International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:2, 2014

# Restructuring of XML Documents in the Form of Ontologies

Jamal Bakkas, Mohamed Bahaj, Abdellatif Soklabi

***Abstract*—**The intense use of the web has made it a very changing environment, its content is in permanent evolution to adapt to the demands. The standards have accompanied this evolution by passing from standards that regroup data with their presentations without any structuring such as HTML, to standards that separate both and give more importance to the structural aspect of the content such as XML standard and its derivatives. Currently, with the appearance of the Semantic Web, ontologies become increasingly present on the web and standards that allow their representations as OWL and RDF/RDFS begin to gain momentum. This paper provided an automatic method that converts XML schema document to ontologies represented in OWL.

***Keywords*—** XML Schema, OWL, RDB, Mapping, Ontology.

## I. Introduction

CERTAINLY, the current web is a changing environment: the structure of the content and the used standards were deeply modified in such a short time. Indeed, the data structure according to the HTML standard was replaced by a structure based on the standard XML, the latter provides syntactic and structural description of the data. This structure is increasingly replaced by a restructuring based on standards such as RDF [1] and OWL [2], [3], which allows, among other things, to manage the heterogeneity of the data and their semantics, thereby giving rise to a web called "semantic web". The migration to this new web requires an enrichment of content by semantics.

In this paper, we present a new method that allows conversion at the schema level of XML documents to ontologies.

Indeed, an XML document uses other XS documents (. xsd) to define its structure. This document is called “valid” if its content complies with the structure described by the XS grammar. We will therefore be interested in a conversion of XML documents schema to ontologies. The particularity of this approach is to allow the classification of generated classes by converted elements and, unlike other methods that generate classes for most of elements XS, resulting classes of our ontology are classified in hierarchical form , each inherits from a super class that indicate its category.

What remains of this paper is organized as follows: Section II provides related work. A detailed description of the method is presented in Section III. Section IV is devoted to the implementation part. The conclusion will be the subject of the last section.

Jamal Bakkas, Mohamed Bahaj, and Abdellatif Soklabi are with University Hassan 1, Laboratory LITEN, 26000 Settat, Morocco (e-mail: jbakkas@gmail.com, mohamedbahaj@gmail.com, abd.soklabi@gmail.com).

## II. Related works

Several studies have emerged with aim of integrating XML content in the Semantic Web. These researches include:

The proposal of tool called "Janus" by Bedini et al [4], [5] which provides automatic derivation of ontologies from XS files by applying a set of derivation rules, somewhat later, the same group proposed a method based on patterns [6], ie a set of cases occurring in most documents. This proposal deals with 40 patterns, the conversion of each pattern to OWL language is provided.

Another proposal is that of Bohring et al [7]. The authors proposed an approach that covers both schema and data levels. Their method allows the conversion of XS documents to concepts and properties of ontology model, and in the data level, it allows to convert XML documents to instances (individuals), with possible XS files generation for an XML document that does not have a schema. This mapping is implemented by the authors using another XML standard which is XSLT [8].

N. Yahi et al have proposed an approach [9] that covers two levels: schema level and data level. Their methods use a set of Java libraries to generate ontologies from different XML data sources. Thus, the XMLSchema documents are generated for XML documents with no schema using the "Trang" (XML to XML Schema) library. The results of this step are parsed using XSOM library to provide a result that is taken as input by the JUNG API (Java Universal Network / Graph framework) to provide a graph XML-Schema graph (XSG), the latter in turn is passed as input to the Java API to build the final ontology.

Other studies propose a passage by an intermediate to convert XML into ontology: for instance the approach proposed by Xu et al [10]. They proposed the passage by relational databases by converting XML to RDB, and then generate ontology from RDB.

## III. Method Description

Our approach is spread over five main steps:

Preparation: In the first step we create an ontology that we called template ontology, which we use as template for the resulting ontology conversion. This ontology is initialized by set of classes and properties.

Extraction: The second step is to parse the document XS to be converted with an XML parser to extract its contents (elements, attributes, values, types ...).

Conversion: In this step we create the ontology elements

corresponding to the XS document, using the data extracted in the previous step.

