# Exploiting XML Schema for Interpreting XML Documents as RDF

Pham Thi Thu Thuy, Young-Koo Lee, Sungyoung Lee, and Byeong-Soo Jeong
*Department of Computer Engineering, Kyung Hee University, Korea*
*{tttpham, sylee}@oslab.khu.ac.kr, {yklee, jeong}@khu.ac.kr*

## Abstract

*Interpreting legacy XML documents is a great challenge for realizing the vision of the Semantic Web (SW). This paper presents an algorithm to transform XML data into RDF- foundation language of the SW - automatically. Our approach maps element definitions stored in XML Schema to RDF Schema ontology, where the ontology is used to describe the meaning and relationships between XML elements. The RDF results containing XML data at the semantic level while retaining their nesting structure make huge XML data source on the current web be available for the SW.*

## 1. Introduction

XML has received a wide acceptance as a standard for communication on the web. The main success of XML is its flexibility. Users can define their own tags to describe elements in the XML document. Moreover, they can also predefine the structure of XML documents by writing a DTD (Document Type Definition) or an XML Schema. Although DTD and XML Schema provide the structure for XML document, XML Schema is an XML-based language, and it supports data types and namespaces [1]. Therefore, XML Schema is widely used as a standard mechanism to interchange information on the web. For instance, in the electronic commerce, when the associates are unanimous in a common XML Schema, they will produce valid XML documents and carry out their exchange. This provides us a large number of valid XML documents.

However, XML has disadvantages when coming to the semantic interoperability. XML mainly focuses on the grammar but there is no way to describe the semantics of the document [3]. Moreover, because XML enables users to define their own tags, an object can be described in different ways. For example, we label something as <price>$12.00</price> and another organization labels the same field as <cost>$12.00</cost>. In this case, a machine cannot differentiate between two meanings unless the SW technologies such as RDF are added [2]. Furthermore, in the SW, the operability requires not only the structured data but also the semantic content [3]. Therefore, we cannot directly use XML data on the SW, another language that is supported by the SW is needed to interpret these data.

Though, the general purpose language for representing information in the Semantic Web is RDF, it cannot describe classes and properties in structured documents. Instead, they are depicted by the RDF schema [2]. It defines a vocabulary for creating class hierarchies, properties of class, and adding instance data. Furthermore, the data model for XML is a tree-like [4], while RDF is a graph-based data model which is a collection of the subject-predicate-object triples [5]. Hence, we try to exploit the tree structure of XML by accessing to the XML Schema to generate corresponding class hierarchy in RDF. Our main contribution is a set of rules that derive classes and properties from XML Schema and automatically interpret all XML elements as RDF triples by using RDF schema vocabularies. The generated result follows in XML structure and provides much semantics about XML data which can be use by SW.

The remainder of the paper is organized as follows. In section 2, we briefly introduce the related work. Section 3 describes the mapping notation from XML Schema to ontology and the algorithm and the corresponding example. Finally, section 4 concludes this paper.

## 2. Related work

Recently, several strategies for interpreting XML data as semantic sources have been proposed. Some of them transform XML into RDF and the others map XML to OWL. However, there is no completed approach targeted on interpreting XML as RDF by using XML Schema.

Melnik [6] assumed that every XML document had an RDF model and described a mapping from XML to RDF. However, he just focused on how to transform all XML elements into RDF and did not concern about exploiting domain's information. Therefore, the issues followed the structures of XML but bore little meaning and did not fit well into RDF model. Our method aims at drawing the semantic information based on XML Schema, therefore the mapping result still remains structure of XML and provides more semantics for XML documents.

C-Web project [7] replied on DTD to define the meaning for every XML element and used XPath to map information in XML documents to domain specific ontology. This proposal exploited more specific meaning and structure of the XML documents. However, beside reference to XML document and its DTD, it required referring to the specification of rules which is not needed in our approach.

Klein [8] introduced a procedure to transform XML data into RDF data by annotating the XML documents via external RDF Schema specifications. This approach is close to our method. However, it only translated some pieces of information in the document. Moreover, elements in XML document were decided to be classes or properties depending on user's opinion. Our approach transforms all the XML document and draw classes and properties based on element definitions in the XML Schema, therefore human intervention is not necessary.

In the previous work [9], we proposed a procedure for transforming valid XML documents into RDF via RDF Schema. This procedure derived classes and properties from DTD, then matched them with elements in XML documents and interpreted all XML data as RDF statements. However, as we mentioned, XML Schema is more and more popular than DTD. Furthermore, sometimes definitions in DTD or XML Schema have mal-function, we cannot completely rely on them. In this paper, in the mapping stage from XML Schema to ontology, we specifically consider the content in every element before deciding it is a class or property. Moreover, we expand RDF Schema by defining new RDF Schema property to describe the nesting relationship of XML Schema in the RDF Schema.

