Using OWL and SWRL for the Semantic Analysis of XML Resources[⋆]

Jesús M. Almendros-Jiménez

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

Abstract. In this paper we describe how to analyze the semantic content of XML documents. With this aim, XML resources are mapped into an OWL ontology and SWRL rules are used for specifying XML semantic content. We have implemented and tested the approach. The implementation is based on a semantic web library for XQuery which includes primitives for mapping XML into OWL, and for specifying and executing SWRL.

1 Introduction

Extensible Markup Language (XML) has been introduced for representing Web data. XML allows give structure to Web data thanks to the use of labels which can be nested. The hierarchical structure of XML documents permits to describe data to be exchange between applications. XML has been successfully applied to e-business and service-oriented computing, among others.

XML is equipped with data definition languages (DTD and XML Schema) whose aim is to describe the syntactic structure of XML documents. Well-formed XML documents conform to the corresponding DTD and XML Schema. However, even though XML documents can be well-formed, the content can be redundant, which can lead to inconsistency, as well as the content can violate imposed restrictions on data.

In the database scientific community many efforts have been achieved to give semantic content to data sources. The entity-relationship (ER) model, functional dependences and integrity constraints are key elements of database design. ER model gives semantics to data relations and restrict cardinality of relations. Functional dependences establish data dependences and key values. Integrity constraints impose restrictions on data values in a certain domain.

However, most available XML resources lack in the description of the semantic content. But it also happens for the syntactic content: DTD and XML schemas might not available in Web resources. It makes that data exchange fails due to bad understanding of XML content. Nevertheless, XML records can follow

⋆ This work has been supported by the Spanish Ministry MICINN under grant TIN2008-06622-C03-03, Ingenieros Alborada IDI under grant TRA2009-0309, and the JUNTA de ANDALUCÍA (proyecto de excelencia) ref. TIC-6114.

the same pattern and the user could help to discover and recover the semantic content of XML resources. This task is not easy and should be supported by tools. Such tools should be able to analyze the semantic content of XML data revealing fails on interpretation. In order to help tools in the interpretation, the user could provide the intended meaning of resources.

RDF(S) and OWL emerge as solutions for equipping Web data with semantic content. Unfortunately, most of database management systems do not offer exporting facilities to RDF(S) and OWL, although some efforts have been carried out (see [7] for a survey). XML has been usually adopted as database exchange format. For this reason, some authors have described how to map XML into RDF(S)/OWL and add semantic content to XML documents. On one hand, most proposals focus on the mapping of the XML schema into RDF(S)/OWL. In some cases, the mapping is exploited for reasoning (conformance and completion, among others). On the other hand, in many cases, the mapping, when the XML schema is present, can be automatically accomplished, however, some of them work when the XML schema is missing, requiring the user intervention, who specifies a set of mapping rules. Finally, in most of cases tools have been developed with this end, based on XML transformation and query languages like XSLT and XPath.

In this paper we describe how to analyze the semantic content of XML resources. With this aim, we map XML documents into an OWL ontology. In addition, we make use of SWRL for adding semantic content to XML documents, and for expressing restrictions imposed on XML data. Our approach has been implemented and tested. The implementation is based on a semantic web library for XQuery which includes primitives for mapping XML into OWL, and a SWRL engine.

Our proposal aims to provide XML programmers, which are habituated to XQuery, with a tool for transforming XML resources to OWL and specifying XML analysis rules with SWRL. Mappings and SWRL rules are embedded in XQuery code, and can be triggered. The tool is based on a semantic web library for XQuery which can be used from any XQuery interpreter, and includes a SWRL engine on top of XQuery. Such semantic web library has been already used for ontology querying in a previous work [2].

We will illustrate our approach with an example that shows that user intervention is vital to XML completion and SWRL helps to detect relevant data constraints. The semantic web library for XQuery together with the running example of the paper can be downloaded from `http://indalog.ual.es/XQuerySemanticWeb`.

1.1 Contributions

We can summarize the main contributions of our proposal as follows:

- XML to OWL mapping is carried out by specifying mappings from tags and attributes into concepts, roles and individuals of the ontology. Such mappings are defined with XPath, and XQuery is used for generating the

target ontology. The target ontology is not required to be previously defined. The ontology is created from the mapping, firstly, at instance level, and secondly, by adding ontology axioms with SWRL.
- SWRL is used for two main tasks: (a) to add new semantic information to the ontology instance generated from the XML document. Such information can be considered as a completion of the XML model. In particular, such rules can create additional classes and roles, and therefore extending the ontology initially created from the mapping with new axioms and instance relations. In other words, SWRL serves as ontology definition language in the phase of completion; and (b) to express data constraints on the XML document. SWRL can be used for expressing relations that the ontology instance has to satisfy. Therefore SWRL is a vehicle for reasoning with the semantic content and therefore for analyzing XML resources. SWRL rules can be specified and triggered with XQuery.
- XQuery is taken as transformation language in which mappings and rules can be defined in the code and triggered from any XQuery interpreter.

The advantages of our approach are the following. Our approach aims to provide a tool for specifying a transformation rather than to consider automatic mapping from the XML Schema. Firstly, transformations are defined with XQuery. It means that the most popular XML query language is used for the mapping. Secondly, XML completion and data constraints are specified with SWRL. Thus, a very simple rule language, integrated with OWL, is used for semantic completion and reasoning. The semantic completion of the XML document can be mapped into a semantic completion of the corresponding ontology. Besides, SWRL serves for specifying and reasoning with data constraints.