Classification: Each concept or property created in the conversion step is positioned in its location in the template ontology; the objective of this operation is to differentiate the resulting concepts depending on the XS element converted.

Flattening: The last step is to flatten the result of the previous step, to get the classes of the ontology representing only elements "xs: element" xsd document.

In what remains of this section we present the classes and properties that compose the template ontology, which serves as container of the resulting ontology. Then we detail how each XS element is converted into its equivalent in ontology and how it is inserted into template ontology.

*A. Template Ontology*

The template ontology elements are:

The Element class is a class from which inherits all classes that represent elements with attributes and/or nested elements.

The ComplexeType class is a class from which inherits all classes that represent the xs:complexType elements.

The GroupElement class is a class from which inherits all classes that represent groups of elements. Recalled that the xs:group element used to group a set of elements under one name.

Template Ontology also contains a property called data type DATA, it is the super-property from which inherits all kinds of data contained in the document XS to convert. This property is extended by three additional sub-properties that are:

The ATTRIBUTE property: it is used to extend all converted properties from the attributes of the complex type elements.

TEXT property: it is used to extend converted properties from text content of the converted complex type elements.

The EmptyElement property: it is used to extend the converted properties from the elements with a simple type.

Once the template ontology is created, we parse the XS document, and each of its elements is converted and inserted in the template ontology provided in Fig. 1 according to the cases bellow:

*B. XSD Elements Conversion*

*1) Simple Element:*

A simple element is mapped directly to a data type property with the same name as the element. This property inherits from the property EmptyElement. The range of this property is the type of the XS element if it is specified, otherwise it takes the type xs:string

*2) Complex Element:*

Each xs:complexType element having a name generates a class named the same name as this type and inherits from the "ComplexeType" class. Then each element having this type is converted to a class having the same name as element, inherits from the super class "Element", and related with the class that represents the type by the "hasType_ComplexName" object property.

An anonymous complex type is normally created to define the structure of a single element; this type is mapped to a class whose name is formed by the name of the element like this: "CplxT_ElementName".

Let CT be an xs:complexType element, this element can contain nested elements and/or attributes:

If CT contains attributes, they are converted to data type properties that inherit from the property "ATTRIBUTE", each of them is named "hasAttributeName" with as domain the class that represents CT and as range the type XS of this attribute. If the attribute "use" with "required" value is used to define converted attribute, this case is mapped by a cardinality restriction securing it to 1. Otherwise, no restriction mentioned.

If CT contains elements, they are converted according to if they are simple or complex elements as indicated above.

If ElementEltName is a complex element, the class that represent this element is related to the class that represents CT by the transitive object property named "hasNestedElementEltName" to indicate the nesting of the elements, the domain of this property is the class that represents CT and its range is the class representing ElementEltName.

If an element refers to another element using the "ref" attribute; the classes representing these elements are related one to another by the object property "referToeReferencedElementName".

If ElementEltName element contains text, we propose to add a data type property that inherits from the property "TEXT" and bearing the name "hasEelemntName_Text". The domain of this property is the class that represents the element and its range is the XS type if it is specified in XS document, otherwise the type xs:string is used.

If the CT element contains a simple element, it is mapped to a data type property as described above; the domain of this property is the class that represents CT.

We gave great importance to the nomination of the various resources in order to ensure the uniqueness of their URI. We take into consideration that an XML document can have several elements with the same name whereas the resources of URI in the ontology must be different all.

The following example shows how to convert a complex type element containing a simple element and an attribute:

```
<xs:element name="eltName">
<xs:complexType>
   …
 <xs:attribute name="attributeName" type="xs:string"/>
  <xs:element name="subEltName" type="xs:dataType"/>
   …
</xs:complexType>
</xs:element>
```

