Journal of Theoretical and Applied Information Technology
30th April 2017. Vol.95. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

XSD2OWL2: AUTOMATIC MAPPING FROM XML SCHEMA INTO OWL2 ONTOLOGY

[1]OUSSAMA EL HAJJAMY, [2]LARBI ALAOUI , [3]MOHAMED BAHAJ
[1,3]University Hassan I, FSTS Settat, Morocco
[2]International University of Rabat 11100 Sala Al Jadida, Morocco
E-mail: [1]elhajjamyoussama@gmail.com, [2]larbi.alaoui@hotmail.de, [3]mohamedbahaj@gmail.com

ABSTRACT

XML (extensible Markup Language) nowadays is a common format widely used by domain experts to exchange data and information on the internet. It allows systems to agree on a common syntax and understand each data source that they access. On the other hand, OWL (Ontology web language) contains a group of concepts and properties to make the information in the Web processable and semantically understandable by machines. Compared to XML, OWL is a vision for the future web (web semantic), it gives explicit meaning to information and provides additional vocabulary to formally describe the meaning of the terminology used to annotate Web resources. In this paper we provide and develop a new solution that converts the XML schema into OWL2 ontology. This solution takes an existing XML schema (XSD) as input, loads the XSD document and parses it using DOM parser. Then it extracts its elements with as much constraints as possible and applies our mapping algorithm to create the resulting OWL2 document. Moreover, whole of the transformation processes are done automatically without any outside intervention. Our aim in this work is to take a further step in the existing research works by considering other important XSD aspects and minimizing our algorithm execution cost. In order to apply our approach in real environments, we have developed a tool XSD2OWL2 that implements our mapping algorithm for our conversion model and demonstrates the effectiveness and power of our strategy.

Keywords:- XML schema, XSD, OWL2, ontology, DOM parser

1. INTRODUCTION

Today, large volumes of data and information are becoming available over the web. This information's can be structured, such as relational databases, and semi-structured, such as XML documents. However, the access to all information available in applications that deal with heterogeneous data remains limited and the formalism of XML do not provide a format that is at the same time both human readable and machine interpretable. In contrast, OWL leverages current languages and additionally provides the means by which the semantics can be assigned to data through a set of terms related to a body of knowledge. This knowledge provides a technology to solve the semantic heterogeneity problem. Therefore, the problem of migrating XML to OWL is becoming an active research domain.

Several reasons motivated this choice, the first of them being that XML can translate data grammars, whereas ontologies try to represent the semantics of the objects. Another reason is that applications based on ontologies are more and more numerous since the emergence of the semantic web, however, XML is the standard format for data exchange between enterprise applications on the Internet and companies do always wish to keep the existing systems having in mind the time and money already spent on them. Thus, instead of rebuilding the applications and in order to make the existing systems available for the semantic web, it is more suitable to find good solutions for the migration from xml document to ontology web.

Towards this goal, we hereby propose a method, called XSD2OWL2, for ontology creation from xml schema. This method is based on some mapping rules that automatically generate OWL ontology from an XML data source.

Currently, there are two options recommended by theW3C for defining an XML schema. One is the Document Type Definition (DTD) and the other is the XML Schema (XSD). We choose XML Schema because:

- it has a powerful set of types and constraints which leads to a better translation;
- it provides us with a more flexible and powerful mechanism through "key" and "keyref" constructs;
- and with XSD we are able to model complex constraints.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Our aim in this work is to take a further step in the existing research works by identifying the weaknesses and limitations of the different existing techniques and proposals, and address other very important aspects that have not been touched yet in the world of conversion from XML to OWL. These aspects are mainly related to Transitivity, circular relations, bidirectional relations and some other constraints.

We perform our work at two levels, one providing a comparison of the existing mapping methods from XML to OWL and the other proposing a novel migration solution XSD2OWL2 that generalizes these methods, optimizes the constraints extraction and refines the mapping rules to be more expressive and less complicated.

The rest of the paper is organized as follows. In the following section we present an overview of the different XML to OWL schema transformation proposals. Needful terminology and several rules to convert XML schema into OWL2 ontology are presented in section 3. To illustrate how to combine the rules together for a concise mapping, sections 4 outline the automatic mapping algorithm based on the list of rules. The implementation based on the conversion approach is presented in section 5. Finally, section 6 includes some conclusions and future work.

2. COMPARISON OF EXESTING MAPPING METHODS

As mentioned in section 1, there are many researches that have been proposed to achieve XML to OWL conversion. However the existing studies do not provide a complete solution to this problem and so far there still be no effective proposals that could be considered as a standard method that preserves the whole original structure and constraints of the XML schema.

In this section, we make a literature review to investigate some of existing approaches that address the problematic of generating OWL ontology from XML document. The investigation of these methods is done with the focus on their shortcomings with regards to the relevant elements and constraints that are not considered in their mapping process.

Jyun-Yao propose in [9] a template that can handle extremely large XML data and provides user friendly templates composed of RDF triple patterns including simplified XPath expressions. However this method has inherent drawbacks because RDF does not have enough expressive power to capture the knowledge of the source xml document, and the generated RDF files are not really semantically richer than the mapped XML Schemas.

Ferdinand et al. [10] propose a mechanism to lift XML structured data to semantic web. This approach is twofold: mapping concepts from XML to RDF and from XML Schema to OWL. In the first mapping process, two categories of XML elements are distinguished: 1) elements that have sub-elements and/or attributes, 2) attributes and elements that carry only a simple data type. In this case, the mapping is performed by the following procedure: for each sub-element and attribute of the element that is currently processed, an RDF property on the RDF resource created before in the previous step is created. The value of this property is determined as follows: elements and attributes with data type component generate RDF literal on the respective property, and elements and attributes with Object component an anonymous RDF resource is created and assigned as the value of the respective property. Then, this component is processed recursively. The second part is based on a set of interpretation and transformation rules. Each complexType is mapped to an owl:Class. Each element and attribute declaration is mapped to an OWL property. Elements of simpleType and all attributes are mapped to an owl:DatatypeProperty and elements of complexType are mapped to an owl:ObjectProperty. This part also deals with model groups, specialization and cardinality constraints. However, these mapping concepts are independent and they neglect many elements and constraints of the source XML document.

F. Breitling [1] proposes a standard mapping method from XML to RDF via XSLT. It provides direct conversion and contains XPath information for retaining the hierarchical information of the XML data source. However, this approach has two drawbacks. First users must have knowledge of the complicated XSLT transformation language. Second, the method requires the human interactions to replace the subject URIs of the generated RDF documents if the XPath-style expressions are not deemed suitable as unique URIs.

The approach proposed by An Yuan et al [22] consists in constructing mapping rules between XML schema and OWL ontology. Unlike other methods, this method requires the existence of the target ontology, and it is based on a heuristic algorithm to find correspondences between the tree structure of the XML Schema and the connections of the different components of the ontology.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Jiuyun Xu et al [8] propose an indirect method to realize this transformation through the entity-relation model to alleviate the difficulties of transformation. Instead of a direct mapping, the authors propose an XTR mapping (XML to relational) to pass from the XML to the relational model, followed by another RTO mapping (relational to ontology) to extract the ontology from this model. However, this approach is not enough for expressing the full semantics of the database domain, because the passage XTR and RTO leads to a loss of information from the xml source document.