The drawbacks of our approach are the following. SWRL has a limited expressivity. Since we create the target ontology from the transformation and with SWRL, OWL meta-data axioms are defined with SWRL rules. In other words, SWRL is taken as ontology definition language in the line of [8,9]. SWRL has a limited expressivity and therefore there are some completions/data constraints that cannot be specified. In particular, those involving universally quantified constraints cannot be specified. One feasible extension of our work is to move to SQWRL [12], which extends SWRL to a more powerful query language. It is considered as future work.

With regard to the choice of SWRL as ontology definition language, we could express completions/data constraints with a suitable OWL 2 fragment (EL, RL and QL). However, in such a case, we would also have a limited expressivity. Furthermore, it would require the use of an external (from XQuery point of view) reasoner. Our framework embeds SWRL specification and reasoning in XQuery, that is, the programmer defines with XQuery the mapping and the SWRL rules to be applied, and (s)he can examine the result of the analysis from XQuery. Nevertheless, it does not mean that our approach can be integrated with existent OWL tools. That is, XQuery could be used for the specification of the mapping and the SWRL rules. Once they have been specified, an OWL document can be obtained from XQuery, containing the OWL ontology and the

SWRL rules in XML format, which can be imported from an external OWL tool to analyze XML documents. We have also tested our approach with the Protégé tool using the Hermit reasoner for triggering SWRL rules.

1.2 Related Work

XML Schema mapping into RDF(S)/OWL has been extensively studied. In an early work [5] the XML Schema is mapped into RDF(S) meta-data and individuals. In [4], they propose an small set of mapping rules from XML Schema into OWL and define a transformation with XSLT. In [16], they also propose an XSLT based transformation from the XML Schema into OWL that allows the inverse transformation, i.e., convert individuals from OWL to XML. The inverse transformation is usually called *lowering*, and the direct one is called *lifting*. Besides, in [19], XML schemas are mapped into RDF(S) and they make use of the mapping for query processing in the context of data integration and interoperation. This is also the case of [3], in which the authors propose a set of patterns for automatically transforming an XML schema into OWL. Such patterns are obtained from pattern recognition. Manual mapping has been considered in [15], which translates XML schemas and data into RDF(S) with the user intervention.

In some cases the target ontology has to be present, that is, the mapping from XML into OWL aims to populate the ontology with individuals. For instance, the proposals [14,1,13,17] transform XML resources into RDF(S) and OWL format, in the presence of an existing ontology and with user intervention. The mapping rules can be specified in Java (for instance, in [14]), and in domain specific languages, for instance, XSPARQL [1], a language which combines XQuery and SPARQL. Manual mappings are, in many cases, based on XPath expressions. In [6], they describe how to map XML documents into an ontology, using XPath for expressing the location in the XML document of OWL concepts and roles. In [11], they employ XPath to map XML documents into OWL, extending the set of OWL constructors to represent XML resources. Besides, XPath is used to describe mappings in the XSPARQL framework [1].

Finally, some authors have explored how to exploit the mapping for XML analysis. For instance, in [18], they describe how to map the XML Schema into Description Logic (DL) and make use of DL for model validation and completion, and as query language. Besides, in [20], they map XML into OWL (and DL) and they study how to reason on the XML schema, in the sense of how to check conformance of XML documents, and to prove inclusion, equivalence and disjointness of XML schemas. SWRL has been employed in [10] for data normalization, encoding schemas and functional dependences by means of DL.

Comparing our work with existent proposals we find some similarities with [1]. Here we make use of XQuery as transformation language which embeds SWRL rules as part of the code. Manual mapping is achieved with XPath. In [1] they embed SPARQL code in XQuery code and study lifting and lowering with manual mapping. There are also similarities with the work [11] in which they employ XPath for manual mapping and a rich OWL fragment for describing concept and role relations.

```
<?xml version='1.0'?>
<conference>
<papers>
<paper id="1" studentPaper="true">
<title> XML Schemas </title>
<wordCount> 1200 </wordCount>
</paper>
<paper id="2"  studentPaper="false">
<title> XML and OWL </title>
<wordCount> 2800 </wordCount>
</paper>
<paper id="3" studentPaper="true">
<title> OWL and RDF </title>
<wordCount> 12000 </wordCount>
</paper>
</papers>
<researchers>
<researcher id="a" isStudent="false" manuscript="1" referee="1">
<name>Smith </name>
</researcher>
<researcher id="b" isStudent="true" manuscript="1" referee="2">
<name>Douglas </name>
</researcher>
<researcher id="c" isStudent="false" manuscript="2" referee="3">
<name>King </name>
</researcher>
<researcher id="d" isStudent="true" manuscript="2" referee="1">
<name>Ben </name>
</researcher>
<researcher id="e" isStudent="false" manuscript="3" referee="3">
<name>William </name>
</researcher>
</researchers>
</conference>
```

Fig. 1. Running example

1.3 Structure of the Paper

The structure of the paper is as follows. Section 2 will present a running example. Section 3 will describe the developed semantic web library for XQuery which is used for defining mapping and SWRL rules. Section 4 will show how to accomplish XML analysis, and present the evaluation of the approach. Section 5 will conclude and present future work.

2 Running Example

Let us suppose that we have the XML resource of Figure 1. The document lists *paper*s and *researcher*s involved in a *conference*. Each *paper* and *researcher* has an identifier (represented by the attribute *id*), and has an associated set of labels: *title* and *wordCount* for *paper*s and *name* for *researcher*s. Furthermore, they have attributes *studentPaper* for *paper*s and *isStudent*, *manuscript* and *referee* for *researcher*s. The meaning of *manuscript* and *referee* is that the given researcher has submitted the paper of number described by *manuscript* as well as has participated as reviewer of the paper of number given by *referee*. It is worth observing that we have taken identifiers for cross references between papers and researches but it is just for simplifying the example, given that it is not real restriction of our approach. It does not mean that interesting examples come from resources when cross references are given.

