Journal of Theoretical and Applied Information Technology
15th March 2018. Vol.96. No 5

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

CONVERTING OF AN XML SCHEMA TO AN OWL ONTOLOGY USING A CANONICAL DATA MODEL

[1]ADIL JOUNAIDI, [2]MOHAMED BAHAJ

[1]Phd, Department of Mathematics and Computer Sciences, University Hassan I Settat, Morocco

[2]Prof., Department of Mathematics and Computer Sciences, University Hassan I Settat, Morocco

E-mail : [1]jounaidiadil@gmail.com, [2]mohamedbahaj@gmail.com

ABSTRACT

The eXtensible Markup Language (XML) has known since its beginnings an undeniable success. Defined since its origins as a meta-language facilitating the development of specialized tags Languages, nowadays, many documents benefit from the XML frame. But even if this language is strongly used in the web as a way of data exchange between applications, it still lacks the capacity of defining the web resources and the system that uses them, and also the capacity of expressing the knowledge provided by XML documents. It's these lacks that proposes the Web Ontology Language (OWL) proposes to fill. In fact, OWL is a language for ontologies representation in the context of Semantic Web (SW). It's in this context that we're obliged to come up with a solution that allows migration to the SW in order to follow the WEB evolution.

Among the suggested solutions, our approach is based on the Canonical Data Model (CDM) through the implementation of a set of rules allowing the transformation of an XML schema's definition (XSD) to an OWL ontology. This mapping will transform not only the nodes of an XML file, but also the relationships between these nodes in order to maintain the same structure.

Keywords: *eXtensible Markup Language (XML), Complex type, Web Ontology Language (OWL), XML Schema Definition (XSD), Document Type Definition (DTD), Canonical Data Model (CDM), Ontology, Resource Description Framework (RDF), eXtensible Style Language Transformations (XSLT).*

1. INTRODUCTION

During the last years, XML has been known as a relevant recommendation for storing and exchanging data on the web, in fact, from the moment when two applications agree on a unique data format XML, they can exchange data between each other's.

To ensure these transactions XML document often follow A predefined format expressed either in DTD (Document Type Definition) or in XSD, in other words, these schemas contain the structure knowledge, the data type and the relationships between the elements in the XML document.

Before we talk about the problematic that motivates us to propose this approach, we will start by talking about the advantage of the XML language:

Understandable: it uses a readable human language, and it's easy to understand.

Interoperability: it uses a compatible language with other programing languages such as Java, C++, …

Flexibility: it allows to all users describing their contents easily, by creating their own tags, however this freedom can cause a misunderstanding between the author of the document and the consumer, and consequently an XML element can be interpreted by many vocabulary, it's hard for the machine to make the difference between the elements, and to know their significance.

On the other hands, XML presents some inconveniences, in fact, XML mainly focuses on grammar, there is no way to describe the semantic of a document [1]. Consequently, the problem occurs when the software agents want to understand and reason about these XML data.

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

In order to remedy to the insufficiency of the classic web and the WEB 2.0, researchers proposed a new vision of the web named the Semantic Web. It is within this context our approach take part.

At the present time, many of the data circulating on the WEB are in XML format, and the extraction of data from a database can be in XML format, therefore the core of WEB 1.0 and WEB 2.0 is XML. We can conclude that to migrate from WEB 1.0 or WEB 2.0 to WEB 3.0 (Semantic Web), we must be able to transform the XML file into an OWL ontology.

In fact, to make a sense of the data circulating on the web, we propose a rule-based mapping approach to build an OWL ontology from XML schema, indeed, the Semantic Web is based on RDF [2] data model, which is a standard model for data interchange on the Web.

OWL has been recommended by the W3C as the language of choice for knowledge representation in the so called Semantic Web. In OWL, objects of the domain are represented as interrelated resources and identified by Uniform Resources Identifiers (URI), while attribute values are represented by literals (a string, possibly characterized by a language or a data type) [2].

The purpose of this article is to automate the process of transforming any XML schema file to an OWL ontology, to do this, we proceed in three steps, the first is to determine all the relationships between the complexType of XML Schema, the The second step is to classify all complexType in a CDM to facilitate their transformation and the last one corresponds to the transformation of the CDM to an OWL ontology.