Bedini et al [3], propose a tool called "Janus", this last provides automatic derivation of ontologies from XS files by applying a set of derivation rules. Then, the same group proposed a method based on patterns [4] that deals with 40 patterns and convert each pattern to equivalent OWL ontology. However, this approach have limited capacity in integration and merging and further research is still needed to improve the capacity to detect well-formed sources and semantics.

All aforementioned ontology based transformation present limitations in treating various important XSD elements related to the art of elements, relations or constraints. Our approach and implementation present some innovative advantages, it give other very important aspects that have not been touched yet in the world of conversion from XML to OWL.

3. XSD TO OWL2 MAPPING MODEL

Our approach aims at defining a correspondence between the xml schema and OWL2 ontology. It maintains the structure as well as the meaning of XML schema. Moreover, our mapping method provides more semantics for XML instances via adding more definitions for elements and their relationships in OWL ontology by using OWL2 functional-style syntax.

Our strategy consists of three separate phases as shown in figure 1.

In the first step the system loads the XML document and parses it using DOM technology (Document Object Model). The output of DOM is a set of objects representing the different elements of the source xml schema (such as complex types, simple types, Specialization, elements, attributes, constraints and several relationships), this output is extracted and used as the input of our mapping algorithms in the second step. Finally the system applies our algorithm based on the list of rules to create the equivalent ontology in owl2. Figure 1 below shows the architecture of XSD2OWL2 implementation.

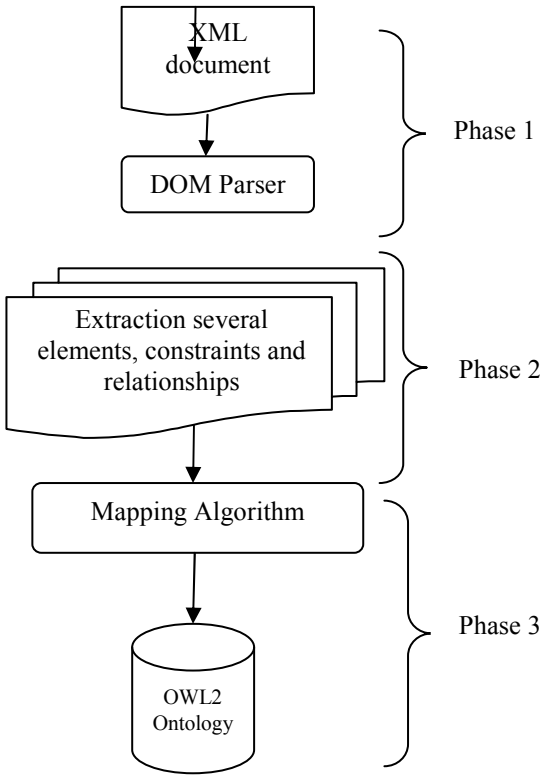

Figure 1 : *XSD2OWL2 framework architecture*

In the following, we propose to give clear and concise conversion rules by taking into account all such constructs of the source XML document. The rules allow us to derive an algorithm that is as simple as possible and which does not use any intermediate language. To this end we consider relevant categorizations related to the various components in XML schema:

A. Mapping Complex type:

Complex type is normally used to define components with child elements, attributes or text. We can distinguish two kinds of complex types:

- Global named complex types: when the complex type is globally declared for an element.
- Local anonymous complex types: without name when the complex type is used locally for an element.

Rule1. Both cases are mapped to OWL classes. The generated class will have the name of its surrounding element.

	Global named complex types: *<xsd:complexType name="ctName">* *----------* *</xsd:complexType>*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

XML	*Local anonymous complex types:* *<xsd:element name="ctName">* *<xsd:complexType>* *----------* *</xsd:complexType>* *</xsd:element>*
OWL 2	*Declaration(Class(:ctName))*

B. **Mapping Simple type**

A simple type means that the content of an element or attribute can only be a value of a predefined data type such as an integer or a string. We can derive a new simple type by restricting an existing simple type. The Restriction on XML simple type is normally a range of conditions to be applied on the XML element to define accepted values for this element.

Rule2. In XML schema, simple types are used in defining concrete data type. For this reason, every simple type is mapped into OWL2 datatype.

XML	*<xsd:simpleType name="stName">*
OWL 2	*Declaration(Datatype(:stName))*

There are 3 ways in which a simple type can be extended; Restriction, List and Union:

•An xsd:restriction: child element derives by restricting the legal values of the base type.

•An xsd:union is a mechanism for combining two or more different data types into one.

•An xsd:list child element derives a type as a white space separated list of base type instances. A List is constructed in a similar way to a Union. The difference being that we can only specify a single type. This mechanism has no equivalence in owl 2. So we treat it as a simple type.

Restriction element XML Schema provides a mechanism of restricting a given simple type, which is known as facets. We can establish a set of translation rules for all constraining facets provided by XML Schema:

Rule3. Restriction using regular pattern: XML schema defines the pattern constraint (xsd:pattern) to limit the content of an XML element to define a series of numbers or letters.

XML	*<xsd:simpleType name="stName">* *<xsd:restriction base="xsd:integer">* *<xsd:pattern value="[0-9]{5}"/>* *</xsd:restriction>* *</xsd:simpleType>*
OWL 2	*Declaration(Datatype(:stName))* *DatatypeDefinition(* *:stName* *DatatypeRestriction(xsd:integer* *xsd:pattern "[0-9]{5}"))*

Rule4. Restriction on value: Specifies the bounds for numeric values (xsd:minIclusive, xsd:minExclusive, xsd:maxInclusive, xsd:maxExclusive).

XML	*<xsd:simpleType name="stName">* *<xsd:restriction base="xsd:integer">* *<xsd:minExclusive value="0" />* *<xsd:maxExclusive value="20" />* *</xsd:restriction>* *</xsd:simpleType>*
OWL 2	*Declaration(Datatype(:stName))* *DatatypeDefinition(* *:stName* *DatatypeRestriction(xsd:integer* *xsd:minExclusive "0"^^xsd:integer* *xsd:maxExclusive "20"^^xsd:integer))*

Rule5. Restriction on length: XML use the length, maxLength, and minLength constraints to limit the length of a value in an element.

XML	*<xsd:simpleType name="stName">* *<xsd:restriction base="xsd:string">* *<xsd:minLength value="10" />* *<xsd:maxLength value="20" />* *</xsd:restriction>* *</xsd:simpleType>*
OWL 2	*Declaration(Datatype(:stName))* *DatatypeDefinition(* *:stName* *DatatypeRestriction(xsd:string* *xsd:minLength* *"10"^^xsd:integer* *xsd:maxLength* *"20"^^xsd:integer))*

Rule6. Restriction on Set of Values: XML schema defines the enumeration constraint (xsd:enumeration) to limit the content of an XML element to a set of acceptable values.

XML	*<xsd:simpleType name="stName">* *<xsd:restriction base="xsd:string">* *<xsd:enumeration value="value1"/>* *<xsd:enumeration value="value2"/>* *<xsd:enumeration value="value3"/>* *</xsd:restriction>* *</xsd:simpleType>*
OWL 2	*Declaration(Datatype(:stName))* *DatatypeDefinition(* *:stName* *DataOneOf("value1"^^xsd:string* *"value2"^^xsd:string* *"value3"^^xsd:string))*

Rule7. Union element: This mechanism is transformed in OWL2 to a new data type, using DataUnionOf axiom.