The equivalent in ontology language is as follows:

```
<owl:Class rdf:ID="CplxT_EltName">
 <rdfs:subClassOf rdf:resource="#ComplexType" />
</owl:Class>
<owl:Class rdf:ID="eltName">
```

```
  <rdfs:subClassOf rdf:resource="#Element" />
</owl:Class>
<owl:ObjectProperty rdf:ID="hasType_ CplxT_EltName">
  <rdfs:domain rdf:resource="#eltName"/>
  <rdfs:range rdf:resource="#CplxT _EltName"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasNestedEltName">
  <rdfs:domain rdf:resource="#CplxT _EltName"/>
  <rdfs:range rdf:resource="#eltName"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="hasattributeName">
  <rdfs:domain rdf:resource="#CplxT_EltName" />
  <rdfs:range  rdf:resource="&xsd;dataType"/>
  <rdfs:subPropertyOf rdf:resource="#Attribute" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="subEltName">
  <rdfs:domain rdf:resource="#CplxT_EltName" />
  <rdfs:range  rdf:resource="&xsd;dataType"/>
  <rdfs:subPropertyOf rdf:resource="#emptyelement" />
</owl:DatatypeProperty>
```

A complex type element can be an extension of another complex type element in this case each of the two is converted to a class; these classes are related by an inheritance relationship as shown in the following example:

```
<xs:complexType name="cmplxName1">
  <xs:complexContent>
    <xs:extension base="cmplxName2">
          …
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

This case is converted as follows:

```
<owl:Class rdf:ID="cmplxName1">
  <rdfs:subClassOf rdf:resource="#cmplxName2"/>
            …
</owl:Class>
```

*3) Indicators:*

We distinguish seven indicators: order indicators such as All, Choice and Sequence; The occurrence indicators such as: minOccurs and maxOccurs. And indicators of group such as Group and attributeGroup

The order indicators: They indicate the order in which the elements appear. These indicators are mapped into RDF element containers. The table below shows for each order indicator the equivalent container.

TABLE I
EQUIVALENT RDF CONTAINER ELEMENTS TO THE INDICATORS OF ORDER

| *XML Schema* | *RDF* |
|---|---|
| xs:All | rdf:Bag |
| xs:Choice | rdf:Alt |
| xs:Sequence | rdf:Seq |

The occurrence indicators: The cardinality restrictions applied to an XS element are mapped to cardinality restrictions on the data type property that represents the element if the element is a simple type, otherwise, they are mapped to the cardinality restrictions on object properties linking the class that represents the element to the class that represents the complex type element that contains this element.

```
<xs:element name="paper">
  <xs:complexType>
…
    <xs:element name="autorName" type="xs:string" maxOccurs="3"/>
…
  </xs:complexType>
</xs:element>
```

This example is converted as follows:

```
<!—creation of different elements and properties -->
 <owl:Restriction>
      <owl:onProperty rdf:resource="#hasautorName" />
      <owl:maxCardinality
            rdf:datatype="&xsd;nonNegativeInteger">
                  3
      </owl:maxCardinality>
</owl:Restriction>
```

Groups of indicators: Two group indicators are distinguished: the group of elements and attribute group.

The xs:group element of XS is used by a complex type to insert all elements of the group among the elements of the type. This element is mapped to a class that has the same name as the group name and inherits from the GroupElement class; this class is related to the class that represents the complex type by the object property named "hasNestedGroupGroupname". It is also related to classes that represent the member elements of the group by properties of objects each of which bears the name "hasElementName".

The xs:attributeGroup element is used by a complex type element to insert a group of attributes among its attributes. This element is mapped to a data type property with the name "attributeGroupName", which inherits from the data type property AttributeGroup, this property has as domain the classes that represent the complex types that refer to this group of attributes. Then all attributes of the group are mapped to data type properties that inherit from the property representing the group.

```
<xs:attributeGroup name="attributeGroupName">
   <xs:attribute name="attribute1" type="xs:integer"/>
   <xs:attribute name="attribute2" type="xs:string"/>
                …
   <xs:attribute name="attributn" type="xs:string"/>
</xs:attributeGroup>
<xs:complexType name="cmplxTName1">
   <xs:attributeGroup ref="attributeGroupName"/>
</xs:complexType>
<xs:complexType name="cmplxTName2">
   <xs:attributeGroup ref="attributeGroupName"/>
</xs:complexType>
```

This script is converted in ontology language as follows:

```
<owl:DatatypeProperty rdf:ID="attributeGroupName">
```

```
    <rdfs:domain rdf:resource="#cmplxName1"/>
    <rdfs:domain rdf:resource="#cmplxName2"/>
    <rdfs:range  rdf:resource="&xsd;string"/>
    <rdfs:subPropertyOf rdf:resource="#AttributeGroup"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="attribute1">
    <rdfs:range  rdf:resource="&xsd;integer"/>
    <rdfs:subPropertyOf
              rdf:resource="#attributeGroupName"/>
</owl:DatatypeProperty>
        …
<owl:DatatypeProperty rdf:ID="attributen">
      <rdfs:range  rdf:resource="&xsd;string"/>
      <rdfs:subPropertyOf
              rdf:resource="#attributeGroupName"/>
</owl:DatatypeProperty>
```

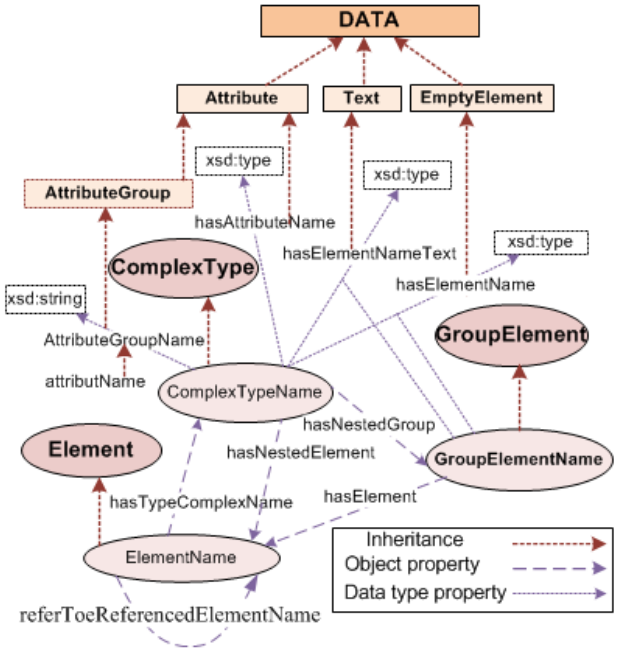


Fig. 1 Insertion on classes and properties in the template ontology

## IV. Flattening

This step is applied only if we want to switch to the data level, ie the conversion of the content of XML documents to individuals of ontology. Indeed, XML documents instances of XSD document contain only instances of elements defined using the tag "xs: element" in XSD. The objective of this step is to flatten the previous model to obtain an ontology that only keeps the classes that represent the elements "xs: element" of XSD. This is done through SPARQL queries. To achieve this, we apply the following rules:

**Rule 1:** If a subclass (cmplxT) of ComplexType class is related to another subclass (grpElt) of GroupElement class by an object property then any object property or data type property having the domain grpElt changes its domain by the cmplxT class, thus any e subclasse of the Element class related to the GrpElt class will be directly linked to the CmplxT class. Thereafter, we proceed to the removal of all object properties that link the subclasses of GroupElement class to subclasses of Element class, as well as those that link the subclasses of ComplexType class to those of GroupElement class.

The following SPARQL request is used to return all subclasses of the ELEMENT class belonging to the grpElt group which is linked to the cmplxT class

```
PREFIX authorsdb:
              <C:/Users/Jamal/Desktop/ontologie.RDF#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
SELECT    ?e
FROM  <C:/Users/Jamal/Desktop/ontologie.RDF>
WHERE{
     ?cmplxT    rdf:type      owl:Class.
     ?e         rdf:type      owl:Class.
     ?grpElt    rdf:type      owl:Class.
     ?cmplxT    rdfs:subClassOf  authorsdb:ComplexType.
     ?grpElt    rdfs:subClassOf  authorsdb:GroupElement.
     ?e         rdfs:subClassOf  authorsdb:ELEMENT.
     ?o         rdf:type         owl:ObjectProperty.
     ?ob        rdf:type         owl:ObjectProperty.
     ?o         rdfs:domain      ?cmplxT.
     ?o         rdfs:range       ?grpElt.
     ?ob        rdfs:domain      ?grpElt.
     ?ob        rdfs:range       ?e
  }
```

**Rule 2:** Let A be a data type property that represents a group of attributes, it inherits the AttributeGroup data type property.

A can have multiple classes as domain. These are the classes that represent the elements of complex type that includes the attribute group. In this case, all sub properties of A will have as domain the domain of A, and will inherit directly from the Attribute property instead of A. this is equivalent in XS to integrate the attributes of attributes group in all complexType elements which incorporates the group. This procedure is applied to all direct sub properties of AttributeGroup.