Now, let us suppose that we would like to analyze the semantic content of the XML document. We would like to know whether some paper violates the restriction on the number of words of the submission. In order to do this we could execute an XPath query of the style:

```
/conference/papers/paper[wordCount>10000]
```

and it gives us the papers whose *wordCount* is greater than 10000. This is a typical restriction that can be analyzed with XPath. However, we can complicate the situation when papers are classified as student and senior papers which have different restrictions on length. Fortunately, each paper has been labeled with this information, that is, *studentPaper*. Nevertheless, it is redundant in the document. That is, we have information about submitters and whether they are students or not. In the case papers are not labeled with the attribute *studentPaper*, the XPath query becomes more complex. In general, missing and redundant information in XML resources makes XPath analysis complex to make.

The idea of our approach is to be able to extract from the XML document the semantic content and analyze the content. In particular, it involves to analyze cross references. However, the process of extraction is guided by the user who knows (at least suspects) the meaning of the given labels and attributes. In XML Schema into OWL translations, such semantic information could not be specified in the document therefore the user intervention is also required. The semantic analysis of XML documents ranges from restrictions imposed on data such as *wordCount* has to be smaller than 10000 to properly integrity constraints as the reviewer of a paper cannot be the submitter. Assuming that senior papers cannot have students as authors, inconsistent information comes from senior papers having an student as author.

Analysis can be improved by completing the XML model. Completion means to add new information that can be deduced from the original resource. For instance, in the running example, we can add *author* for each *paper*, which can be obtained as the inverse of the value *manuscript* of *researcher*. In the case *studentPaper* was not included, we can add this information from the information about authors. However, the semantic extraction can be more accurate. For instance, we can define classes *Student*, *Senior* as subclasses of *Researcher* and *PaperofSenior* and *PaperofStudent* as subclasses of *Paper*. *PaperofSenior* class is defined as papers whose value *studentPaper* is false and *Student* can be defined as researchers whose value *isStudent* is true.

The definition of such ontology completion facilitates the description of data constraints. For instance, *author*s of *PaperofSenior* cannot be *Student*s. In order to express data constraints we have adopted a simple solution in our approach. We define new ontology classes that represent data constraints. For instance the class *BadPaperCategory* can be defined as the class of senior papers having

```
declare function sw:toClass($c as xs:string) as node()*{
<owl:Class rdf:about="{$c}"/>
};
declare function sw:toProperty($p as xs:string) as node()*{
<rdf:Property rdf:about="{$p}"/>
};
declare function sw:toDataProperty($x as xs:string) as node()*{
<owl:DatatypeProperty rdf:about="{$x}" />
};
declare function sw:toIndividual($x as xs:string) as node()*{
<owl:Thing rdf:about="{$x}"/>
};
declare function sw:toClassFiller($x as xs:string, $c as xs:string)
    as node()*{
<owl:Thing rdf:about="{$x}">
<rdf:type rdf:resource="{$c}"/>
</owl:Thing>
};
declare function sw:toObjectFiller($x as xs:string, $p as xs:string,
    $y as xs:string) as node()*{
<owl:Thing rdf:about="{$x}">
{element {$p}
 {attribute
    {QName("http://www.w3.org/1999/02/22-rdf-syntax-ns#",'rdf:resource')}
    {$y}}
}
</owl:Thing>
};
declare function sw:toDataFiller($x as xs:string, $p as xs:string,
       $y as xs:string,$datatype as xs:string) as node()*{
<owl:Thing rdf:about="{$x}">
{element {$p}
 {attribute
    {QName("http://www.w3.org/1999/02/22-rdf-syntax-ns#",'rdf:datatype')}
    {concat("http://www.w3.org/2001/XMLSchema#",$datatype)}, $y}
}
</owl:Thing>
};
```

Fig. 2. Semantic Web Library: XQuery Functions for XML2OWL Mapping

an student as author, while *NoSelfReview* can be defined as the class of papers having a referee which is also the author of the paper. When *BadPaperCategory* and *NoSelfReview* are not empty, the XML document violates the required constraints.

3 A Semantic Web Library for XQuery

In this section we will describe the main elements of the semantic web library for XQuery. The library permits to map XML into OWL by providing XQuery functions for creating OWL elements. Moreover, the library includes XQuery functions to create and trigger SWRL rules.

Figure 2 shows the XQuery functions for mapping XML into OWL. The functions have as parameters strings which are obtained from XML elements (i.e., node names, attributes and text nodes). They build the corresponding OWL elements from the parameters. For instance, toClass(c) generates an OWL class c in

```
declare function swrle:swrl($ABox as node()*, $rules as node()*){
let $Classes := swrle:Classes($ABox) return
        let $Properties := swrle:Properties($ABox) return
<result>
{
for $Imp in $rules//swrl:Imp
        return swrle:evaluate($Classes,$Properties,$Imp)
}
</result>
};
declare function swrle:evaluate($Classes as node()*,
        $Properties as node()*, $Imp as node()){
for $Head in $Imp/swrl:head/swrl:AtomList return
        swrle:evaluate-rule($Classes,$Properties,$Imp/swrl:body,$Head)
};
declare function swrle:evaluate-rule($Classes as node()*,
        $Properties as node()*, $Body as node()*, $Head as node()*){
for $case in
        swrle:evaluate-body($Classes,$Properties,$Body,())
return
        swrle:evaluate-head($Head,$case)
};
```

Fig. 3. Semantic Web Library: SWRL engine

```
declare function swrle:equal($x as node(), $y as node())
      as  xs:boolean{
($x = $y)
};
declare function swrle:notEqual($x as node(), $y as node())
     as  xs:boolean{
not ($x = $y)
};
declare function swrle:lessThan($x as xs:float, $y as xs:float)
     as xs:boolean{
$x < $y
};
declare function swrle:greaterThanOrEqual($x as xs:float, $y as xs:float)
     as xs:boolean{
$x >= $y
};
```

Fig. 4. Semantic Web Library: SWRL Built-ins

XML/RDF format, toProperty(p) generates an OWL property p, and toIndividual(x) generates an OWL individual x. The functions toClassFiller(c,x), toObjectFiller(x,p,y) and toDataFiller(x,p,y,d) generate OWL class $C(x)$, object property $p(x, y)$ and data property $p(x, y)$ of type d instances, respectively.