This paper is organized as follows: Section 2 presents the related works where we locate our approach in comparison with the others, Section 3 presents the classification rules of all types of relationships between the complex Type in a CDM. The implementation of rules of an XML schema's transformation to OWL ontology is presented in section 4, while transforming the different links between all nodes of an XML Schema, Section 5 describes the prototype algorithm suggested in sections 3 and 4, Section 6 focuses on the implementation of the obtained result and the experimental study is presented on the section 7, finally, Section 8 contains a conclusion and the future work.

2. RELATED WORKS

Different works have suggested a mapping of Relational Databases (RDB) to RDF or Ontology Web Language (OWL).

In [2] the paper proposes an approach to map the relation schema information into the ontology as concepts, thereafter achieve the attributes, and map them to the properties in the ontology.

The authors of [3] propose methodologies to store XML data into new ORDB data structures, such as user-defined type, row type and collection type. This methodology has preserved the conceptual relationship structure in the XML data, including aggregation, composition and association for XML data retrieval.

Another paper [4] presents an approach of RDF graph generation from relational databases. This approach consists of structures creation of ontology including classes, properties, hierarchy, cardinality and instances creation.

Sedighi SM and Javidan R in [5] propose an approach which enables semantic Web applications to access data stored in relational databases using a corresponding ontology. Domain ontology can be used to formulate relational database queries to simplify the data access of the underlying data sources. This method involves two main phases: the construction of a local ontology from a relational database and a semantic query in a relational database using relational database query language (RDQL). In the first phase, we construct Web ontology language ontology from data in a relational database. In the second phase, this work proposes a technique to automatically extract the semantics of relational databases and transform this information into a representation that can be processed and understood by a machine.

Nora Yahia, Sahar A. Mokhtar and AbdelWahab Ahmed proposed in [6] an approach that allows the automatic generation of the OWL from an XML data source, this solution is very heavy since it should pass from multiple steps to perform the OWL generation, the first step is to transform an XML document to an XML Schema, the second one is to analyze the XML Schema

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

based on the XML Schema Object Model (XSOM), the third one is to use the Java Universal Network/Graph (JUNG) framework on the output of the XSOM in order to generate the XML Shema Graph (XSG) that define the schema in the form of graph, and finally, the fourth step is to use the Jena API while we rely on the XSG in order to generate the OWL entity. Despite the heaviness of this transformation this approach doesn't define the transformation of different links type between the nodes, but rather the nodes of an XSD.

Another interesting approach on transforming XML Schema to OWL ontology is [7], this approach allows to transform an XML to an OWL instance in three steps, first, generate an XSD from an XML, secondly convert the generated XSD to an OWL model based on the eXtensible Stylesheet Language Transformations (XSLT), thirdly and finally, generate an OWL instance based on the generated OWL model.

Even if this approach is very interesting in terms of migration of a structure to another, it still doesn't allow defining every type of relationships between the nodes of an XSD file that are, the inheritance, the composition… in the contrary to our approach.

Other approaches more interesting than [4] are [9, 10, 11] that proposes some solutions to the transformation of an XML Schema to an Ontology, [9, 10] suggest some solutions of transformation based on the XSLT language that it's only a language of style transformation, on the other hand these papers aren't interested in the conversion of types of relationships between XSD nodes, as for the paper [11], it proposes a solution that transform a Document Type Definition (DTD), which is a very poor language in terms of description of relation types between XML file nodes.

In our paper, we propose a set of management rules allowing not only transforming an XSD file to an OWL file but also a modelling of different links between the nodes of an XSD file to a new modelling of these nodes in OWL form which can be used eventually in a Semantic Web application.

3. CDM'S DEFINITION

A canonical data model (CDM) defines the relevant entities for a specific domain, their attributes, their associations and their semantics. As a reference model, the CDM defines the associations, and the types of attributes, it is a method to extend and exchange the schema. The CDM is a data reference model that is designed to allow the sharing of information and data to reuse.

Our CDM is defined as a set of complex types CDM: = {CT | CT: = [ctn, cls, EAcdm, RLcdm, REFcdm]}, where each ComplexType C has a name ctn, has a classification cls, a set of elements and attributes EAcdm (Elements|Attributs cdm), a set of relationship RLcdm (ReLations cdm), and finally keys and keyrefs.

3.1 Classification (cls)

We applied the approach in [12] to classify the ComplexTypes, this approach offers five classes:

3.1.1 Shareable and Existence-Independent Aggregation complex type (SEIA)

If a Complex Type can be shareable with others complex Types and its existence is independent of all of them we can classify this complex type as a (SEIA): "cls=SEIA".