```
PREFIX authorsdb:
        <C:/Users/Jamal/Desktop/ontologie.RDF#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
SELECT    ?dtp
FROM  <C:/Users/Jamal/Desktop/ontologie.RDF>
WHERE {
      ?dtp          rdf:type      owl:DatatypeProperty.
      ?superdtp     rdf:type      owl:DatatypeProperty.
      ?cmplxT       rdf:type      owl:Class.
      ?cmplxT    rdfs:subClassOf  authorsdb:ComplexType.
      ?superdtp      rdfs:domain          ?cmplxT.
```

```
    ?dtp            rdfs:subPropertyOf    ?superdtp
}
```

**Rule 3:** If a class E subclass of the ELEEMENT class is related to CmplxT subclass of the ComplexType class by an object property that has the domain E and the range CmplxT, then any object property or data type property having as the domain CmplxT class, will change its domain by class E.

**Rule 4:** If an E class, subclass of class EElement linked by an object property (hasType_ CmplxT) to CmplxT subclass of the ComplexType class, then all the properties of objects (hasNestedElement) will change its domain which is cmplxt by the E class.

This query will return for each e class representing an element, all classes that represent nested elements in this element.

```
PREFIX authorsdb:
<C:/Users/Jamal/Desktop/ontologie.RDF#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
SELECT    ?e    ?subElt
FROM  <C:/Users/Jamal/Desktop/ontologie.RDF>
WHERE{
    ?cmplxT    rdf:type      owl:Class.
    ?e         rdf:type      owl:Class.
    ?subElt    rdf:type      owl:Class.
    ?cmplxT    rdfs:subClassOf  authorsdb:ComplexType.
    ?e         rdfs:subClassOf  authorsdb:ELEMENT.
    ?subElt    rdfs:subClassOf  authorsdb:ELEMENT.
    ?hastype   rdf:type         owl:ObjectProperty.
    ?hastype   rdfs:domain      ?e.
    ?hastype   rdfs:range       ?cmplxT.
    ?hasNested   rdf:type       owl:ObjectProperty.
    ?hasNested   rdfs:domain    ?cmplxT.
    ?hasNested   rdfs:range     ?subElt;
}
```

**Rule 5:** The last step is to remove all unnecessary items from the ontology, thus all object properties whose domain or range is a subclass of the GroupElement class or of complexType class are eliminated. Then we delete the GroupElement class after deleting all its subclasses, we also remove the AttributeGroup property and all its sub properties, and finally, we delete ComplexType class and all its descendant classes.

After these modification, and unlike other methods that propose to generate classes for the various XS elements, we got a schema where all classes represent the elements "xs:element" of document XS. The importance of this simplification is more visible when passing in conversion of XML documents at the data level. This conversion is then made by making assertions of various classes and properties of the model obtained using the content of XML document.

The form of the resulting ontology is provided by the following figure:

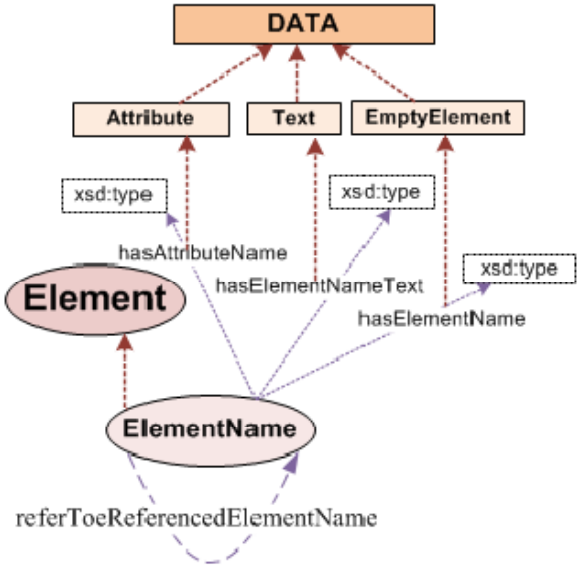


Fig. 2 The resulting ontology after flattening

## V. Implementation and Case Study

In order to implement this approach, we developed a prototype using the java language and its various libraries. Thus, we used the XML parser JDOM [11] to parse the elements of the XML Schema documents, given that these documents are XML documents. The output of JDOM is a set of objects representing the elements and attributes with their values, this output is used as input of a module based on the Jena API [12] module. This module is responsible for the creation of ontology template, and then fills it by classes and properties created using the module inputs. The output of this module is the resulting ontology of conversion.

We pass several XML Schema document at input to our prototype, and we have got ontologies consistent with desired results. Hereafter, we present an example of XML schema document with its resulting ontology of conversion.