XML	*<xsd:simpleType name="stName1">* *<xsd:union memberTypes="xsd:string* *stName2"/>* *</xsd:simpleType>*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OWL 2	*Declaration(Datatype(:stName1))* *DatatypeDefinition(:stName1* *DataUnionOf(xsd:string stName2))*

C. **Mapping Specialisation:**

XML schema supports two mechanisms of Specialization: extension and restriction. Both of these inheritance mechanisms can be included in the following xsd elements: <xsd:simpleContent> and <xsd:complexContent>. XSD provides two forms of sub-classing type components:

Rule8. The first form extends the definition of another base simple content (element or attribute) to another specified data type. This construct can be interpreted as a way to express new data range that contains all tuples of literals that are contained in the old and new simple types.

XML	*<xsd:complexType name="ctName" > <xsd:simpleContent> <xsd:extension base="stName"> <xsd:attribute name="attr" type="xsd:string"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType>*
OWL 2	*Declaration(DataProperty(:attr))* *DataPropertyDomain(:attr :ctName)* *DataPropertyRange(:attr* *DataUnionOf(:stName xsd:string))*

Rule9. The second form defines a complex type as an extension or a restriction of another base complex type by using the XSD complex content. In this case the class corresponding to this type is set as subclass of the class corresponding to the base type.

XML	*<xsd:complexType name="ctName1" > <xsd:complexContent> <xsd:extension base="ctName2"> ---------- </xsd:extension> </xsd:complexContent > </xsd:complexType>*
OWL 2	*SubClassOf(:ctName1 :ctName2)*

D. **Mapping Element**

The <xsd:element> element is the tag name that will be used within the XML instance document. It allows the description of simple and complex entities. Elements can be declared via several methods:

Rule10. Element declared with primitive data type: is mapped directly to datatype properties by respectively associating with its domain and range the URI of the class corresponding to parent element and the XSD type corresponding to the type of the element.

XML	*<xsd:complexType name="ctName" > <xsd:sequence> <xsd:element name="eName" type="xsd:string"/> </xsd:sequence> </xsd:complexType>*
OWL 2	*Declaration(DataProperty(:eName))* *DataPropertyDomain(:eName :ctName)* *DataPropertyRange(:eName xsd:string))*

Rule11. Element declared with simple type: is mapped directly to datatype properties by respectively associating with its domain and range the URI of the class corresponding to parent element and the inline simple type.

XML	*<xsd:complexType name="ctName" > <xsd:sequence> <xsd:element name="eName" type="stName"/> </xsd:sequence> </xsd:complexType>*
OWL 2	*Declaration(DataProperty(:eName))* *DataPropertyDomain(:eName :ctName)* *DataPropertyRange(:eName :stName))*

Rule12. Element refer to complex type: when a complex type c1 contains an element e of type c2, then an object property is created such that its domain is the concept corresponding to c1 and its range is the concept corresponding to c2.

XML	*<xsd:complexType name="ctName1" > <xsd:sequence> <xsd:element name="eName" type="ctName2"/> </xsd:sequence> </xsd:complexType>*
OWL 2	*Declaration(ObjectProperty(:haseName))* *ObjectPropertyDomain(:haseName :ctName1)* *ObjectPropertyRange(:haseName :ctName2)*

Rule13. Global element declared with complex type (global declaration): is mapped directly to OWL class, and the class corresponding to the element is set as subclass of the class corresponding to its type.

XML	*<xsd:element name="eName" type="ctName"/>*
OWL 2	*Declaration(Class(:eName))* *SubClassOf(:eName :ctName)*

Elements that embed other elements or attributes must have a complex type. In this case, we use Rule1 to convert them to owl classes.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

E. Mapping Attribute

An attribute is used to declare simple values for a given complex element. Attributes themselves are always declared with a simple type or primitive data type.

Rule14. Attribute with primitive data type is treated as simple element with primitive data type (Rule10) and will be mapped to datatype properties by respectively associating with its domain and range the URI of the class corresponding to parent element and the XSD type corresponding to the type of the attribute.

Rule15. Attribute with simple type is treated as element declared with simple type (Rule11) and will be mapped to datatype properties by respectively associating with its domain and range the URI of the class corresponding to parent element and the inline simple type.

F. Mapping Grouping concepts

The XML Schema Recommendation allows to specify groups of elements and groups of attributes using <xsd:group> and <xsd:attributeGroup>. When a group is referred to, it is as if its contents have been copied into the location it is referenced from.

Rule16. The XSD element group and attribute group are used by a complex Type to assemble several elements together. These elements are mapped to a simple class. This class is linked to the class that represents the complex type by an object property axiom named "hasElementGroupeName". The class representing the group is also the domain of the data type properties that represents the simple elements/attributes of the group.

XML	*<xsd:group name="grName">* *<xsd:sequence>* *----------* *</xsd:sequence>* *</xsd:group>* *<xsd:complexType name="ctName">* *<xsd:sequence>* *<xsd:group ref="grName" />* *</xsd:sequence>* *</xsd:complexType>*
OWL 2	*Declaration(Class(:grName))* *Declaration(Class(:ctName))* *Declaration(ObjectProperty(* *:hasgrName))* *ObjectPropertyDomain(:hasgrName* *:ctName)* *ObjectPropertyRange(:hasgrName* *:grName)*

G. Mapping Transitive Chain

Rule17. Let ctName1, ctName2 and ctName3 be three different complex types or Global elements declared with complex type. If ctName2 is declared as a child of ctName1 and ctName3 as a child of ctName2, then there is a transitivity chain between ctName1 and ctName3. We use TransitiveObjectProperty axiom to express it.

XML	*<xsd:element name="ctName1">* *<xsd:complexType>* *<sequence>* *<xsd:element name="ctName2">* *<xsd:complexType>* *<sequence>* *<xsd:element name="ctName3">* *<xsd:complexType>* *----------* *</xsd:complexType>* *<sequence>* *</xsd:complexType>* *</sequence>* *</xsd:complexType>* *</xsd:element>*
OWL 2	*Declaration(ObjectProperty(* *:ctName1_has_ctName3))* *ObjectPropertyDomain(* *:ctName1_has_ctName3 :ctName1)* *ObjectPropertyRange(* *:ctName1_has_ctName3 :ctName3)* *TransitiveObjectProperty(* *:ctName1_has_ctName3)*

H. Mapping Integrity constraints

XML Schema supports two mechanisms to represent identity and reference, key/keyref. It is similar to the primary key and foreign key in the database. The reason for this is that keys and foreign keys establish meaningful connections between different elements. In general, the semantic assertion of a key is that the data entities in an XML document are unique and not null, and the purpose of foreign key is to define the association of two elements in a XML document.

Rule18. An xsd:key uniquely identifies the element in xml document. This implies that the values of the data type property that represent this element must be unique. Therefore, these properties must be declared with OWL2 HasKey property.

