="http://xml.insee.fr/ldap/c=fr"/>
  <l:ancestorOrSelf rdf:resource="http://xml.insee.fr/ldap/"/>
  <telephoneNumber>0383918546</telephoneNumber>
  <cn>Laurent Dupondt</cn>
  <inseeFonction
rdf:resource="http://xml.insee.fr/ldap/uid=GS10,ou=Fonctions,o=insee,c=fr"/>
  <inseeTimbre>SED</inseeTimbre>
  <uid>R2D2</uid>
  <inseeNomGIP>DUPONDT</inseeNomGIP>
  <inseeDomaineNT>DR40A</inseeDomaineNT>
  <sn>Dupondt</sn>
  <inseeGroupeDefaut>MVS:SE40</inseeGroupeDefaut>
  <employeeType>Interne</employeeType>
  <inseePrenomGIP>LAURENT</inseePrenomGIP>
  <inseeServeurExchange>S40X01</inseeServeurExchange>
  <roomNumber>113</roomNumber>
  <inseeGrade>Attaché</inseeGrade>
  <personalTitle>M</personalTitle>
  <mail>laurendt.dupondt@pas-de-pourriel.fr</mail>
  <givenName>Laurent</givenName>
  <inseeUnite rdf:resource="http://xml.insee.fr/ldap/ou=DR54-SED,ou=Unit%C3%
A9s,o=insee,c=fr"/>
  <objectClass>top</objectClass>
  <objectClass>person</objectClass>
  <objectClass>organizationalPerson</objectClass>
  <objectClass>inetOrgPerson</objectClass>
  <objectClass>InseePerson</objectClass>
  <employeeNumber>12345</employeeNumber>
  <ou>DR54-SED</ou>
 </inseePerson>
```

The distinguished names have been translated into URIs usable as RDF identifiers and the hierarchy has been expressed using <l:dn>, <l:parent> and <l:ancestorOrSelf> elements.

The RDF tax has been limited to using these URIs as identifiers in rdf:about and rdf:resource attributes and using <rdf:RDF> as the document element.

The choice of these elements has been dictated by the fact that they match the three ways to define the scope when doing LDAP queries and we'll come back to that point later on.

LDAP as a graph AND as a tree

When I have started writing this introduction, it suddenly occurred to me that LDAP wasn't

the only technology that superposes a graph and a tree and that this was also the case of RDF/XML documents that are both a graph if you read them with a RDF parser and a tree if you read them with a XML parser.

There is thus a third way which would be to use the XML hierarchy to describe the hierarchical dimension of LDAP and to use the RDF features to describe the graph dimension of LDAP.

This third way could look like:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://xml.insee.intra/schema/annuaire/"
xmlns:a="http://xml.insee.intra/schema/annuaire/"
 xmlns:l="http://xml.insee.intra/schema/ldap/">
 <l:root>
  <l:node rdf:parseType="Resource">
   <c>fr</c>
   <l:node rdf:parseType="Resource">
    <o>insee</o>
    <l:node rdf:parseType="Resource">
     <ou>Personnes</ou>
     <inseePerson rdf:parseType="Resource">
      <telephoneNumber>0123456789</telephoneNumber>
      <cn>Laurent Dupondt</cn>
      <inseeFonction
rdf:resource="http://xml.insee.fr/ldap/uid=GS10,ou=Fonctions,o=insee,c=fr"/>
      <inseeTimbre>SED</inseeTimbre>
      <uid>R2D2</uid>
      <inseeNomGIP>DUPONDT</inseeNomGIP>
      <inseeDomaineNT>DR40A</inseeDomaineNT>
      <sn>Dupondt</sn>
      <inseeGroupeDefaut>MVS:SE40</inseeGroupeDefaut>
      <employeeType>Interne</employeeType>
      <inseePrenomGIP>LAURENT</inseePrenomGIP>
      <inseeServeurExchange>S40X01</inseeServeurExchange>
      <roomNumber>113</roomNumber>
      <inseeGrade>Attaché</inseeGrade>
      <personalTitle>M</personalTitle>
      <mail>laurendt.dupondt@pas-de-pourriel.fr</mail>
      <givenName>Laurent</givenName>
      <inseeUnite rdf:resource="http://xml.insee.fr/ldap/ou=DR54-SED,ou=Unit%C3%
A9s,o=insee,c=fr"/>
      <objectClass>top</objectClass>
      <objectClass>person</objectClass>
      <objectClass>organizationalPerson</objectClass>
      <objectClass>inetOrgPerson</objectClass>
      <objectClass>InseePerson</objectClass>
      <employeeNumber>12345</employeeNumber>
      <ou>DR54-SED</ou>
     </inseePerson>
    </l:node>
   </l:node>
```

```
  </l:node>
 </l:root>
</rdf:RDF>
```