```
<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="book">
<xs:complexType>
  <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" maxOccurs="3">
      <xs:complexType>
   <xs:sequence>
   <xs:element name="name" type="xs:string"/>
    <xs:element name="born" type="xs:date"/>
   <xs:element name="dead" type="xs:date"/>
   <xs:element name="adresse" type="adresseType"/>
   </xs:sequence>
      </xs:complexType>
      </xs:element>
      <xs:group ref="editorGroup"/>
  </xs:sequence>
  <xs:attribute name="isbn" type="xs:string" />
  <xs:attribute name="nbPage" type="xs:integer" />
  <xs:attribute name="lang" type="xs:string" />
</xs:complexType>
</xs:element>
<xs:group name="editorGroup">
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="webSite" type="xs:string"/>
   <xs:element name="tel" type="xs:string"/>
   <xs:element name="adresse" type="adresseType"/>
  </xs:sequence>
</xs:group>
 <xs:complexType name="adresseType">
  <xs:sequence>
      <xs:element name="number"   type="xs:integer"/>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city"   type="xs:string"/>
      <xs:element name="zip"    type="xs:decimal"/>
      <xs:element name="country"  type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Fig. 3 Example of xml schema

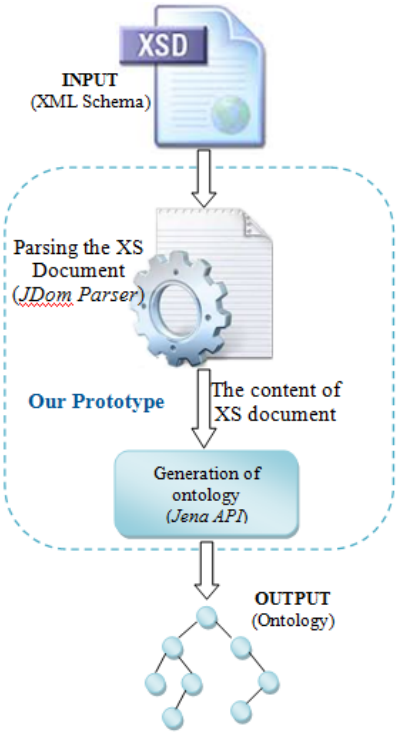


Fig. 4 Descriptive figure of our prototype

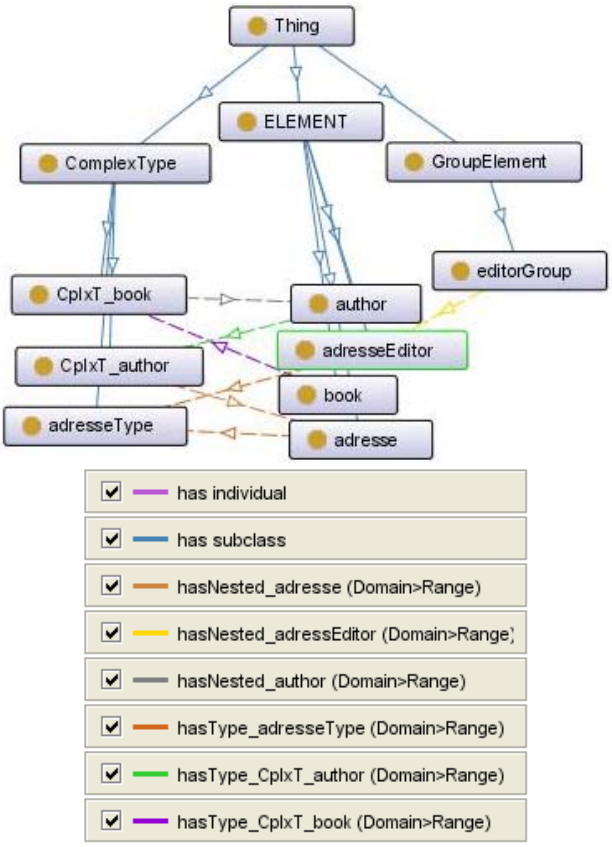


Fig. 5 Resulting ontology viewed under Protégé

## VI. Conclusion

In this paper, we proposed a method to generate an ontology from a XS document. By applying this method, all the generated classes represent only the "xs_element" elements of xsd document, unlike methods that generate classes for other elements as "xs:complexType", "xs: group" and other. The advantage of this, is manifested when passing to the data level, in this level, a valid XML document is composed only of a set of instances of elements of its xsd schema, which will also facilitate a conversion at this level by assertions of the components of the generated model. In addition, the hierarchy that we propose for the data type properties allows us to classify the data converted according to their structures in ML documents.

## References

[1] Richard Cyganiak, David Wood, Markus Lanthaler, "RDF 1.1 Concepts and Abstract Syntax" W3C Recommendation 25 February 2014, website [Online]. Available: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein, " OWL Web Ontology Language", website [Online]. Available: http://www.w3.org/TR/owl-ref/