Ferdinand et al. [10] proposed mapping rules from XML to RDF and from XML Schema to OWL ontology. However, the latter results from XML Schema may not suit to OWL model. Furthermore, the mapping from XML to RDF just concentrated on how to translate all XML elements into RDF and did not focus on meaning of elements. Therefore, this drawback is the same to [6].

Besides, there are several approaches creating new OWL ontology for XML Schema [11, 12]. Rodrigues et al. [13] invented a mapping notation for every XML Schema and transform XML documents into existing OWL. This approach provided more specific mapping but users have to define relations for every XML element. Our target is not at OWL but in RDF, which is the foundation language for the SW.

To our knowledge, most of the related researches which are found during our study have not supported the transforming from XML Schema into existing RDF. Hence, our research is a unique work which makes a contribution to the SW applications. This paper proposes a strategy to map XML Schema to the ontology (with considering the proper functions of classes and properties) and automatically interprets valid XML data (of that XML Schema) as existing RDF statements which can be directly used by the SW. Nevertheless, if XML Schema is absent, we can also create ontology based on XML document. Hence, we can tackle the problem when XML Schema is not available.

# 3. XML transforming

## 3.1. XML Schema mapping

In this stage, we create the collection of classes and properties from the given XML Schema as an input. This collection will be used to model data in the next step. The mapping notation from XML Schema to RDF Schema is shown in Table 1.

TABLE 1: XML SCHEMA MAPPING

| XML Schema concepts | RDF Schema concepts |
|---|---|
| Complex-type element | Class |
| Simple-type element | Property |
| Attribute | Property |
| Type | Datatype |

For specifying properties and data type, we use *rdf:Property* and *rdf:datatype* which are available vocabularies of RDF. However, in order to describe the relationship between classes, we decide to extend RDF Schema's vocabulary to ensure the semantics and the structure of XML documents. Particularly, we define new RDF Schema property, *rdfx:contain* (*rdfx* stands for namespace where *contain* is defined), to describe a class contains a sub-class. We avoid using *rdfs:subClassOf* because sometimes in the XML documents one class is belong to another class but it is not exactly a sub-class. For instance, in section 3.3 class *article* includes class *author* but *author* cannot be a sub-class of *article* according to database definition. Furthermore, in order to make our procedure perform independently from human intervention, *rdfx:contain* is the correct choice.

## 3.2. XML transforming algorithm

After deriving classes and properties from an XML Schema, we continue to examine the valid XML document. The URI of the XML document will be the subject of the first statement. The algorithm starts traversing from the beginning of the XML document and finishes when it meets the close tag of root element.

Because instances of RDF classes are characterized by having unique identifier, when generating RDF instances statements, we have to be sure that only one identifier is created for each individual. In our procedure, if there is more than one element with the same name, and in the same levels in the XML tree, another property of each node is added as its ID. In that case, *rdf: ID* is inserted to connect this property to the XML node. The ID is the name of XML node concatenating number 1, number 2 for the second node and so on.

For every element in the XML document, we verify whether it is a name of a class or a property in the generated ontology. If it does not match with any class or property, we skip it and continue to the next element. Based on this matching, our procedure decides what RDF statements should be created.

1. If the tag matches with a class, consider three cases:

a) If this is the root-class, create the first statements:

| | | |
|---|---|---|
| URI of document | rdfs: Resource | root-class |
| Root-class | rdf: Namespace | Namespace |

These statements are used once in our procedure. Since in an XML document, there is only one root-class and all other classes are its children, when we meet the root-class, we use rdfs: Resource to connect the resource of XML document (URI) to the root-class of the document. The property rdf: Namspace is used to connect XML namespace to the root class.

b) Else, this class can be a child of root-class or another class. If the previous statement is unfinished (statement with only two properties such as subject and predicate are filled, the object is empty), complete this statement by supplementing the parent class in the object and add one or two more statement to describe this class.

| | | |
|---|---|---|
| | | parent-class-name |
| parent-class-name | rdfx: contain | Class-name |

c) Else, create the new statement (simple case of b):

| | | |
|---|---|---|
| parent-class-name | rdfx: contain | Class-name |

It means when finding out a class, we have to specify its parent. Moreover, if there is more than one Class-name with the same name in the same level, and there is no element ID definition for it, one more statement is added:

| | | |
|---|---|---|
| Class-name | rdf: ID | Class-name_number |

2. If the element associates with a property, we verify the class which this property describes for and predict the property value. However, because our RDF statements are sometimes unfinished, we consider two cases:

a) If the previous statement is unfinished, complete it with the name of class which this property belongs to, and create new unfinished statement:

| | | |
|---|---|---|
| | | Class-name |
| Class-name | rdf: Property | property-name |
| Property-name | rdf:datatype | data type |
| property-name | rdf: literal | |

rdf: literal is declared for the value of this property.

b) Else, we also describe which class this property depicts for and create an unfinished statement:

| | | |
|---|---|---|
| Class-name | rdf: Property | property-name |
| Property-name | rdf:datatype | data type |
| property-name | rdf: literal | |