Being a brand new idea coming very late after the vocabulary has been more or less finalized, this snippet isn't something polished at all but just there to give you an idea of where that could lead!

THE PROBLEM

So far, so good and I think that we've made a decent job in our quest to define a "low RDF tax" RDF/XML vocabulary. This has been a fun small project which could justify a proposal to another XML conference but probably not to Extreme!

The problem became more challenging when we've started to address the second request: define an XML vocabulary to express queries against the LDAP repository using a syntax that was coherent with what we'd done so far.

We had basically three types of standards on which we could have relied...

LDAP filters (RFC 2254)

The first one was the RFC 2254 that defines a text format to express LDAP filters.

LDAP queries are specified by defining a search base and a LDAP filter. For instance, the filter "(inseeRoleApplicatif=RP$$P$SUP*)" search for objects which attribute " inseeRoleApplicatif" matching the value "RP$$P$SUP*". The star character ("*") being a wildcard that replaces any character, this match means that the attribute has to begin with "RP$$P$SUP".

As a vocabulary, we could have embedded such filters into a wrapper element together with the search base:

```
<?xml version="1.0" encoding="utf-8"?>
<ldapSearch>
  <base scope="subTree">Personnes,o=insee,c=fr</base>
  <filter>(inseeRoleApplicatif=RP$$P$SUP*)</filter>
</ldapSearch>
```

That would have been handy for people familiar with LDAP, but we wanted to provide something usable for users knowing our RDF/XML vocabulary without requiring that they learn LDAP is they didn't know it.

W3C XQuery

XQuery was another natural candidate. The same query would give:

```
xquery version "1.0";
declare namespace a = "http://xml.insee.intra/schema/annuaire/";
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
{
      for $x in /rdf:RDF/a:inseePerson
```

```
      where $x[a:inseeRoleApplicatif[ starts-with(., 'RP$$P$SUP') ]]
      return $x }
</rdf:RDF>
```

That's handy for XML heads who see the RDF/XML serialization as pure XML but where are the triples that RDF heads see?

Furthermore, XQuery is fairly complex and this complexity would create two issues in our context:

- Users would have to learn XQuery.
- Implementing an XQuery engine on top of the LDAP gateway would be a daunting task.

We've thus considered that while XQuery didn't preserve the balance between XML and RDF that we had carefully built, it was also be an overkill!

W3C SPARQL (or any other RDF query language)

I am not a SPARQL guru, so please be kind with me if the following snippet contains errors, but the same query expressed in SPARQL would look like:

```
PREFIX : http://xml.insee.intra/schema/annuaire/
SELECT ?inseePerson
WHERE {
      ?inseePerson rdf:type :inseePerson;
      ?inseePerson :inseeRoleApplicatif ?inseeRoleApplicatif.
      FILTER regex(str(?inseeRoleApplicatif), "^RP$$P$SUP ")

}
```

Some RDF heads would be much happier with this expression but not all of them would agree that this is the way to go: RDF query languages are still very young and many different approaches have been proposed.

Furthermore, XML heads would be totally lost, users would have to learn SPARQL (and before that they would have to learn how to distinguish triples in XML fragments) and implementing a SPARQL engine on top of our LDAP gateway would be almost as challenging as implementing an XQuery engine!

Home bred

After having rejected the three standards on which we could have relied, we came to the conclusion that we would have to create our own query language that should be both coherent with the syntax used by our RDF/XML vocabulary and easy to use by non-guru-XML-literates.