3.1.2 Non-Shareable and Existence-dependent Aggregation complex type (NSEDA)

If a Complex Type that cannot be shareable with other Complex Types, and its existence depends to that of the others, this complex type is classified as "cls=NSEDA".

3.1.3 Association 1: N complex type (A1N)

In this case, if a complex Type contains a reference which can be implemented inside another Complex Type, as its element with maxOccurs "unbounded" , therefore it is classified as "cls=A1N".

3.1.4 Association M: M complex type (AMM)

In the XML Schema for many-to-many association relationship, each types in the association has maxOccurs = « unbounded ». Each element will be linked to another element by using the attribute name that refers to another element ID.

Journal of Theoretical and Applied Information Technology
15th March 2018. Vol.96. No 5

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

In this case we classify the complex type as (AMM).

3.1.5 Inheritance complex type (INHER)

If a complex type extends an existing complexType element, it classified as (INHER) in the CDM.

3.2 Elements|Attributes cdm (EAcdm)

Each complexType element has a set of attributes and EAcdm:= {a | a: = [Ele, Type, MinO/MaxO, Use]}, where each Element|Attribute belongs to a class that we presented at the beginning of this paragraph Cls = {SEIA, NSEDA, A1N, AMN, INHER}, Ele is the name of the element or attribute, type is the type of the element, MinO/MaxO is the minimum/maximum of occurrence, Use is to say that this element or attribute is mandatory or not.

3.3 Cdm's relationships (RLcdm)

Each complexType has a set of relationships with other complexTypes, each relationship rl ? RLcdm between complexTypes C1 and C2 is defined in C1, and represents an association, aggregation, composition, or inheritance. RLcdm = {rl | rl: = [RlType, DirC]}, where RlType is the type of relationship and DirC, is the name of the complexType C2.

3.4 Keys and Keyrefs (REFcdm)

Data dependencies are represented by keys and KeyRefs as for each keyref tag there is a reference to a key of a complex Type. REFcdm= {keys:= [k, kr]}.

4. RULES OF THE MAPPING PROCESS

In this part, we will propose the rules to map XML schema based on different types of relationships (Composition, aggregation …) into OWL ontology.

In order to justify all types of relationships between the nodes of an XSD file, we will rely on our approach [10] where we already defined these kinds of relationships.

To deal with any type of relationship, we will use the XML schema of the purchase order application requisitions (see Fig. 1). This diagram shows not only the connection between the complex type, but also the type of relationship and its semantic constraints.

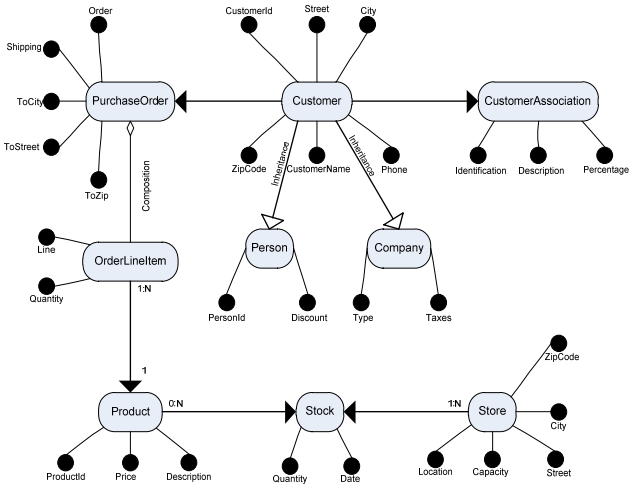

Figure 1 : Purchase order XML document diagram

4.1 Complex Type transformation

Each complex type with element/Attribute should be mapped to a class (see Figure 2):

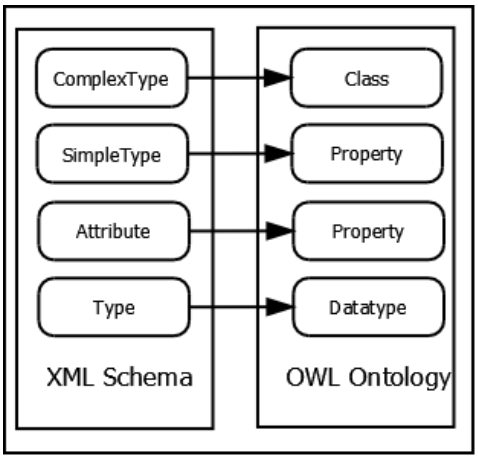

Figure 2 : XML Schema mapping

<owl:Class rdf:ID=″Customer″/>
<owl:Class rdf:ID= ″Customer Association″/>
<owl:Class rdf:ID= ″Purchase Order″/>
<owl:Class rdf:ID= ″Person″/>
<owl:Class rdf:ID= ″Company″/>
<owl:Class rdf:ID= ″Orderlineitem″/>
<owl:Class rdf:ID= ″Products″/>
<owl:Class rdf:ID= ″Stock″/>
<owl:Class rdf:ID= ″Store″/>

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

4.2 Transformation of inheritance relationship

In the XML schema there is an extension <xsd:extension base=Ctype> to show that the element name, that is mentioned before it, is an inheritance with Ctype, then the elements can be mapped to sub-class of the others.