Figure 3 shows the main functions of the XQuery implementation of SWRL. The function swrl can be called to trigger a given rule (and rules) w.r.t. a given **ABox**. The result of the execution of the SWRL rules is obtained in OWL (XML/RDF) format. The function evaluate calls to evaluate-rule which at the same time calls to evaluate-body and evaluate-head. The implementation of the SWRL engine is based on the representation of SWRL with XML/RDF. In order to trigger SWRL rules, the XQuery code traverses the XML representation

```
declare function swrle:ClassAtom($c as xs:string,
       $x as xs:string){
<swrl:ClassAtom>
        <swrl:classPredicate rdf:resource="{$c}"/>
        <swrl:argument1 rdf:resource="{$x}"/>
</swrl:ClassAtom>
};
declare function swrle:IndividualPropertyAtom($p as xs:string,
       $x as xs:string, $y as xs:string){
 <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="{$p}"/>
        <swrl:argument1 rdf:resource="{$x}"/>
        <swrl:argument2 rdf:resource="{$y}"/>
 </swrl:IndividualPropertyAtom>
};
declare function swrle:Imp($Antecedent as node()*,
        $Consequent as node()*){
<swrl:Imp>
        <swrl:body>
             {$Antecedent}
        </swrl:body>
        <swrl:head>
             {$Consequent}
        </swrl:head>
</swrl:Imp>
};
```

Fig. 5. Semantic Web Library: SWRL definition

of SWRL, matching SWRL atoms with the **ABox**, and variable bindings are represented and stored in XML format and passed as parameter to be used for the matching.

Figure 4 shows how to define SWRL built-ins in XQuery. Each built-in is defined by a XQuery function. Therefore the SWRL library can be easily extended with new built-ins. Finally, Figure 5 shows how to define SWRL rules in XQuery code. The main function is `Imp` which builds an SWRL rule of the form "Antecedent $\rightarrow$ Consequent". Each SWRL atom is built from an XQuery function. For instance, ClassAtom(#c,#x) and IndividualPropertyAtom(#p,#x,#y) build SWRL atoms for class $C(?x)$ and object properties $p(?x, ?y)$.

4 Using XQuery for XML Analysis

Our approach aims to provide a methodology for XML analysis. Our proposal distinguishes three steps:

(1) The first step consists in the XML into OWL mapping. Mapping rules are expressed by XQuery expressions and map XPath expressions into ontology concepts

$$xp_1 \mapsto C_1, \ldots, xp_n \mapsto C_n$$

and pairs of XPath expressions to ontology roles

$$(xp'_1, xp''_1) \mapsto r_1 \ldots, (xp'_m, xp''_m) \mapsto r_m$$

The mapping works at the instance level, creating instances of concepts and roles from XML items. Let us remark that our approach does not require an existent ontology to work.

For instance, let us suppose that the programmer wants to transform the running example. Firstly, the programmer has to define paths to access to individuals and property values:

```
(a) doc('papers.xml')//researchers/researcher/@id
(b) doc('papers.xml')//researchers/researcher/name
(c) doc('papers.xml')//researchers/researcher/@isStudent
(d) doc('papers.xml')//researchers/researcher/@manuscript
(e) doc('papers.xml')//researchers/researcher/@referee
```

and maps them into ontology concepts and roles

$$\mathsf{a} \mapsto \mathsf{Researcher}$$
$$(\mathsf{a},\mathsf{b}) \mapsto \mathsf{name}$$
$$(\mathsf{a},\mathsf{c}) \mapsto \mathsf{isStudent}$$
$$(\mathsf{a},\mathsf{d}) \mapsto \mathsf{manuscript}$$
$$(\mathsf{a},\mathsf{e}) \mapsto \mathsf{referee}$$

In order to define the mapping the programmer can use the `for` construction of XQuery as follows:

```
let $ontology1 :=
(for $y in doc('papers.xml')//researchers/researcher return
   sw:toClassFiller(sw:ID($y/@id),"#Researcher")
   union
   sw:toDataFiller(sw:ID($y/@id),"name",$y/name,"xsd:string")
   union
   sw:toDataFiller(sw:ID($y/@id),"isStudent",$y/@isStudent,"xsd:
        boolean")
   union
   sw:toObjectFiller(sw:ID($y/@id),"manuscript",sw:ID($y/@manuscript))
   union
   sw:toObjectFiller(sw:ID($y/@id),"referee",sw:ID($y/@referee)))
return $ontology1
```

The same can be done from papers (see appendix). Let us remark that a `for` expression is considered for each item of the XML document. In the example one for doc('papers.xml')//researchers/researcher, and one for doc('papers.xml')//papers/paper.

(2) The second step consists in the XML completion using SWRL rules. The completion is defined in terms of the mapped OWL ontology. SWRL rules are used to define new concepts and roles $C'_1, \ldots, C'_s$, $r'_1, \ldots, r'_l$ from $C_1, \ldots, C_n$ and $r_1, \ldots, r_m$. Completion aims to structure the semantic information and to infer new information.