These requirements rang two bells...

When I started giving presentations on XML schema languages, I used to say that XML schemas are more complex than the instances they describe and ended up proposing Examplotron, a schema language based on examples.

Before that, one of the easiest database I have ever used had been Borland's Paradox, and its query language was a Query By Example (QBE) language.

QUERY BY EXAMPLE (QBE)

In his "Principles of Database Systems", Jeffrey D. Ullman defines QBE as a "domain calculus language" developed at IBM that "contains a number of features not present in relational algebra or calculus, or in any of the implemented query languages we have discussed."

The QBE we had in mind is not that ambitious, but that statement is a good indication that, while we would probably limit the features we would implement, we would not be limited by the method itself.

Since we are not designing a user interface but an XML/RDF query language, our QBE would naturally be based on a RDF/XML syntax that would be as close as possible to the XML/RDF vocabulary used to formalize the results. Being a XML vocabulary, it was also natural enough to use a specific namespace to distinguish the instructions of the query language from the actual examples.

Also, as already mentioned, one of our goals is to define a query language that can be read both as XML documents but also as valid XML/RDF models and processed both from its XML infoset or from its sets RDF triples.

Basic query structure

In IBM's original QBE, the same set of columns were used to indicate the list of columns to print and the conditions on the columns. The distinction between these two features was done using different sets of operators.

This reduces the "legibility as examples" of the values placed in the columns and we have considered that it would be more readable to separate the definition of what needs to be returned from the definition of the conditions.

This led us toward a structure that looks like the classical "select X from Y" structure of a SQL select statement:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:what>
            .../...
        </q:what>
        <q:where>
            .../...
        </q:where>
    </q:select>
</rdf:RDF>
```

Typical XML/RDF documents have a relatively flat structure and we can follow this style to express the what and where clauses of our request :

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
```

```
        <q:what rdf:resource="#what"/>
        <q:where rdf:resource="#where"/>
    </q:select>
    <inseePerson rdf:ID="what"/>
    <inseePerson rdf:ID="where">
        <mail>jean.dupondt@insee.fr</mail>
    </inseePerson>
</rdf:RDF>
```

The what clause is optional and when it is omitted, the query returns the complete element specified in the where clause (this would be equivalent to a SQL "select * from ..." query). In this example, that would give:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:where rdf:resource="#where"/>
    </q:select>
    <inseePerson rdf:ID="where">
        <mail>jean.dupondt@insee.fr</mail>
    </inseePerson>
</rdf:RDF>
```

In these two equivalent queries, we are requesting "inseePerson" elements that have mail addresses equal to "jean.dupondt@insee.fr".

While this flat style looks XML/RDFish, it is not that usual to XML heads and we accept an alternative Russian doll syntax (using what RDF heads call a blank node):

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:what>
            <inseePerson/>
        </q:what>
        <q:where>
            <inseePerson>
                <mail>jean.dupondt@insee.fr</mail>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

This alternative syntax produces a very similar set of RDF triples and we consider it as strictly equivalent.

The last alternative for this same query would be to omit the what clause:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:where>
            <inseePerson>
                <mail>jean.dupondt@insee.fr</mail>
```

```
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

Introducing functions

What happens if we want to express something a little bit more complex and write that we want the <inseePerson/> whose email address ends with "@insee.fr" ?

Since we are in design mode, we have many possibilities to implement that feature.

The first one which is the one that has been adopted by IBM's original QBE would be to use functions in literals:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:where>
            <inseePerson>
                <mail>ends-with(@insee.fr)</mail>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

I don't like this solution for a couple of reasons:

- The functions are called using a plain text syntax that requires a parsing which is not obvious to do with "pure" XML tools such as XSLT 1.0.
- If you look at this document with RDF glasses, the triple '_:genid3 <http://xml.insee.intra/schema/annuaire/mail> "ends-with(@insee.fr)" ' should mean "_:genid3 has a mail property equal to 'ends-with(@insee.fr)' ". Here we are we are interpreting it as "_:genid3 has a mail property matching the condition 'ends-with(@insee.fr) ' " and that just doesn't sound right.

The way to avoid these plain text function calls is to replace them by XML elements (or RDF triples if you prefer). The first (naive) attempt to do so would be to write :</para

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:what>
            <inseePerson/>
        </q:what>
        <q:where>
            <inseePerson>
                <mail>
                    <q:ends-with>@insee.fr</q:ends-with>
                </mail>
```

```
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

This fine looks nice to XML heads, but makes Jena scream:</para