XML	*<xsd:element name="ctName">* *<xsd:complexType>* *<sequence>* *<xsd:attribute name="idattr"* *type="xsd:attrType">* *</sequence>* *</xsd:complexType>* *</xsd:element>*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

	<xsd:key name="ctNameKey"> <xsd:selector xpath=".//ctName"/> <xsd:field xpath="./@idattr"/> </xsd:key>
OWL 2	Declaration(Data Property(:idattr)) DataPropertyDomain(:idattr :ctName) DataPropertyRange(:idattr xsd:attrType) HasKey(:ctName :idattr)

Rule19. xsd:KeyRef : For complex types (or Global elements declared with complex type) ctName1 and ctName2, if an attribute att2 in ctName2 references another attribute att1 in ctName1, then an object property is generated, and with its domain and range we respectively associate the URI of the class corresponding to ctName2 and the URI of the class that represents ctName1. To ensure atomicity of the attribute we declare the object property as a "FunctionalObjectProperty".

XML	<xsd:key name=" ctNameKey"> <xsd:selector xpath=".//ctName1"/> <xsd:field xpath="./@att1"/> </xsd:key> <xsd:keyRef name="ctNameRef" refer="ctNameKey"> <xsd:selector xpath=".//ctName2"/> <xsd:field xpath="./@att2"/> </xsd:keyRef>
OWL 2	Declaration(ObjectProperty(:ctNameRef)) ObjectPropertyDomain(:ctNameRef: :ctName2) ObjectPropertyRange(:ctNameRef: :ctName1) FunctionalObjectProperty(:ctNameRef)

I. Mapping Cyclic Relations

For a set of complex types (or Global elements declared with complex type) ctName1 … $ctName_n$ ($n \geq 2$) such that $ctName_i$ is referenced by $ctName_{(i+1)}$ ($2 \leq i \leq n$) and $ctName_n$ is referenced by ctName1, we say that a cyclic relationship exists between these elements. Note that if n= 2 then we get a Bidirectional relation, and if n > 2 then we get a circular relationship between the elements.

Rule20. Bidirectional relations are represented by two Key/Keyref references in the XML Schema. In this case we generate two pairs of inverse object properties.

XML	<xsd:element name="ctName1"> <xsd:complexType> <xsd:sequence> <xsd:element ref=" ctName2" minOccurs="0"/> </xsd:sequence> </xsd:complexType> </xsd:element> <xsd:element name=" ctName2"> <xsd:complexType> <xsd:sequence> <xsd:element ref=" ctName1" minOccurs="0"/> </xsd:sequence> </xsd:complexType>
OWL 2	Declaration(ObjectProperty(:hasctName1)) ObjectPropertyDomain(:hasctName1 :ctName2) ObjectPropertyRange(:hasctName1 :ctName1) FunctionalObjectProperty(:hasctName1) Declaration(ObjectProperty(:hasctName2)) ObjectPropertyDomain(:hasctName2 :ctName1) ObjectPropertyRange(:hasctName2 :ctName2) FunctionalObjectProperty(:hasctName2) InverseObjectProperty(:hasctName1 :hasctName2)

Rule21. A circular relation is defined as a set of relations ctName1 ... $ctName_n$ ($n > 2$), where $ctName_i$ is referenced by $ctName_{i+1}$ ($2 \leq i \leq n$) and $ctName_n$ is referenced by ctName1. In OWL2 this can be expressed using chain axiom property and self restriction objectHasSelf.

XML	<xsd:element name="ctName1"> <xsd:complexType> <xsd:sequence> <xsd:element ref=" ctName2" minOccurs="0"/> </xsd:sequence> </xsd:complexType> </xsd:element> <xsd:element name=" ctName2"> <xsd:complexType> <xsd:sequence> <xsd:element ref=" ctName3" minOccurs="0"/> </xsd:sequence> </xsd:complexType> </xsd:element> <xsd:element name=" ctName3"> <xsd:complexType> <xsd:sequence> <xsd:element ref=" ctName1" minOccurs="0" /> </xsd:sequence> </xsd:complexType> </xsd:element>
OWL 2	SubObjectPropertyOf(ObjectPropertyChain(:ctName1 _ctName2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

	:ctName2_ctName3 *:ctName3_ctName1) :Z)* *SubClassOf(ObjectHasSelf(:Z)* *:ctName1_ctName1)*

J. Mapping Constraints

In our transformation rules, other constraints, such as xsd:unique, xsd:use, and max/min occurence are also taken into account to make the mapping complete. We aim to preserve as many constraints as possible.

Rule22. For each element with a UNIQUE constraint we set maxCardinality restriction to 1 in order to prevent the creation of individuals having the same value.

XML	*<xsd:element name="ctName">* *<xsd:complexType>* *<sequence>* *<xsd:attribute name="uqattr"* *type="xsd:attrType">* *</sequence>* *</xsd:complexType>* *</xsd:element>* *<xsd:unique name="ctNameKey">* *<xsd:selector xpath=".//ctName"/>* *<xsd:field xpath="./@uqattr"/>* *</xsd:unique>*
OWL 2	*Declaration(Data Property(:uqattr))* *DataPropertyDomain(:uqattr* *:ctName)* *DataPropertyRange(:uqattr* *xsd:attrType)* *DataMaxCardinality(1 :uqattr)*

Rule23. The syntax for specifying occurrences of attributes is different than the syntax for elements. Attributes can be declared with a use attribute to indicate whether the attribute is required, optional, or even prohibited. On the other side, elements use Occurrence indicators to define how often an element can occur. The associated conversion rules are given as follow:

Attributes	
use="required	Set DataMinCardinality to 1
use="optional"	Set DataMinCardinality to 0
Elements converted to DataProperty	
minOccurs	DataMinCardinality
maxOccurs	DataMaxCardinality
Elements converted to ObjectProperty	
minOccurs	ObjectMinCardinality
maxOccurs	ObjectMaxCardinality

4. XML TO OWL2 MAPPING ALGORITHM

In this section, we present our algorithm for the automatic construction of OWL2 Ontology from xml schema. This algorithm takes into consideration all the aforementioned conversion rules. It captures the semantic properties of XSD such as specialization, restriction, extension, transitivity, cyclic relationships … .

We introduce a simple XML schema as a running example. This example will be used throughout the following sections in order to illustrate the different steps of mapping generation algorithm and ontology generation process.