This relationship is classified as an (INHER) in CDM.

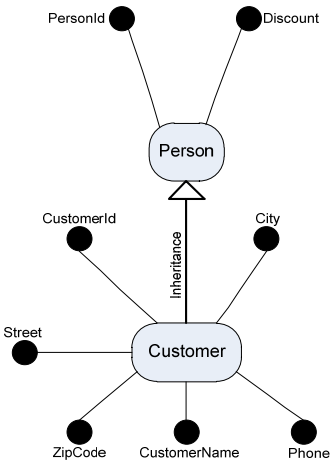

Figure 3 : Inheritance relationship example

<owl:Class rdf:ID=″Person″>
<rdfs:subClassOf rdf:resource=″Customer″/>
</owl:class>

4.3 Transformation of composition relationship

[13] Composition in UML is special kinds of associations between classes.

1) An object must not be part of more than one composition.

2) An object of a class that is part of a composition must not exist without the class it belongs to.

Restriction (1) can be enforced with a FunctionalProperty or InverseFunctionalProperty axiom. If the composition association is navigable bi-directionally the user is free to choose. Otherwise the following rules apply: If the association is navigable from ‘part’ to ‘whole’ a FunctionalProperty is required. A connection from an individual of the 'part' class to more than one individual of the 'whole' class would make the ontology inconsistent. An InverseFunctionalProperty is required if the association is navigable from 'whole' to 'part'.

Restriction (2) impossible the individual might be part of a composition that is simply not listed in the ontology.

Therefore, for two types namely Composant_Type and Composer_Type having a composition relationship, implement one InverseFunctionalProperty, the domain is the class corresponding to the complexType Composant_Type, range is the class referred complexType Composer_Type.

According to the rule 4, we can get one object properties:

<**xsd:complexType** name='Composer_Type'>
<xsd:sequence>
<xsd:element> ...
<**xsd:complexType** name = 'Composant_Type'>
<xsd:sequence>
...
</xsd:sequence>
</**xsd:complexType**>
</xsd:element>
</xsd:sequence>
</**xsd:complexType**>

The transformation to the OWL ontology is as follows:

<owl:**InverseFunctionalProperty** rdf:ID="contain">
<rdf:domain rdf:resource="#**Composer_Type**" />
<rdf:range rdf:resource="# **Composant_Type**" />
</ owl:InverseFunctionalProperty>

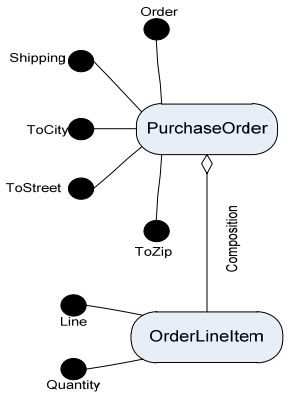

Figure 4 : Composition relationship example

< Owl:Class ID="PurchaseOrder" />
<Owl:Class ID="ORDERLINEITEM_Type" />

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

```
<Owl:InverseFunctionalProperty ID="contain">
  < rdf:domain rdf:resource="#PurchaseOrder" />
  <rdf:range
rdf:resource="#ORDERLINEITEM_Type" />
</InverseFunctionalProperty>
```

This relationship, we can classify it as a Non-Shareable and Existance-dependent Aggregation complex type (NSEDA).

4.4 Transformation of one to many relationship

For two types namely One_Type and Many_Type if One_Type and Many_Type having 1:N association relationship, implement two inverse object properties. For one object property, the domain is the class corresponding to the complexType One_Type; range is the class referred complexType Many_Type.