For instance, let us suppose that the programmer wants to specify the completion of the running example by defining the concepts *PaperofSenior* as subclass of *Paper*, and *Student* as subclass of *Researcher*:

```
PaperofSenior(?x)->Paper(?x)
studentPaper(?x,false)->PaperofSenior(?x)
Student(?x)->Researcher(?x)
isStudent(?x,true)->Student(?x)
```

Moreover, the programmer defines the inverse relations of *manuscript* and *referee* (which defines as *author* and *submission*), as follows:

```
manuscript(?x,?y)->author(?y,?x)
referee(?x,?y)->submission(?y,?x)
```

Let us remark that SWRL is used for describing meta-data relationships, and some of them correspond to OWL 2 relationships. For instance, $\mathsf{PaperofSenior} \sqsubseteq \mathsf{Paper}$, $\exists \mathsf{studentPaper}.\{\mathsf{false}\} \sqsubseteq \mathsf{PaperofSenior}$, and $\mathsf{manuscript}^{-} \sqsubseteq \mathsf{author}$. However, we make use of SWRL as ontology definition language for expressing ontology relationships.

Now, the SWRL rules can be defined with XQuery code. For instance, *author*, *submission* and *Student* rules are defined as follows:

```
let $rule1 := swrle:Imp(
     swrle:AtomList(
          swrle:IndividualPropertyAtom("#referee","#x","#y")),
     swrle:AtomList(
          swrle:IndividualPropertyAtom("#submission","#y","#x")))
let $rule2 := swrle:Imp(
     swrle:AtomList(
          swrle:IndividualPropertyAtom("#manuscript","#x","#y")),
     swrle:AtomList(
          swrle:IndividualPropertyAtom("#author","#y","#x")))
let $rule3 := swrle:Imp(
     swrle:AtomList(
          swrle:DatavaluedPropertyAtomValue("#isStudent","#x","true",
               "xsd:boolean")),
     swrle:AtomList(
          swrle:ClassAtom("#Student","#x")))
let $ruleall := $rule1 union $rule2 union $rule3
return $ruleall
```

Now, the mapping together with the completion rules can be triggered as follows:

```
let $mapping := $ontology1 union $ontology2
let $completion :=
 swrle:swrl(<rdf:RDF> {$mapping} </rdf:RDF>,
                              <rdf:RDF> {$ruleall} </rdf:RDF>)
```

Assuming $ontology1 and $ontology2 are the definitions of paper and researcher ontologies, respectively.

(3) The last step consists in the definition of data constraints with SWRL. New concepts are defined from $C_1, \ldots, C_n$, $C'_1, \ldots, C'_s$, $r_1, \ldots, r_m$ and $r'_1, \ldots, r'_l$, and individuals of such concepts violate data constraints.

For instance, in the running example, the programmer can define the concepts *PaperLength*, *NoSelfReview*, *NoStudentReview* and *BadPaperCategory* with the following rules:

```
wordCount(?x,?y),greaterThanOrEqual(?y,10000)->PaperLength(?x)
manuscript(?x,?y),submission(?x,?y)->NoSelfReview(?x)
Student(?x),submission(?x,?y)->NoStudentReviewer(?x)
manuscript(?x,?y),isStudent(?x,true),studentPaper(?y,false)
                                                ->BadPaperCategory(?x)
author(?x,?y),isStudent(?y,true),studentPaper(?x,false)
                                                ->BadPaperCategory(?x)
```

which can be specified with XQuery code as follows:

```
let $integrity1 := swrle:Imp(
      swrle:AtomList((
          swrle:DatavaluedPropertyAtomVars("#wordCount","#x","#y"),
          swrle:BuiltinAtomArg2("swrlb:greaterThanOrEqual","#y
              ","10000","xsd:integer"))),
      swrle:AtomList(
          swrle:ClassAtom("#PaperLength","#x")))
let $integrity2 := swrle:Imp(
      swrle:AtomList((
          swrle:IndividualPropertyAtom("#manuscript","#x","#y"),
          swrle:IndividualPropertyAtom("#submission","#x","#y"))),
      swrle:AtomList(
          swrle:ClassAtom("#NoSelfReview","#x")))
let $integrity3 := swrle:Imp(
      swrle:AtomList((
          swrle:ClassAtom("#Student","#x"),
          swrle:IndividualPropertyAtom("#submission","#x","#y"))),
      swrle:AtomList(
          swrle:ClassAtom("#NoStudentReviewer","#x")))
let $integrity4 := swrle:Imp(
      swrle:AtomList((
        swrle:IndividualPropertyAtom("#manuscript","#x","#y"),
        swrle:DatavaluedPropertyAtomValue("#isStudent","#x","true",
              "xsd:boolean"),
        swrle:DatavaluedPropertyAtomValue("#studentPaper","#y","false",
              "xsd:boolean"))),
      swrle:AtomList(
        swrle:ClassAtom("#BadPaperCategory","#x")))
let $integrity5 := swrle:Imp(
      swrle:AtomList((
        swrle:IndividualPropertyAtom("#author","#x","#y"),
        swrle:DatavaluedPropertyAtomValue("#isStudent","#y","true",
            "xsd:boolean"),
        swrle:DatavaluedPropertyAtomValue("#studentPaper","#x","false",
            "xsd:boolean"))),
      swrle:AtomList(
        swrle:ClassAtom("#BadPaperCategory","#x")))
let $integrityall := $integrity1 union $integrity2 union $integrity3
     union $integrity4 union $integrity5
return $integrityall
```