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="targetNamespaceURI"
  xmlns="targetNamespaceURI"
  elementFormDefault="qualified">
<xsd:element name="Author" type="Person"/>
<xsd:complexType name="Person">
<xsd:sequence>
<xsd:group ref="PersonInfo"/>
<xsd:element maxOccurs="1" ref="University"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Address">
<xsd:sequence>
<xsd:element name="name" type="xsd:string" />
<xsd:element name="street" type="xsd:string" />
<xsd:element maxOccurs="1" ref="City />
</xsd:sequence>
<xsd:attribute name="refCountry"
                type="xsd:string" />
</xsd:complexType>
<xsd:element name="Country ">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="City" minOccurs="0"/>
<xsd:element name="nameCountry"
                type="xsd:string" />
</xsd:sequence>
<xsd:attribute name="idCou" type="xsd:integer"
                                use="required"/>
</xsd:complexType>
<xsd:key name=" CountryKey">
<xsd:selector xpath=".//Country"/>
<xsd:field xpath="./@idCou"/>
</xsd:key>
<xsd:keyRef name="CountryRef"
            refer="CountryKey">
<xsd:selector xpath=".//Address"/>
<xsd:field xpath="./@refCountry"/>
</xsd:keyRef>
</xsd:element>
<xsd:element name="City">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="University" minOccurs="0"/>
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
<xsd:element name="nameCity" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="University">
 <xsd:complexType>
 <xsd:sequence>
 <xsd:element ref="Country" minOccurs="0" />
<xsd:element name="nameUniversity"
                    type="xsd:string" />
</xsd:sequence>
<xsd:simpleContent>
<xsd:extension base="xsd:integer">
<xsd:attribute name="idUniv" type="xsd:string"
                                  use="required"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
<xsd:unique name=" UniversityKey">
<xsd:selector xpath=".//University"/>
<xsd:field xpath="./@idUniv"/>
</xsd:key>
<xsd:keyRef name="ctNameRef"
            refer="ctNameKey">
<xsd:selector xpath=".//ctName2"/>
<xsd:field xpath="./@att2"/>
</xsd:keyRef>
</xsd:element>
<xsd:element name="Student" type="Person"/>
<xsd:complexType>
<xsd:sequence>
<xsd:element name="namStudent" type="string" />
<xsd:element name="Department">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="nameDepartment"
                    type="xsd:string" />
<xsd:element name="researchLab" />
<xsd:complexType>
<xsd:sequence>
<xsd:element name="nameresearchLab"
                    type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:group name="PersonInfo">
<xsd:sequence>
<xsd:element name="namePerson"
                    type="xsd:string" />
<xsd:element name="gender" type="GenderType" />
<xsd:element name="AddressPerson"
                    type="Address" />
</xsd:sequence>
</xsd:group>
<xsd:simpleType name=" GenderType ">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Male"/>
<xsd:enumeration value="Female"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

Our algorithm is based on an XML Schema Graph G = (V,E).

V is the vertex set who can be one of the following elements:

Complex type (CT), simple type (ST), elements (ET), attributes (att), group (GR) or Global element declared with complex type (GEwCT).

E is the edge set, who can be one of the following elements:

Contain (Cont), Refer (Ref), has key (hasKey), has keyRef (hasKeyRef), type, extension base CT (ECT) or extension base ST (EST).

For example, Figure 2 shows the XSG of the XML schema of our running example.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

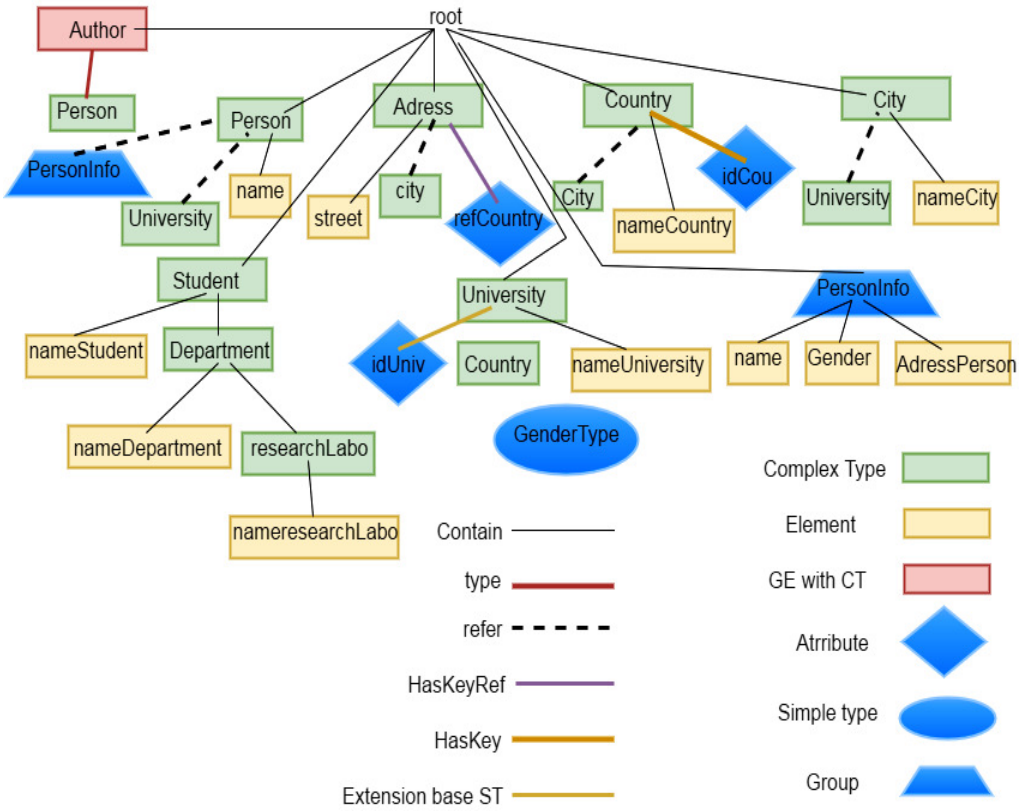

Figure 2 : *XSG of the running example*

MappingXMLSchema()

```
Input: XSD schema S
Begin
   MappingConcepts(S)
   MappingCircularRelation(S)
   MappingTransitiveChain(S)
   MappingBinaryRelations(S)
End
```

MappingConcepts

```
Input: XSG = (V,E)
Begin
  For each v ∈ V Loop
   e = incoming edge of v
   e' = outgoing edge of v
    if v = {CT or GR} then
      if e ≠ {refer and type} then
         Apply rule 1 or 16 : create OWL2 class
      End if
    Else if v = GEwCT then
      if e' = type then
         Apply rule 13 : use class and subClassOf axiom
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
    End if
  Else if v = ST then
    Apply rule 2 : create DataType axiom
    Apply rule 3, 4, 5, 6 or 7 to convert restriction constraints if they exist
  Else if v = CT and e = ECT then
    Apply rule 9 : use SubClassOf OWL2 axiom
  Else if v = {element or attribute} then
    if e' = type and v' = {Primitive type or simple type } then
        Apply rule 10, 11, 14 or 15 : Create OWL2 DataProperty axiom
        Convert constraints if they exist
    End if
  Else if v = CT and e = ECT then
    Apply rule 8 : Create DataProperty axiom with DataUnionOf axiom
  Else if v = {CT, GR or att} and e = {refer, contain or hasKeyRef} then
     Apply rule 12, 16 or 19 : create ObjectProperty axiom
     Convert constraints if they exist
  Else if e = hasKey then
     Apply rule 18 : Create DataProperty with OWL2 HasKey axiom
  End if
 End loop
End
```

To convert the binary relations, transitive chain and circular relation, we used the algorithms presented in our previous work [6]. MappingCircularRelation() procedure uses a recursive function (FindCircularRelation()) to detect if there are any circular relations in XML schema.

5. IMPLEMENTATION AND VALIDATION

In this chapter, we present XSD2OWL2, our tool for XML schema to ontology mapping. This tool takes as input an XML schema document. Then, it extracts elements, relations and all constraints using DOM technology and applies our algorithm based on the list of rules to create the equivalent OWL2 ontology. The DOM parser allows a convenient method for accessing any piece of data in the XML document and also preserves the order of elements. The created ontology is described in OWL2 functional-style syntax. The tool is implemented using Java solutions mainly due to its platform-independent capabilities.