```
_:jA3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xml.insee.intra/schema/qbe/ends-with> .
Error: file:///home/vdv/cvs-private/presentations/en/extreme2005/query4.rdf[11:43]:
{E202} Expected whitespace found: '@insee.fr'. Maybe a missing rdf:parseType='Literal',
or a striping problem.
```

Why is that? In fact, the rules that bind XML nodes to RDF triples in the XML/RDF syntax are well designed enough that they've produced what we'd expected so far but they're biting us in this last example! Let's think more about these rules...

When we write :

```
<rdf:RDF>
  <foo>
   <bar>
    <baz>
      <bat>XXX</bat>
    </baz>
  </bar>
 </foo>
<rdf:RDF>
```

the different elements are not treated as equal by the XML/RDF binding rules and the triples that a RDF parser will extract from this document are:

```
Subj.  Pred.   Obj.
<foo> <bar> <baz>
<baz> <bat> "XXX"
```

<foo> and <baz> are resources that are used as subject and cannot have literals directly attached to them while <bar> and <bat> are resources that are used as predicates and can have literals directly attached to them.

In our naive attempt that made Jena scream, we've attached the literal "@insee.fr" to the resource <q:ends-with> that was in a position where it was considered as a subject.

There are a couple of solutions to work around this error.

The first one is to use the rdf:parseType attribute to specify that <mail> should not be considered only as a predicate but also as a resource that can be the subject of a triple :

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:where>
            <inseePerson>
                <mail rdf:parseType="Resource">
                    <q:ends-with>@insee.fr</q:ends-with>
                </mail>
            </inseePerson>
        </q:where>
```

```
    </q:select>
</rdf:RDF>
```

The trick, here, is that since <mail> is now a subject, <q:ends-with> becomes a predicate that can have a literal directly attached to it.

Technically speaking, this does the job of making Jena happy, but I don't like it that much since it requires a good understanding of the XML/RDF binding rule to be able to tell when and where you need to add rdf:parseType="Resource" attributes. Requiring such an understanding doesn't meet our goal to define a query language that is understandable by "pure XML heads".

The second solution (which we've adopted) is to add a level of hierarchy:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:where>
            <inseePerson>
                <mail>
                    <q:conditions>
                        <q:ends-with>@insee.fr</q:ends-with>
                    </q:conditions>
                </mail>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

The trick here is that since <q:ends-with> needs to be a predicate, we've added <q:conditions> to serve has its subject. RDF parsers are happy and that seems to be cleaner than the previous workaround because that does not change the nature of <mail> which is and stays a predicate.

Of course, the same query can be written with a flat style with or without an explicit what clause:</para

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

```
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:where rdf:resource="#where"/>
    </q:select>
    <inseePerson rdf:ID="where">
        <mail>
            <q:conditions>
                <q:ends-with>@insee.fr</q:ends-with>
            </q:conditions>
        </mail>
    </inseePerson>
</rdf:RDF>
```

Joins

So far, we've queried the <mail> predicate of <inseePerson> resources. Now, let's see what happens if we want to get <inseePerson> resources that have an <inseeUnite> which <ou> predicate is equal to "DG75-C460". In XML, that would mean that we are looking for situations such as:

```
<rdf:RDF>
    <insee:Person>
        <inseeUnite rdf:resource="xxx"/>
    </insee:Person>
    <inseeUnite rdf:about="xxx">
        <ou>DG75-C460</ou>
    </inseeUnite>
</rdf:RDF>
```

But, since we are also thinking to RDF heads, such a situation needs to be equivalent to:

```
<rdf:RDF>
    <insee:Person>
        <inseeUnite rdf:parseType="Resource">
            <ou>DG75-C460</ou>
        </inseeUnite>
    </insee:Person>
</rdf:RDF>
```

And we'll leave people use indifferently the first or the second style. Following a Russian doll style, our query can thus be written:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:where>
            <inseePerson>
                <inseeUnite rdf:parseType="Resource">
                    <ou>DG75-C460</ou>
                </inseeUnite>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

Why do we accept rdf:parseType="Resource" in this case when we've rejected it in the previous section? That's a good question :-) ...