Finally, they can triggered as follows:

```
swrle:swrl(<rdf:RDF> {$mapping union $completion} </rdf:RDF>,
                <rdf:RDF> {$integrityall} </rdf:RDF>)
```

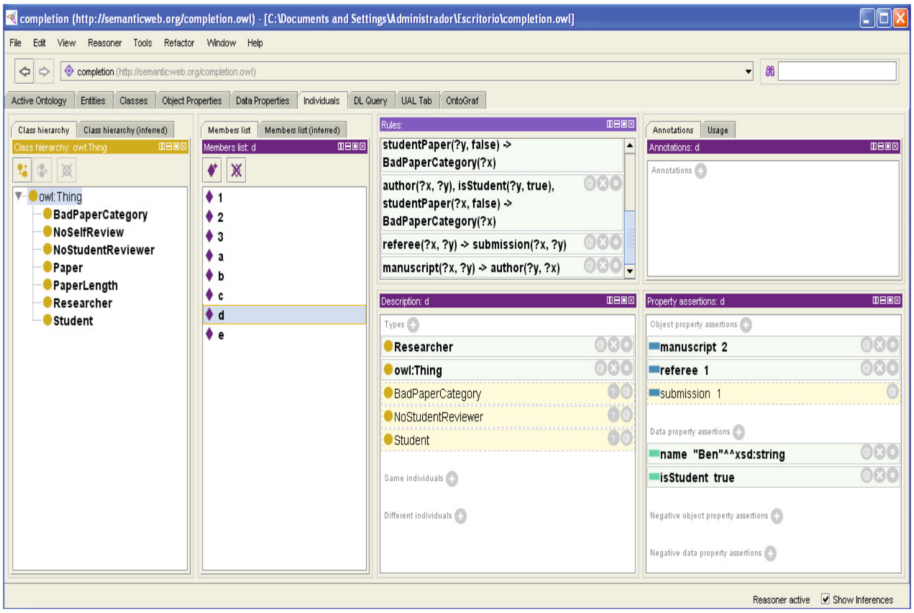
obtaining the following results:

```
<result>
    <owl:Thing  rdf:about="#3">
      <rdf:type rdf:resource="#PaperLength"/>
    </owl:Thing>
    <owl:Thing   rdf:about="#a">
      <rdf:type rdf:resource="#NoSelfReview"/>
    </owl:Thing>
    <owl:Thing  rdf:about="#e">
      <rdf:type rdf:resource="#NoSelfReview"/>
    </owl:Thing>
    <owl:Thing   rdf:about="#b">
      <rdf:type rdf:resource="#NoStudentReviewer"/>
    </owl:Thing>
    <owl:Thing   rdf:about="#d">
      <rdf:type rdf:resource="#NoStudentReviewer"/>
    </owl:Thing>
    <owl:Thing  rdf:about="#d">
      <rdf:type rdf:resource="#BadPaperCategory"/>
    </owl:Thing>
    <owl:Thing  rdf:about="#2">
      <rdf:type  rdf:resource="#BadPaperCategory"/>
    </owl:Thing>
</result>
```

The result shows that paper length is exceeded by paper `#3`, authors of `#a` and `#e` have reviewed their own paper, `#b` and `#d` are students that review a paper, and finally the researcher `#d` is an student with a senior paper and `#2` is a senior paper with an student as author. A full version of the code of transformation can be found in the appendix.

The mapping of XML into OWL together with completion rules and data constraints provide a semantic description of the content of the XML document. Our approach permits, in particular, to generate an OWL document from them.

Size	*Mapping/Size*	*Completion/Size*	*XML Analysis*
100Kb	78ms(648Kb)	338ms(884Kb)	181 ms
1MB	598ms(6,3MB)	6750ms(8,7MB)	4697 ms
5MB	2534ms(31,8MB)	4m 45s(43,5MB)	2m 27s
10MB	5066ms(63,6MB)	30m 13sg(90MB)	12m 4s


completion (http://semanticweb.org/completion.owl) - [C:\Documents and Settings\Administrador\Escritorio\completion.owl]

Class hierarchy: owl:Thing
- owl:Thing
  - BadPaperCategory
  - NoSelfReview
  - NoStudentReviewer
  - Paper
  - PaperLength
  - Researcher
  - Student

Members list: d
- 1
- 2
- 3
- a
- b
- c
- d
- e

Rules:
- studentPaper(?y, false) -> BadPaperCategory(?x)
- author(?x, ?y), isStudent(?y, true), studentPaper(?x, false) -> BadPaperCategory(?x)
- referee(?x, ?y) -> submission(?x, ?y)
- manuscript(?x, ?y) -> author(?y, ?x)

Description: d
- Types: Researcher, owl:Thing, BadPaperCategory, NoStudentReviewer, Student

Property assertions: d
- Object property assertions: manuscript 2, referee 1, submission 1
- Data property assertions: name "Ben"^^xsd:string, isStudent true

File Edit View Reasoner Tools Refactor Window Help
completion (http://semanticweb.org/completion.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals DL Query UAL Tab OntoGraf
Class hierarchy Class hierarchy (inferred)
Members list Members list (inferred)
Annotations Usage
Annotations: d
Annotations
Same individuals
Different individuals
Negative object property assertions
Negative data property assertions
Reasoner active Show Inferences

Fig. 6. Visualization of Analysis in Protegé

4.1 Evaluation

We have tested our approach with the `BaseX` XQuery interpreter in an Intel 2.66 GHz machine under Mac OS X operating system, comparing runtime execution on several file sizes: 100 KB, 1MB, 5MB and 10MB. The mapping from XML into OWL works in a reasonable time for the considered sizes. Completion and XML analysis require more time due to the size of document obtained from the mapping and completion procedure, respectively. In order to give a comparative of times for XML analysis we have considered a single SWRL rule in the XML analysis. Times and sizes of documents are given for each phase in the following table:

We have also tested our approach with the Protégé tool (version 4.1) using the Hermit reasoner (version 1.3.4) The Hermit reasoner has been used for triggering the completion rules and data constraints from the mapping of XML into OWL. Figure 6 shows an snapshot of the analysis results obtained with Hermit.