According to the rule 4, we can get two object properties One_Type.InMany_Type and Many_Type.hasOne_Type :

```
<xsd:complexType name=″ONE_Type″>…
</xsd:complexType>
<xsd:complexType name=″MANY_Type″>
  <xsd:attribute name=″ONE_Key″ …
        maxOccurs=″unbounded″/>…
</xsd:complexType>

<key name=″ONE_Key″>
  <selector xpath=″ONE_Type″>…</key>
<keyref name=″ONE_Key_Ref″
refer=″ONE_Key″>
  <selector xpath=″MANY_Type″>…
</keyref>
```

The transformation to the OWL ontology is as follows:

```
<owl:ObjectProperty
rdf:ID="ONE_Type.inMANY_Type">
        <rdfs:domain
rdf:resource="ONE_Type"/>       <rdfs:range
rdf:resource="MANY_Type"/>
</owl:ObjectProperty>
<owl:ObjectProperty
rdf:ID="MANY_Type.hasONE_Type">
        <rdfs:domain
rdf:resource="MANY_Type"/>     <rdfs:range
rdf:resource="ONE_Type"/>
<owl:inverserOf
rdf:resource="ONE_Type.inMANY_Type">
</owl:ObjectProperty>
```

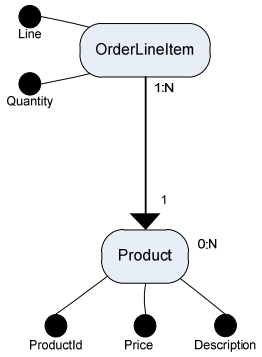

Figure 5 : One to Many association example

```
<owl:ObjectProperty
rdf:ID="Products.inOrderlineitem">
<rdfs:domain rdf:resource="Products"/>
<rdfs:range rdf:resource="Orderlineitem"/>
</owl:ObjectProperty>
<owl:ObjectProperty
rdf:ID="Orderlineitem.hasProducts">
<rdfs:domain rdf:resource="Orderlineitem"/>
<rdfs:range rdf:resource="Products"/>
<owl:inverserOf
rdf:resource="Products.inOrderlineitem"/>
</owl:ObjectProperty>
```

This relationship will be classified as a 1 : N (A1N) Association.

4.5 Transformation of many to many relationship

For two types, namely T1 and T2 having association relationship with T3, implement two object properties, the domain and range of the two object properties is inversed.

According to the rule 5, we can get two object properties T1.T3 and T2.T3T1:

```
<xsd:complexType name=”T1”>
            ...
<xsd:attribute name=“T1_ID” type=“xsd:ID"
        use="required"/>
</xsd:complexType>
<xsd:complexType name=”T2”>...
<xsd:element name=“T3”
        maxOccurs=”unbounded”>
        <xsd:complexType>
            <xsd:sequence>...
            <xsd:attribute name=“T1_ID”
            type=“xsd:IDREF"/>
```

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

```
  </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:attribute name="T2_ID" type="xsd:ID"
use="required"/>
</xsd:complexType>
<key name="T1_ID_Key">
        <selector xpath="T1"/>
<field xpath="@T1_ID"/>
</key>
<keyref name="T1_ID_T3_Ref"
refer="T1_ID_Key">
         <selector path="T2/T3"/>
         <field xpath="T2_ID"/>
</keyref>
```

The transformation to the OWL ontology is as follows:

```
<owl:ObjectProperty rdf:ID="T1.T3">

<rdfs:domain rdf:resource="T1"/>
  <rdfs:range rdf:resource="T2"/>
  </owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="T2.T3T1">

<rdfs:domain rdf:resource="T2"/>
  <rdfs:range rdf:resource="T1"/>
  <owl:inverserOf rdf:resource="T1.T3"/>
</owl:ObjectProperty>
```

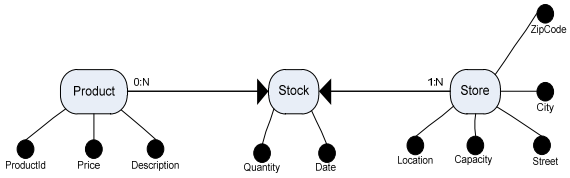

Figure 6 : Many to Many association example

```
<owl:ObjectProperty rdf:ID="Products.Stock">
  <rdfs:domain rdf:resource="Products"/>
  <rdfs:range rdf:resource="Store"/>
</owl:ObjectProperty>

<owl:ObjectProperty
rdf:ID="Store.StockProducts">
  <rdfs:domain rdf:resource="Store"/>
  <rdfs:range rdf:resource="Products"/>
  <owl:inverserOf rdf:resource="Products.Stock"/>
</owl:ObjectProperty>
```

This case can be classified as an M: M (AMM) Association.