In the following, we provide an example of our platform conversion. Figures 2 and 3 respectively show the screenshot of XSD2OWL2 tool and the OWL2 structure corresponding to the XML schema in our running example.

Journal of Theoretical and Applied Information Technology
30th April 2017. Vol.95. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

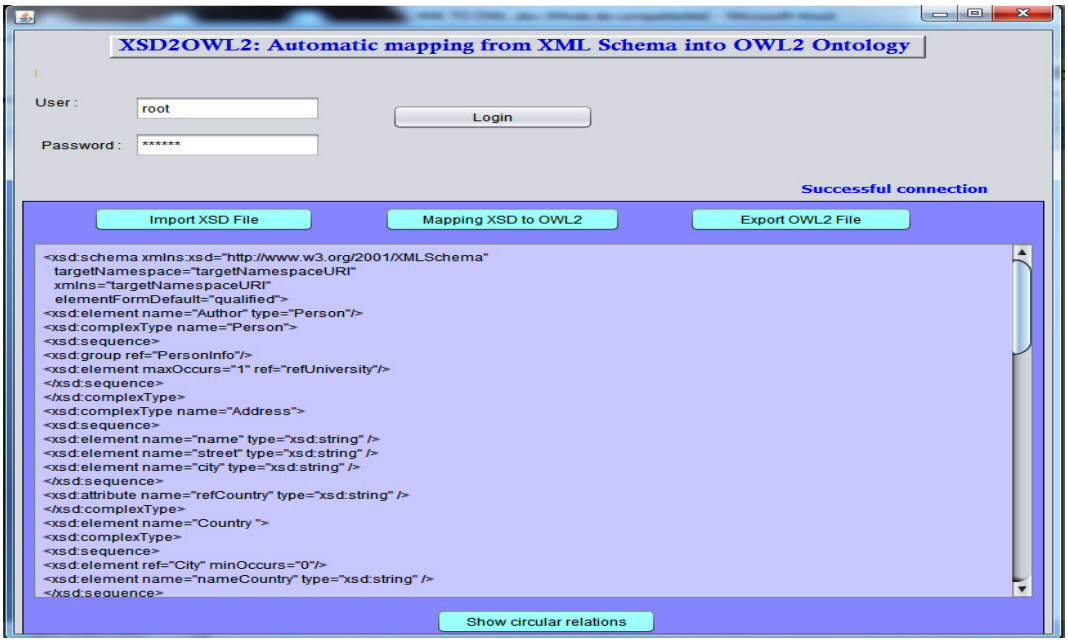

Figure 3 : *Screenshot of XSD2OWL2 tool*

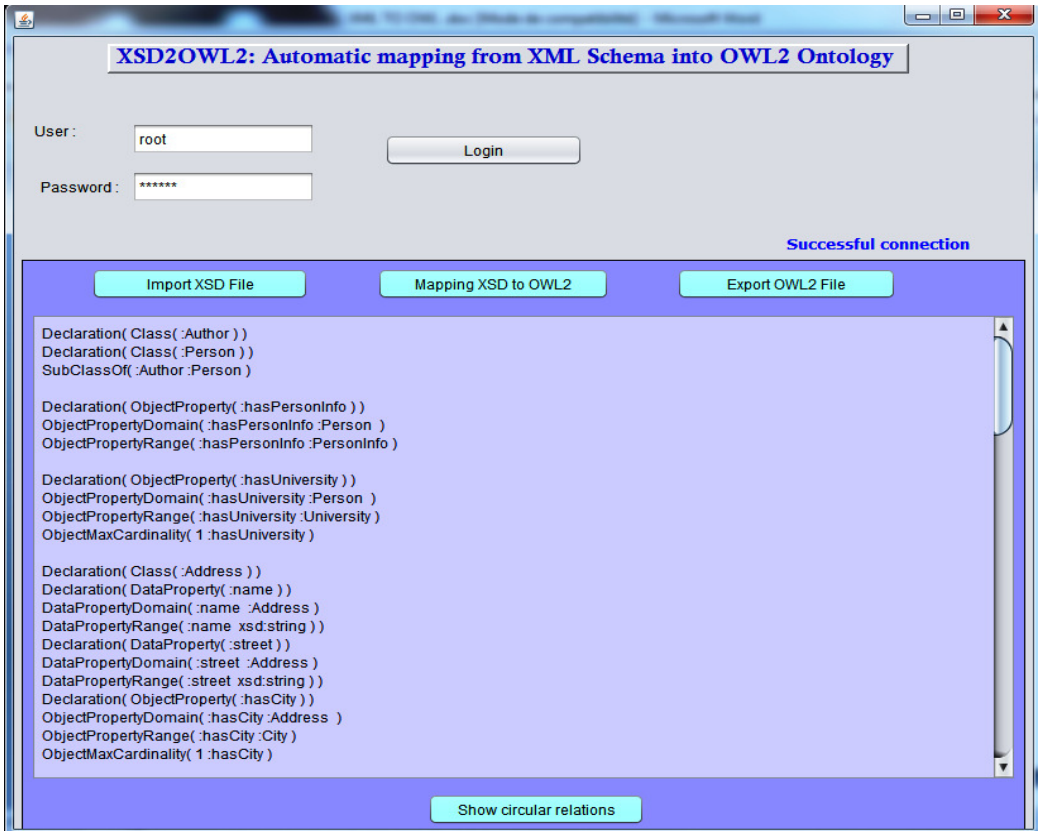

Figure 4 : *Mapping result of XML schema*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The sample screenshot in Figure 6 shows both the extracted circular relationships and their converted OWL2 parts.

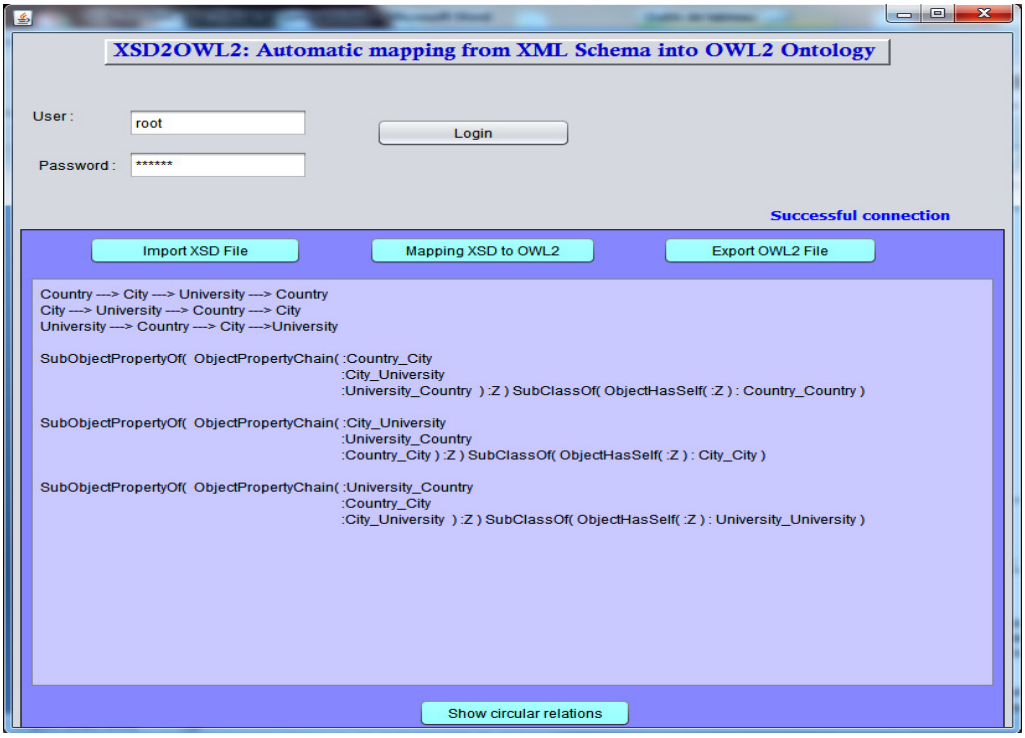

Figure 5 : *Mapping result of circular relations*

The basic ontology graph structure of our running example is as follow:

Declaration(Class(:Author))
Declaration(Class(:Person))
SubClassOf(:Author :Person)

Declaration(ObjectProperty(:hasPersonInfo))
ObjectPropertyDomain(:hasPersonInfo :Person)
ObjectPropertyRange(:hasPersonInfo :PersonInfo)

Declaration(ObjectProperty(:hasUniversity))
ObjectPropertyDomain(:hasUniversity :Person)
ObjectPropertyRange(:hasUniversity :University)
ObjectMaxCardinality(1 :hasUniversity)

Declaration(Class(:Address))
Declaration(DataProperty(:name))
DataPropertyDomain(:name :Address)
DataPropertyRange(:name xsd:string))
Declaration(DataProperty(:street))
DataPropertyDomain(:street :Address)
DataPropertyRange(:street xsd:string))
Declaration(ObjectProperty(:hasCity))
ObjectPropertyDomain(:hasCity :Address)
ObjectPropertyRange(:hasCity :City)
ObjectMaxCardinality(1 :hasCity)

Declaration(Class(:Country))
Declaration(ObjectProperty(:hasCity))
ObjectPropertyDomain(:hasCity :Country)
ObjectPropertyRange(:hasCity :City)
ObjectMinCardinality(1 :hasCity)
Declaration(DataProperty(:nameCountry))
DataPropertyDomain(:nameCountry :Country)
DataPropertyRange(:nameCountry :xsd:string))
Declaration(Data Property(:idCou))
DataPropertyDomain(:idCou :Country)
DataPropertyRange(:idCou xsd:integer)
HasKey(:Country :idCou)
Declaration(ObjectProperty(:CountryRef))
ObjectPropertyDomain(:CountryRef :Address)
ObjectPropertyRange(:CountryRef :Country)
FunctionalObjectProperty(:CountryRef)

Declaration(Class(:City))
Declaration(ObjectProperty(:hasUniversity))
ObjectPropertyDomain(:hasUniversity :City)
ObjectPropertyRange(:hasUniversity :University)
ObjectMinCardinality(1 :hasUniversity)
Declaration(DataProperty(:nameCity))
DataPropertyDomain(:nameCity :City)
DataPropertyRange(:nameCity :xsd:string))

Declaration(Class(:University))
Declaration(ObjectProperty(:hasCountry))
ObjectPropertyDomain(:hasCountry :University)
ObjectPropertyRange(:hasCountry :Country)

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
ObjectMinCardinality(1 :hasCountry )
Declaration( DataProperty( :nameUniversity ) )
DataPropertyDomain( :nameUniversity  :University )
DataPropertyRange( :nameUniversity  :xsd:string ) )
Declaration( DataProperty( :idUniv ) )
DataPropertyDomain( :idUniv  :University )
DataPropertyRange( :idUniv
      DataUnionOf( xsd:integer  xsd:string ) )
DataMaxCardinality( 1  :idUniv )

Declaration( Class( :Student ) )
SubClassOf( :Student  :Person )
Declaration( DataProperty( :nameStudent ) )
DataPropertyDomain( :nameStudent  :Student )
DataPropertyRange( :nameStudent  :xsd:string ) )

Declaration( Class( :Department ) )
Declaration( ObjectProperty( :hasDepartment ) )
ObjectPropertyDomain( :hasDepartment :Student  )
ObjectPropertyRange( :hasDepartment :Department )
Declaration( DataProperty( :nameDepartment ) )
DataPropertyDomain( :nameDepartment
                    :Department )
DataPropertyRange( :nameDepartment  :xsd:string ) )

Declaration( Class( :researchLab ) )
Declaration( ObjectProperty( :hasresearchLab ) )
ObjectPropertyDomain( :hasresearchLab
                    :Department  )
ObjectPropertyRange( :hasreseachLab :researchLab )
Declaration( DataProperty( :nameresearchLab) )
DataPropertyDomain( :nameresearchLab
                    :reseachLab )
DataPropertyRange( :nameresearchLab  :xsd:string) )

Declaration( ObjectProperty(
    :Student_has_researchLab ) )
ObjectPropertyDomain( :Student_has_researchLab
                    :Student  )
ObjectPropertyRange( :Student_has_researchLab
                    :researchLab )

TransitiveObjectProperty(:Student_has_researchLab )

Declaration( Class( :PersonInfo ) )
Declaration( DataProperty( :namePerson ) )
DataPropertyDomain( :namePerson  :PersonInfo )
DataPropertyRange( :namePerson  :xsd:string ) )
Declaration( DataProperty( :gender ) )
DataPropertyDomain( :gender  :PersonInfo )
DataPropertyRange( :gender  :GenderType ) )
Declaration( ObjectProperty( :AddressPerson ) )
ObjectPropertyDomain(:AddresPerson  :PersonInfo  )
ObjectPropertyRange( :AddressPerson  :Address )

Declaration( Datatype( :GenderType ) )
DatatypeDefinition( :GenderType
                    DataOneOf( "Male"^^xsd:string
                    "Female"^^xsd:string ) )

SubObjectPropertyOf(
    ObjectPropertyChain( :Country_City
                         :City_University
                         :University_Country  ) :Z )
SubClassOf( ObjectHasSelf( :Z ) : Country_Country )

SubObjectPropertyOf(
    ObjectPropertyChain( :City_University
                         :University_Country
                        :Country_City ) :Z )
 SubClassOf( ObjectHasSelf( :Z ) : City_City )