5 Conclusions and Future Work

In this paper we have studied how to analyze XML documents from a mapping into OWL and the use of SWRL. We have presented a semantic web library for XQuery which allows to specify in XQuery code XML to OWL transformations, and SWRL rules. Such specification serves as a procedure for analyzing XML documents from the semantic content. We have described how to complete XML/OWL models and how to specify data constraints with SWRL. In conclusion, our framework enables to a XQuery programmer to write his(er) own transformations and completions, and data constraints.

As future work, we would like to extend our work in the following directions. Firstly, we could move to a more expressive language, SQWRL, in order to be able to express more complex data constraints. Secondly, we can study how to map XML into OWL by using the XML Schema. Fortunately XML Schema is described by XML itself and therefore the integration in our framework is possible. Finally, we would like to integrate our proposal with the Protégé tool. We have in mind the development of a Protégé plugin for the edition and execution of XQuery transformations in Protégé. Such integration would allow to analyze XML documents in Protégé, and the use of OWL constructors/SWRL for specifying completions/data constraints.

References

1. Akhtar, W., Kopecký, J., Krennwallner, T., Polleres, A.: XSPARQL: Traveling between the XML and RDF Worlds – and Avoiding the XSLT Pilgrimage. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 432–447. Springer, Heidelberg (2008)
2. Almendros-Jiménez, J.M.: Querying and Reasoning with RDF(S)/OWL in XQuery. In: Du, X., Fan, W., Wang, J., Peng, Z., Sharaf, M.A. (eds.) APWeb 2011. LNCS, vol. 6612, pp. 450–459. Springer, Heidelberg (2011)

3. Bedini, I., Matheus, C., Patel-Schneider, P., Boran, A., Nguyen, B.: Transforming xml schema to owl using patterns. In: 2011 Fifth IEEE International Conference on Semantic Computing (ICSC), pp. 102–109. IEEE (2011)
4. Bohring, H., Auer, S.: Mapping XML to OWL ontologies. Leipziger Informatik-Tage 72, 147–156 (2005)
5. Ferdinand, M., Zirpins, C., Trastour, D.: Lifting XML Schema to OWL. In: Koch, N., Fraternali, P., Wirsing, M. (eds.) ICWE 2004. LNCS, vol. 3140, pp. 354–358. Springer, Heidelberg (2004)
6. Ghawi, R., Cullot, N.: Building Ontologies from XML Data Sources. In: 20th International Workshop on Database and Expert Systems Application, DEXA 2009, pp. 480–484. IEEE (2009)
7. Konstantinou, N., Spanos, D., Mitrou, N.: Ontology and database mapping: A survey of current implementations and future directions. Journal of Web Engineering 7(1), 1–24 (2008)
8. Krisnadhi, A., Maier, F., Hitzler, P.: OWL and Rules. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 382–415. Springer, Heidelberg (2011)
9. Krötzsch, M., Rudolph, S., Hitzler, P.: Description Logic Rules. In: Proceeding of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence, pp. 80–84. IOS Press (2008)
10. Li, Y., Sun, J., Dobbie, G., Lee, S., Wang, H.: Verifying semistructured data normalization using SWRL. In: Third IEEE International Symposium on Theoretical Aspects of Software Engineering, TASE 2009, pp. 193–200. IEEE (2009)
11. O'Connor, M., Das, A.: Acquiring OWL ontologies from XML documents. In: Proceedings of the Sixth International Conference on Knowledge Capture, pp. 17–24. ACM (2011)
12. O'Connor, M., Das, A.: SQWRL: a query language for OWL. In: Fifth International Workshop on OWL: Experiences and Directions (OWLED) (2009)
13. Rodrigues, T., Rosa, P., Cardoso, J.: Mapping XML to Exiting OWL ontologies. In: International Conference WWW/Internet, pp. 72–77 (2006)
14. Rodrigues, T., Rosa, P., Cardoso, J.: Moving from syntactic to semantic organizations using JXML2OWL. Computers in Industry 59(8), 808–819 (2008)
15. Thuy, P., Lee, Y., Lee, S., Jeong, B.: Exploiting XML Schema for Interpreting XML Documents as RDF. In: IEEE International Conference on Services Computing, SCC 2008, vol. 2, pp. 555–558. IEEE (2008)
16. Tsinaraki, C., Christodoulakis, S.: XS2OWL: A Formal Model and a System for Enabling XML Schema Applications to Interoperate with OWL-DL Domain Knowledge and Semantic Web Tools. In: Thanos, C., Borri, F., Candela, L. (eds.) Digital Libraries: R&D. LNCS, vol. 4877, pp. 124–136. Springer, Heidelberg (2007)
17. Van Deursen, D., Poppe, C., Martens, G., Mannens, E., Walle, R.: XML to RDF conversion: a Generic Approach. In: International Conference on Automated solutions for Cross Media Content and Multi-channel Distribution, AXMEDIS 2008, pp. 138–144. IEEE (2008)
18. Wu, X., Ratcliffe, D., Cameron, M.: XML Schema Representation and Reasoning: A Description Logic Method. In: IEEE Congress on Services-Part I, 2008, pp. 487–494. IEEE (2008)
19. Xiao, H., Cruz, I.: Integrating and Exchanging XML Data Using Ontologies. In: Spaccapietra, S., Aberer, K., Cudré-Mauroux, P. (eds.) Journal on Data Semantics VI. LNCS, vol. 4090, pp. 67–89. Springer, Heidelberg (2006)
20. Zhang, F., Yan, L., Ma, Z., Cheng, J.: Knowledge representation and reasoning of XML with ontology. In: Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 1705–1710. ACM (2011)