In the previous section, we would have been adding rdf:parseType="Resource" in a <mail> element if and only if we wanted to add a condition and that seemed arbitrary. Here we are adding rdf:parseType="Resource" to <inseeUnite> because we want to specify that <inseeUnite> is a resource and not only a property and that should be much easier to swallow by anyone, XML or RDF head!

Join and condition

So far so good, but how does that scale? Can we, for instance, add a condition after a join and

search the <inseePerson> which <inseeUnite> have a <ou> that starts with "DDEQ"?

That's easy enough and what we've seen so far works well together:</para

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:q="http://xml.insee.intra/schema/qbe/">
    <q:select>
        <q:where>
            <inseePerson>
                <inseeUnite rdf:parseType="Resource">
                    <ou>
                        <q:conditions>
                            <q:starts-with>DG75</q:starts-with>
                        </q:conditions>
                    </ou>
                </inseeUnite>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

What about several conditions?

What if I want to meet two different conditions?

To keep it simple we've adopted the principle that by default, all the conditions grouped under a single where clause are anded:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/">
    <q:select>
        <q:where>
            <inseePerson>
                <cn>
                    <q:conditions>
                        <q:contains>a</q:contains>
                    </q:conditions>
                </cn>
                <mail>
                    <q:conditions>
                        <q:contains>o</q:contains>
                    </q:conditions>
                </mail>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

This simple rule is enough to express set of conditions that must all be verified. For other cases, we have introduced two sets of elements that need to be used in conjunction:

- <q:if> and <q:if-not> are containers for sets of conditions that must or must not be verified.

- <a:any> and <q:all> are containers for sets of conditions that perform a logical or or a logical and between these conditions.

An <q:if> or a <q:if-not> must always be used together with an embedded <q:all> or <q:any>. This allows to represent the four possible combinations and also to meet the XML/RDF triple binding rules without having to require additional rdf:parseType attributes.

The last query is thus a shortcut for:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/">
    <q:select>
        <q:where>
            <inseePerson>
                <q:if>
                    <q:all>
                        <cn>
                            <q:conditions>
                                <q:contains>a</q:contains>
                            </q:conditions>
                        </cn>
                        <mail>
                            <q:conditions>
                                <q:contains>o</q:contains>
                            </q:conditions>
                        </mail>
                    </q:all>
                </q:if>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

The other combinations would be:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/">
    <q:select>
        <q:where>
            <inseePerson>
                <q:if-not>
                    <q:all>
                        <cn>
                            <q:conditions>
                                <q:contains>a</q:contains>
                            </q:conditions>
                        </cn>
                        <mail>
                            <q:conditions>
                                <q:contains>o</q:contains>
                            </q:conditions>
                        </mail>
                    </q:all>
```

```
                </q:if-not>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/">
    <q:select>
        <q:where>
            <inseePerson>
                <q:if>
                    <q:any>
                        <cn>
                            <q:conditions>
                                <q:contains>a</q:contains>
                            </q:conditions>
                        </cn>
                        <mail>
                            <q:conditions>
                                <q:contains>o</q:contains>
                            </q:conditions>
                        </mail>
                    </q:any>
                </q:if>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

and

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/">
    <q:select>
        <q:where>
            <inseePerson>
                <q:if-not>
                    <q:any>
                        <cn>
                            <q:conditions>
                                <q:contains>a</q:contains>
                            </q:conditions>
                        </cn>
                        <mail>
                            <q:conditions>
                                <q:contains>o</q:contains>
                            </q:conditions>
                        </mail>
                    </q:any>
                </q:if-not>
            </inseePerson>
        </q:where>
```