5. CDM'S GENERATION FROM AN XML SCHEMA

After we defined all types of classification, we took the example from [8] to generate the CDM from the XML schema described in Table 1.

The extract of the proposed algorithm defines The CDM's transformation and all the rules cited before in order to perform the translation of a CDM to the OWL file is described in figure 9.

6. IMPLEMENTATION

During the implementation of our algorithm, we used JAVA language and the DOM API that allows parsing all the nodes of the XSD file and translate it to OWL ontology.

Below an extract of the source code implemented:

```
public void TransformInheritenceRelationShip(Element racine) {
    List listComplexType = racine.getChildren();
    Iterator i = listComplexType.iterator();

    while (i.hasNext()) {
        Element courant = (Element)i.next();

        if (courant.getName().toLowerCase().equals("complextype")) {
            String strSuperClassName    = getSuperClass(courant);
            if (strSuperClassName != "") {
                String strSubClassName      = getOwlNameClass(courant);

                addSubClassElement(strSuperClassName, strSubClassName);
            }
        }

        TransformInheritenceRelationShip(courant);
    }
}
```

Figure 7 : An extract source code

7. EXPERIMENTAL STUDY

To demonstrate the validity of our approach, a prototype has been developed by making the algorithm above. The algorithm was implemented using Java and JENA API in order to generate the OWL ontology in a first step.

In a second step, we have examined the differences between the XML schema source and the OWL ontology generated by the prototype, through the query results provided by SPARQL in the JENA API and the ones provided by XQUERY in the stylus studios. The queries have returned the same results. The XML source database is transformed to the OWL target ontology without data loss.

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

```
<owl:Class ID="Customer" />
<owl:Class ID="Order" />
<owl:Class ID="person">
  <SubClassOf resource="Customer" />
</owl:Class>
<owl:Class ID="CustomerType" />
<owl:Class ID="AddressType" />
<owl:Class ID="OrderType" />
<owl:Class ID="ShipInfoType" />
<owl:ObjectProperty ID="Customer.inOrder">
  <rdf:domain resource="Customer" />
  <rdf:range resource="Order" />
</owl:ObjectProperty>
<owl:ObjectProperty ID="Order.hasCustomer">
  <rdf:domain resource="Order" />
  <rdf:range resource="Customer" />
</owl:ObjectProperty>
<owl:ObjectProperty ID="Product.Stock">
  <rdf:domain resource="Product" />
  <rdf:range resource="Store" />
</owl:ObjectProperty>
<owl:ObjectProperty ID="Store.StockProduct">
  <rdf:domain resource="Store" />
  <rdf:range resource="Product" />
  <inverserOf resource="Product.Stock" />
</owl:ObjectProperty>
```

Figure 8 : An extract of the generated OWL file

For the validation of our work, we made several queries on the XML document and the generated ontology: all the answers are identical. We are presenting, some queries that are applied on the XML schema indicated in [8] and their equivalent in the SPARQL generated by the prototype. The table 2 represents the description and the result of each query.

8. CONCLUSION

Our paper proposed an automatic solution to convert an XML schema to an OWL ontology based on a Canonical Data Model CDM. This method will help storing web documents scattered in an OWL ontology so that the users will be able to manage them more easily. Our method is to create a CDM from an XML schema and we used it as an input to a java program, and this last will generate an OWL ontology. The java program was coded in order to respect all the content of the XML schema and the different relationships. In the end, we applied some queries on the XML schema and the ontology generated by the program, the tests proved what we mentioned earlier, and we also noticed a resemblance between the data in the OWL ontology and the CDM, and of course the XML schema.

In the future, we intend to propose a solution allowing the optimization of SPARQL queries, especially in the case of a large ontology.

REFRENCES:

[1] Yutao Ren, Lihong Jiang, Fenglin Bu, Hongming Cai “Rules and Implementation for Generating Ontology from Relational Database” in Second International Conference on Cloud and Green Computing, 2012.

[2] Graham Klyne, Jeremy J. Carroll, Resource Description Framework (RDF): Concepts and Abstract Syntax (W3C Recommendation 10 February 2004) [EB/OL]. http://www.w3.org/TR/rdf-concepts/, (last modified on 10 February 2004).

[3] Eric Pardede, J. Wenny Rahayu, David Taniar, “Object-relational complex structures for XML storage” in Information and Software Technology 48 :6, 2006, pages.370-384.