Appendix I: Example of Transformation

```
let $classes := sw:toClass("#Paper") union sw:toClass("#Researcher")
let $properties := sw:toProperty("#manuscript") union
          sw:toProperty("#referee")
let $dataproperties := sw:toDataProperty("#studentPaper") union
               sw:toDataProperty("#title") union
               sw:toDataProperty("#wordCount") union
               sw:toDataProperty("#name") union
               sw:toDataProperty("#isStudent")
let $name := /conference
let $ontology1 :=
    (for $x in $name/papers/paper return
       sw:toClassFiller(sw:ID($x/@id),"#Paper") union
       sw:toDataFiller(sw:ID($x/@id),"studentPaper",$x/@studentPaper,
           "xsd:boolean") union
       sw:toDataFiller(sw:ID($x/@id),"title",$x/title,"xsd:string") union
       sw:toDataFiller(sw:ID($x/@id),"wordCount",$x/wordCount,
           "xsd:integer"))
let $ontology2 :=
     (for $y in $name/researchers/researcher return
        sw:toClassFiller(sw:ID($y/@id),"#Researcher") union
        sw:toDataFiller(sw:ID($y/@id),"name",$y/name,"xsd:string") union
        sw:toDataFiller(sw:ID($y/@id),"isStudent",$y/@isStudent,
              "xsd:boolean") union
        sw:toObjectFiller(sw:ID($y/@id),"manuscript",
              sw:ID($y/@manuscript)) union
        sw:toObjectFiller(sw:ID($y/@id),"referee",sw:ID($y/@referee)))
return
let $mapping := $classes union $properties union $dataproperties union
                           $ontology1 union $ontology2
let $variables := swrle:variable("#x") union swrle:variable("#y")
let $classes_rules := sw:toClass("#Student")
let $rule1 := swrle:Imp(
      swrle:AtomList(
           swrle:IndividualPropertyAtom("#referee","#x","#y")),
      swrle:AtomList(
           swrle:IndividualPropertyAtom("#submission","#x","#y")))
let $rule2 := swrle:Imp(
      swrle:AtomList(
           swrle:IndividualPropertyAtom("#manuscript","#x","#y")),
      swrle:AtomList(
           swrle:IndividualPropertyAtom("#author","#y","#x")))
let $rule3 := swrle:Imp(
      swrle:AtomList(
           swrle:DatavaluedPropertyAtomValue("#isStudent","#x","true",
                "xsd:boolean")),
      swrle:AtomList(
           swrle:ClassAtom("#Student","#x")))
let $ruleall := $variables union $classes_rules  union
                        $rule1 union $rule2 union $rule3
return
let $completion := swrle:swrl(<rdf:RDF> {$mapping} </rdf:RDF>,
                     <rdf:RDF> {$ruleall} </rdf:RDF>)
return
let $classes_rules2 :=
        sw:toClass("#PaperLength") union
        sw:toClass("#NoSelfReview") union
        sw:toClass("#NoStudentReviewer") union
        sw:toClass("#BadPaperCategory")
let $integrity1 :=
    swrle:Imp(
     swrle:AtomList((
       swrle:DatavaluedPropertyAtomVars("#wordCount","#x","#y"),
       swrle:BuiltinAtomArg2("swrlb:greaterThanOrEqual","#y","10000",
                 "xsd:integer"))),
     swrle:AtomList(
       swrle:ClassAtom("#PaperLength","#x")))
```

```
let $integrity2 :=
       swrle:Imp(
           swrle:AtomList((
               swrle:IndividualPropertyAtom("#manuscript","#x","#y"),
               swrle:IndividualPropertyAtom("#submission","#x","#y"))),
           swrle:AtomList(
               swrle:ClassAtom("#NoSelfReview","#x")))
let $integrity3 :=
       swrle:Imp(
            swrle:AtomList((
              swrle:ClassAtom("#Student","#x"),
              swrle:IndividualPropertyAtom("#submission","#x","#y"))),
       swrle:AtomList(
              swrle:ClassAtom("#NoStudentReviewer","#x")))
let $integrity4 :=
   swrle:Imp(
     swrle:AtomList((
       swrle:IndividualPropertyAtom("#manuscript","#x","#y"),
         swrle:DatavaluedPropertyAtomValue("#isStudent","#x","true",
                "xsd:boolean"),
         swrle:DatavaluedPropertyAtomValue("#studentPaper","#y","false",
                "xsd:boolean"))),
      swrle:AtomList(
        swrle:ClassAtom("#BadPaperCategory","#x")))
let $integrity5 := swrle:Imp(
   swrle:AtomList((
     swrle:IndividualPropertyAtom("#author","#x","#y"),
       swrle:DatavaluedPropertyAtomValue("#isStudent","#y","true",
                 "xsd:boolean"),
       swrle:DatavaluedPropertyAtomValue("#studentPaper","#x","false",
           "xsd:boolean"))),
     swrle:AtomList(
       swrle:ClassAtom("#BadPaperCategory","#x")))
let $integrityall := $classes_rules2 union
       $integrity1 union $integrity2 union $integrity3 union
       $integrity4 union $integrity5
return
        swrle:swrl(<rdf:RDF> {$mapping union $completion} </rdf:RDF>,
                    <rdf:RDF> {$integrityall} </rdf:RDF>)
```