```
    </q:select>
</rdf:RDF>
```

The same <q:if>, <q:if-not>, <q:any> and <q:all> elements can be used with the same meaning under <q:conditions> elements, for instance:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/">
    <q:select>
        <q:where>
            <inseePerson>
                <mail>
                    <q:conditions>
                        <q:if-not>
                            <q:all>
                                <q:contains>a</q:contains>
                                <q:contains>@insee.fr</q:contains>
                                <q:starts-with>laurent</q:starts-with>
                            </q:all>
                        </q:if-not>
                    </q:conditions>
                </mail>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

Search base

We've seen how what LDAP would call query filters can be expressed using our query language, what about LDAP search base?

I have already said that we had added the following properties to our RDF/XML vocabulary to support this feature:

```
  <l:dn>uid=R2D2,ou=Personnes,o=insee,c=fr</l:dn>
  <l:parent rdf:resource="http://xml.insee.fr/ldap/ou=Personnes,o=insee,c=fr"/>
  <l:ancestorOrSelf
rdf:resource="http://xml.insee.fr/ldap/uid=R2D2,ou=Personnes,o=insee,c=fr"/>
  <l:ancestorOrSelf rdf:resource="http://xml.insee.fr/ldap/ou=Personnes,o=insee,c=fr"/>
  <l:ancestorOrSelf rdf:resource="http://xml.insee.fr/ldap/o=insee,c=fr"/>
  <l:ancestorOrSelf rdf:resource="http://xml.insee.fr/ldap/c=fr"/>
  <l:ancestorOrSelf rdf:resource="http://xml.insee.fr/ldap/"/>
```

These properties will be used in our queries like any other property and will match the three different types of search base.

The first one, most commonly used at the INSEE is a search base with a scope equal to "subTree". That means that the search is performed in the object itself and all its descendants. To express that, we will use the <l:ancestorOrSelf> property, for instance:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/"
```

```
    xmlns:l="http://xml.insee.intra/schema/ldap/">
    <q:select>
        <q:where>
            <inseePerson>
                <l:ancestorOrSelf rdf:parseType="Resource">
                    <l:dn>o=insee,c=fr</l:dn>
                </l:ancestorOrSelf>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

The second case is a scope equal to "oneLevel" and this one means that the search is to be performed among the immediate children of the search base. To express that, we will use the <l:parent> property:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:l="http://xml.insee.intra/schema/ldap/">
    <q:select>
        <q:where>
            <inseePerson>
                <l:parent rdf:parseType="Resource">
                    <l:dn>o=insee,c=fr</l:dn>
                </l:parent>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

The third and last case is a scope equal to "base" and in that case, the search has to be performed on the object itself. For that last case, we will be using the <l:dn> property:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:q="http://xml.insee.intra/schema/qbe/"
    xmlns="http://xml.insee.intra/schema/annuaire/"
    xmlns:l="http://xml.insee.intra/schema/ldap/">
    <q:select>
        <q:where>
            <inseePerson>
                <l:dn>o=insee,c=fr</l:dn>
            </inseePerson>
        </q:where>
    </q:select>
</rdf:RDF>
```

CURRENT STATUS

The query language is now more or less stabilized. A first proof of concept has been implemented with Orbeon PresentationServer that transforms these queries into XQuery queries using a XSLT 1.0 transformation and queries an XML database (eXist).

The final implementation is now under development. It will transform these queries into

LDAP searches sent to the LDAP repository and serialize the answers as RDF/XML.

The main difficulty in this implementation is to support the joins that do exist in LDAP filters and have to be performed by the gateway itself.

CONCLUSION

While this project is being developed for a very specific target, the design decisions have not been influenced by the context and I believe that this experience could be generalized to any project needing a query language matching these three conditions :

- The source to query is expressed (or can be expressed) as RDF/XML.
- The query language needs to make sense for both XML and RDF heads.
- The query language needs to look like the source to query (QBE).

I'd like to thank my customer (INSEE) for having funded this work and Franck Cotton who is leading this project.