[4] H.Ling, S.Zhou, “Mapping Relational Databases into OWL Ontology”, In International Journal of Engineering and Technology, Volume 5, Issue 6, 2013, Pages 4735-4740.

[5] Sedighi SM, Javidan R. “Semantic query in a relational database using a local ontology construction”. S Afr J Sci. 2012;108(11/12), Art. #1107, 10 pages. http://dx.doi.org/10.4102/sajs. v108i11/12.1107.

[6] Nora Yahia, Sahar A. Mokhtar, AbdelWahab Ahmed, “Automatic Generation of OWL Ontology from XML Data Source”, International Journal of Computer Science Issues, Volume 9, Issue 2, March 2012.

[7] Hannes Bohring, Sören Auer, “Mapping XML to OWL Ontologies”, Leipziger Informatik-Tage, volume 72 of LNI, 2005, pages 147-156, GI.

[8] Adil JOUNAIDI, Doha MALKI, Mohamed BAHAJ, Ilias CHERTI. “Conversion of an XML schema to object relational databases using a canonical data model”, Vol 93 November 2016 Issue of Journal of Theoretical and Applied Information Technology.

[9] Chrisa Tsinaraki, Stavros Christodoulakis: XS2OWL: A Formal Model and a System for Enabling XML Schema Applications to Interoperate with OWL-DL Domain Knowledge and SW Tools, DELOS,pages. 137-146 (2007).

[10] Chrisa Tsinaraki and Stavros Christodoulakis, “Interoperability of XML

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

Schema Applications with OWL Domain Knowledge and Semantic Web Tools", pages. 850–869, OTM Conference, Springer-Verlag, 2007.

[11] Bernd Amann, Catriel Beeri, Irini Fundulaki, Michel Scholl: Ontology-Based Integration of XML Web Resources. 1st Int. Semantic Web Conf, pages. 117-131, Springer, 2002.

[12] Eric Pardede, J.Wenny Rahayu, David Taniar "On Using Collection for Aggregation and Association Relationships in XML Object-Relational Storage", 2004 ACM Symposium on Applied Computing, 2004.

[13] ZEDLITZ, Jesper; JÖRKE, Jan; LUTTENBERGER, Norbert. From UML to OWL 2. In: Knowledge Technology. Springer Berlin Heidelberg, 2012. S. 154-163.

Journal of Theoretical and Applied Information Technology
15th March 2018. Vol.96. No 5

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

Table 1 : CDM generated from the XML Schema to be converted to OWL Ontology

cn	cls	eacdm				rlcdm		k/kr	
		ele	typ	mino/maxo	use	rltype	dirc	key	keyr
customer	seia	customerid	xsd:id		required			k	
		customername	xsd:string						
		orderid	xsd:integer	unbounded		Asso	purchase_order		kr
customer_association		identification	xsd:string						
		description	xsd:string						
purchase_order	a1n	orderid	xsd:id		required			k	
		shipping	xsd:string						
orderlineitem	nseda	line	xsd:integer			comp	purchase_order	k	
		productid	xsd:integer			comp	purchase_order		
		quantity	xsd:integer			comp	purchase_order		
products		productid	xsd:id		required			k	
		description	xsd:string						
		price	xsd:decimal						
store		location	xsd:id		required			k	
		capacity	xsd:integer						
stock	amn	productid	xsd:integer			asso	store		kr
		quantity	xsd:integer			asso	store		
person	inher	personid	xsd:id		required	inherby	customer	k	
		discount	xsd:integer			inherby	customer		
company	inher	type	xsd:string			inherby	customer		
		taxes	xsd:integer			inherby	customer		

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

BuildOWL

```
For (Cdm c:cdm)
  Evalaute (c.cls)
    Case 'INHER':
      ComplexTypeTransformation
      InheritanceTransformation
    Case 'NSEDA':
      ComplexTypeTransformation
      CompositionTransformation
    Case 'A1N':
      ComplexTypeTransformation
      OneToMayTransformation
    Case 'AMM':
      Implement the third rule ComplexTypeTransformation
      Implement the fifth rule ManyToManyTransformation
  End Evaluate
End For
```

ComplexTypeTransformation

```
Iterate on the list of nodes from XSD
For each (XSD e)
 If (e is a complexType)
          Create an OWL Class
 End if
 If (e is a simpleType)
          Create an OWL Property
 End if
 If (e is an attribute)
   Create an OWL Property
 End if
 If (e is a type)
          Create an OWL Datatype
 End if
End for each
```