SubObjectPropertyOf(
        ObjectPropertyChain( :University_Country
                             :Country_City
                            :City_University  ) :Z )
SubClassOf( ObjectHasSelf( :Z )
          :University_University )
```

We have compared our method to some of the existing approaches. The following table summarizes all mentioned rules and the approaches that have considered them.

TABLE 1 XML TO ONTOLOGY MAPPING COMPARISON METHODS

Constraints	[1]	[4]	[8]	[9]	[10]	[22]	XSD2OWL2
Complex type	✓	✓	✓	×	✓	✓	✓
Simple type	×	✓	✓	✓	✓	✓	✓
Restriction (regular pattern)	×	×	×	×	×	×	✓
Restriction (on value)	×	✓	×	×	×	×	✓
Restriction (on length)	×	×	×	×	×	×	✓
Restriction on set of values	×	✓	×	×	×	×	✓
Union	×	✓	×	×	×	×	✓
Extension from simple type	×	✓	×	×	✓	×	✓
Extension from complex type	×	✓	×	×	✓	×	✓
Element with primitve data type	✓	✓	✓	✓	✓	✓	✓
Element with simple type	✓	✓	✓	✓	✓	✓	✓
Element refer to complex type	✓	✓	✓	✓	✓	✓	✓
Global element declared with CT	×	✓	×	×	✓	✓	✓
Attribute with primitive data type	✓	✓	✓	✓	✓	✓	✓
Attribute with simple type	✓	✓	✓	✓	✓	✓	✓
Group	×	✓	✓	×	✓	×	✓
Transitive chain	×	×	×	×	×	×	✓

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Key	×	×	×	×	×	✓	✓
KeyRef	×	×	×	×	×	✓	✓
Bidirectional relations	×	×	×	×	×	×	✓
Circular relations	×	×	×	×	×	×	✓
Unique	×	×	×	×	×	×	✓
Occurence	×	✓	✓	×	✓	✓	✓

In this table we have identified commonalities and differences between existing mapping techniques and our mapping method. In comparison with other related works, our approach has more advantages since none of the existing transformation tools satisfies all the requirements of transforming xsd schema into OWL ontology. We evaluated each of them on the number of constraints processed during the conversion and the implicit semantics expressed in the source XSD document. Table I shows that these transformation approaches still expose several limitations and do not provide a complete solution to the problematic. Contrary to these existing solutions our developed XSD2OWL2 approach achieves a complete migration of xsd schema into OWL2. Our approach does this conversion in an automatic way, captures richer knowledge of common XSD constraints and uses OWL2 as the target ontology language. Our results can be used immediately without any modification and can be applied to convert arbitrary XSD schema.

6. CONCLUSION AND PERSPECTIVES

The increasing use of ontologies in applications and the wide acceptance of XML as data exchange format have made the problem of migration of XML to the web ontology a fertile area for researchers. In this paper, a systematic approach XSD2OWL2 for an automatic transformation between xml schema and OWL2 is proposed. We especially gave a thorough analysis and comparison of existing mapping methods and identified their weaknesses and limitations. As a result we gave a complete list of elements that are crucial for the conversion and a complete list of associated mapping rules.

Compared to the existing approaches, our new solution optimizes constraints extraction, and supports all of the most common XSD elements such as complex types, simple types, restriction, specialization, integrity constraints, transitive chain, cyclic relations, cardinality constraints and all type of elements and attributes. The XSD2OWL2 tool is much simpler in its design and more complete than others in transformation capacity.

Thanks to OWL 2 the rules are also refined to be more expressive and less complicated using more expressive constructs (e.g., hasKey, DataUnionOf, TransitiveObjectProperty, ObjectHasSelf…). OWL2 also simplifies many programmatic tasks associated with ontologies, including ontology querying and processing. In addition OWL2 can be used to construct full applications that have dependencies on complex ontologies. A limitation of our mapping approach is that it does not treat the mapping at the data-level yet. For our future research related to this topic the focus will be at this "data"-level in order to convert a XML document into the instances part of ontology (ABOX) with all assertions of the different elements from the schema level.

REFERENCES

[1] F Breitling. “A standard transformation from XML to RDF via XSLT”. In: Astronomische Nachrichten 330.7 (2009), pp. 755–760.

[2] G. Klyne and J. Carroll (2004). Resource Description Framework (RDF) Concepts and abstract syntax. W3C Recommendation 10 February 2004, World Wide Web Consortium. http://www.w3.org/TR/rdf-concepts/.

[3] I. Bedini, N. Benjamin , and G. Gardarin, "Janus: Automatic Ontology Builder from XSD files". arXiv preprint arXiv:1001.4892 (2010)

[4] I. Bedini, C. Matheus, P. F. Patel-Schneider, "Transforming XML Schema to OWL Using Patterns". In Semantic Computing (ICSC), 2011 Fifth IEEE International Conference, October 2011.

[5] I. Bedini, "Deriving ontologies automatically from XML Schemas applied to the B2B domain". Doctoral dissertation, University of Versailles, France. January, 2010. Retrieved April 15, 2011.

[6] L.Alaoui, O. Elhajjamy, M. Bahaj, "RDB2OWL2: Schema and Data Conversion from RDB into OWL2
". In International Journal of Engineering Research & Technology (IJERT), Vol. 3 Issue 11, November-2014.

[7] T. Bray, J. Paoli , C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 November 2008".

[8] J. Xu, W. Li, "Using relational database to build OWL ontology from XML data sources". In Computational Intelligence and Security

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Workshops, 2007. (CISW 2007). International Conference on. IEEE. , 15-19 Dec. 2007.

[9] J. Y. Huang, C. Lange, S. Auer, "Streaming Transformation of XML to RDF using XPath-based Mappings". Proceedings of the 11th International Conference on Semantic Systems, SEMANTICS 2015, Vienna, Austria, September 15-17.

[10] M. Ferdinand, C. Zirpins, and D. Trastour, "Lifting XML Schema to OWL". In Web Engineering - 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004, Proceedings (2004).

[11] M. K. Smith, C. Welty, D. L. McGuinness, OWL Web Ontology Language Guide (W3C Recommendation 10 February 2004) [EB/OL]. http://www.w3.org/TR/owl-features/, (last modified on 10 February 2004).

[12] M. Schneider, S. Rudolph2, G. Rudolph, "Modeling in OWL 2 without Restrictions". arXiv: 1212.2902 v3 [cs.AI] 28 Apr 2013.

[13] N. Anicic, N.Ivezic, , and Z.Marjanovic, "Mapping XML Schema to OWL". In Enterprise Interoperability (2007).

[14] N. Kobeissy, , M. G. Genet, and D. Zeghlache, "Mapping XML to OWL for Seamless Information Retrieval in Context-Aware Environments". In International Conference on Pervasive Services (Los Alamitos, CA, USA, 2007), IEEE Computer Society, pp. 361-366.

[15] OWL, "Web Ontology Language (OWL)," http://www.w3.org/2004/OWL, 2004.

[16] P. T. T. Thuy, Y. K. Lee, and S. Lee. "XSD2RDFS and XML2RDF Transformation: a Semantic Approach". In The Second International Conference on Emerging Database (EDB 2010), Jeju, Korea. 2010.

[17] P. V. Biron, and A.Malhotra, "XML Schema Part 2: Datatypes Second Edition". Tech. rep., W3C, October 2004. W3C Recommendation.

[18] R. Ghawi, and N.Cullot, "Building Ontologies from XML Data Sources". In 1st International Workshop on Modelling and Visualization of XML and Semantic Web Data Linz, Austria, September 2009.

[19] S.Tschirner, A.Scherp, S.Staab, "Semantic access to INSPIRE". Terra Cognita Workshop (2011).

[20] W3C, OWL Working Group, "OWL 2 Web ontology language document overview. W3C Recommendation 27 October 2009," http://www.w3.org/TR/owl2-overview/.

[21] W3C, OWL Working Group, "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. W3C Recommendation 11 December 2012," http://www.w3.org/TR/owl2- syntax/

[22] Y. An, a. Borgida, and J. Mylopoulos "Constructing complex semantic mappings between XML data and ontologies". The Semantic Web–ISWC 2005.