[3] G. Antoniou, F. van Harmelen, "Web Ontology Language: OWL". pages 76-92 Springer-verlag .2003
[4] Bedini, Ivan, Georges Gardarin, and Benjamin Nguyen. "Deriving ontologies from XML schema." arXiv preprint arXiv:1001.4901 (2010).
[5] Bedini, Ivan, Benjamin Nguyen, and Georges Gardarin. "Janus: Automatic Ontology Builder from XSD files." arXiv preprint arXiv:1001.4892 (2010).
[6] Bedini, Ivan, et al. "Transforming xml schema to owl using patterns." Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on. IEEE, 2011.
[7] Bohring, Hannes, and Sören Auer. "Mapping XML to OWL Ontologies." Leipziger Informatik-Tage 72 (2005): 147-156.
[8] Extended Style Sheet Transformations http://www.w3.org/TR/xslt
[9] Yahia, Nora, Sahar A. Mokhtar, and AbdelWahab Ahmed. "Automatic Generation of OWL Ontology from XML Data Source." arXiv preprint arXiv:1206.0570 (2012).
[10] Xu, Jiuyun, and Weichong Li. "Using relational database to build OWL ontology from XML data sources." Computational Intelligence and Security Workshops, 2007. CISW 2007. International Conference on. IEEE, 2007.
[11] The JDOM website. [Online]. Available: http://www.jdom.org/
[12] The Jena website. [Online]. Available: http://jena.apache.org/index.html

**Jamal BAKKAS** was born in 1979, in Marrakech, Morocco. He got his special higher studies degree in engineering of information systems after completing his license in computer sciences from University Cadi Ayyad of Marrakech; He is phd student in the Department of Mathematics and computer sciences, Faculty of Sciences & Technology of Settat, University Hassan I, Settat, Morocco. His area of interest includes web ontologies and semantic web.

**Mohamed BAHAJ** was born in 1964, in ouezzane, Morocco. He got his PhD in Applied Mathematics, from University of Pau, France, in 1993. He is now working as a Professor at the Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology Settat, Morocco. His research interests include pattern recognition, Semantic web & Ontology in MAS, Controls of mobiles agents.

**Soklabi Abdellatif** was born in 1985, in El JADIDA, Morocco. He had a license degree in computer engineering in 2009 and a master's degree in computer systems and networks in 2011. Now he is a PhD researcher in mobiles agents and web services in Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology of Settat, Morocco. His research interests include, Load Balancing & Controls of mobiles agents, Interoperability between different MAS.