InheritanceTransformation

```
Iterate on list of nodes from XSD
For each (XSD e)
  If (e is a complexType)
    If (e has an extension node)
      Create e an OWL subClassOf
    End if
  End if
End for each
```

CompositionTransformation

```
Iterate on the list of nodes from XSD
For each (XSD e)
  If (e has an element that contents an XSD complexType
    Create an OWL InverseFunctionalProperty
  End if
End for each
```

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

OneToMayTransformation
Iterate on the list of nodes from XSD
For each (XSD e1)
  If 'e1' has an element key 'k' and 'k' has a
    keyRef that refers another element 'e2'
    Implement two inverse object properties.
    For one object property, the domain is the class
    Corresponding to e1; range is the class referred e2.
  End if
End for each

ManyToManyTransformation
Iterate on the list of nodes from XSD
For each (XSD e1)
  If 'e1' has an element key 'k' and 'k' has a
    keyRef that refers another sub element 'e2/e3'
    Implement two object properties,
    The domain and range of the two
    object properties are inversed
  End if
End for each

Figure 9 : Algorithm to convert the XML Schema to an OWL ontology

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

TABLE 2 : EXPERIMENTAL STUDY

Description	SPARQL	Xquery	Result

Find the name of all Customers of the customer_association Identified by "ASS1" Ordered by name of customer	PREFIX foaf: <http://xmlnx.com/foaf/0.1/>
	PREFIX foo: <http://example.com/resources/>
	
	SELECT ?customerId ?customerName
	WHERE {
	?customerId foaf:name ?customerName .
	?customerId foo:identification "ASS1" .
	}
	ORDER BY ASC(?customerName)
		for $ca in doc('customer.xml')/NewDataSet/Customerassociation ,
		$id in $ca/identification ,
		$c in $ca/Customer
		where $ca/identification='ASS1'
		order by $c/customerName
		return
		<customer>
		{$c}
		</customer>
			12
			Dupont
			10
			Scott
			11
			Smith

The first customer name of the customer_association identified by "ASS1"	PREFIX foaf: <http://xmlnx.com/foaf/0.1/>
	PREFIX foo: <http://example.com/resources/>
	
	SELECT *
	WHERE {
	?customerId foaf:name ?customerName .
	?customerId foo:identification "ASS1" .
	}
	LIMIT 1
		for $ca in doc('customer.xml')/NewDataSet/Customerassociation,
		$id in $ca/identification ,
		$c in $ca/Customer[1]
		where $ca/identification='ASS1'
		return
		<customer>
		{$c}
		</customer>
			10
			Scott
			123456789

Compute the number of all Customer of customer_association	PREFIX foo: <http://example.com/resources/>
	
	SELECT ?customerIdentification COUNT(*) AS ?count
	WHERE {
	?customerId foo:identification ?customerIdentification .
	}
	GROUP BY ?customerIdentification
		for $x in doc('customer.xml')/NewDataSet/Customerassociation
		return
		{$x/identification }
		{number=count($x/Customer)}
			ASS1 3
			ASS2 2
			ASS3 1

Find the orders made by the customer named « Dupont »	PREFIX foaf: <http://xmlnx.com/foaf/0.1/> PREFIX foo: <http://example.com/resources/> SELECT ?customerName ?productName WHERE { ?purchaseOrder foo:shippingName ?customerName . ?purchaseOrder foo:item ?productName . ?customerName	for $po in doc('customer.xml')/NewDataSet/PurchaseOrder, $name in $po/Address/Name, $i in $po/Items/Item where $name='Dupont' return {$name} {$i/ProductName}	Dupont Computer Keyboard Dupont Wireless Mouse

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

	foaf:name "Dupont" . }		

Find the name of all Customers that starts with « S »
Identified by “ASS1”
Ordered by name of customer
	PREFIX foaf:
	<http://xmlnx.com/foaf/0.1/>
	PREFIX foo:
	<http://example.com/resources/>
	
	SELECT ?customerId ?customerName
	WHERE {
	?customerId foaf:name ?customerName .
	?customerId foo:identification "ASS1" .
	FILTER regex(?customerName, "^S")
	}
	ORDER BY ASC(?customerName)
		for $ca in doc('customer.xml')/NewDataSet/Customerassociation ,
		$id in $ca/identification ,
		$c in $ca/Customer
		where $ca/identification='ASS1'
		satisfies starts-with($c/customerName, 'S')
		order by $c/customerName
		return
		<customer>
		{$c}
		</customer>
			10
			Scott
			11
			Smith