3. If it does not match a class or property, we check whether it is a value of a class/property or not.

a) If pervious statement is unfinished, it is surely a value of a property. Because in previous statements, only statements describe for a property is always unfinished statements, we add this value to this empty column:

b) Else, so this value is belong to a class. We describe which class has this value by following statement:

| | | |
|---|---|---|
| Class-name | rdf: literal | value |

## 3.3. Example

In order to illustrate for our procedure, we use sample files "OracleCatalog.xsd" and "OracleCatalog.xml" at http://www.oracle.com/technology/pub/articles/vohra-xmlschema.html. OracleCatalog contains the information about a catalog of an Oracle magazine. Each catalog combines several magazines which also hold a series of articles. Every article is presented by its title and respective author's names. The sample file is as below:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs=
                "http://www.w3.org/2001/XMLSchema">
<xs:element name="catalog">
<xs:complexType> <xs:sequence>
  <xs:element ref="magazine" minOccurs="0"
    maxOccurs="unbounded" /> </xs:sequence>
  <xs:attribute name="title" type="xs:string" />
  <xs:attribute name="publisher" type="xs:string" />
  </xs:complexType> </xs:element>
<xs:element name="magazine">
<xs:complexType> <xs:sequence>
  <xs:element ref="article" minOccurs="0"
    maxOccurs="unbounded" /> </xs:sequence>
  <xs:attribute name="date" type="xs:string" />
  </xs:complexType> </xs:element>
 <xs:element name="article"> <xs:complexType>
<xs:sequence> <xs:element name="title" type="xs:string" />
  <xs:element ref="author" minOccurs="0"
    maxOccurs="unbounded" /> </xs:sequence>
  </xs:complexType> </xs:element>
<xs:element name="author">
 <xs:complexType> <xs:sequence>
  <xs:element name="firstname" type="xs:string" />
  <xs:element name="lastname" type="xs:string" />
  </xs:sequence> </xs:complexType> </xs:element>
```

Using rules in XML Schema mapping, results are presented in the fig.1 below:

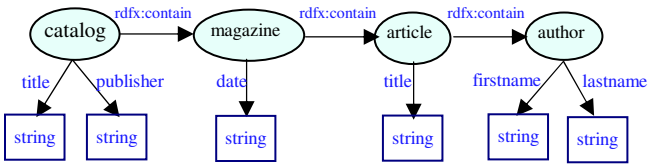


FIG.1. RDF ONTOLOGIES TRANSFORMED FROM XML SCHEMA

After having the set of classes and properties from the previous step, we scan the XML file, "OracleCatalog.xml", to produce RDF statements by using algorithm in XML transforming section. Because the file is quite long with two magazines, each of them have so many articles, we just pick the first magazine to analyze. Following is XML file with the first magazine:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <catalog xmlns:xsi=
          "http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="OracleCatalog.xsd"
      title="Oracle Magazine" publisher="Oracle Publishing">
  <magazine date="November-December 2003">
  <article> <title>Updating XQuery</title>
  <author> <firstname>Jason</firstname>
    <lastname>Hunter</lastname>
    </author> </article> <article>
    <title>Servlets and JSP Step Up</title>
  <author> <firstname>Budi</firstname>
    <lastname>Kurniawan</lastname>
    </author>  </article> </magazine>  ... </catalog>
```

The XML document above is based on real messages with a web-service for online searching. Users can find the detail information for each magazine in a catalog.

This XML document is interpreted as RDF triples based on the generated ontology presenting in the table 2.

TABLE 2: RDF STATEMENTS FROM THE XML DATA

| Subject | Predicate | Object |
|---|---|---|
| …xmlschema.html | rdfs: Resource | Catalog |
| Catalog | rdf:Namespace | …/XMLSchema-instance |
| Catalog | rdf: Property | Title |
| title | rdf:datatype | "string" |
| title | rdf: literal | "Oracle Magazine" |
| Catalog | rdf: Property | publisher |
| publisher | rdf:datatype | "string" |
| publisher | rdf: literal | "Oracle Publishing" |
| Catalog | rdfx: contain | magazine |
| magazine | rdf: Property | Date |
| date | rdf:datatype | "string" |
| date | rdf: literal | "November-December 2003" |
| magazine | rdfx: contain | article |
| article | rdf:ID | article_1 |
| article | rdf: Property | Article |
| title | rdf:datatype | "string" |
| title | rdf: literal | "Updating XQuery" |
| article | rdfx: contain | Author |
| author | rdf: Property | firstname |
| firstname | rdf:datatype | "string" |
| firstname | rdf: literal | "Jason" |
| author | rdf: Property | Lastname |
| lastname | rdf:datatype | "string" |
| lastname | rdf: literal | "Hunter" |
| magazine | rdfx: contain | Article |
| article | rdf:ID | article_2 |
| article | rdf: Property | Title |
| title | rdf:datatype | "string" |
| title | rdf: literal | "Servlets and JSP Step Up" |
| article | rdfx: contain | Author |
| author | rdf: Property | firstname |
| firstname | rdf:datatype | "string" |
| firstname | rdf: literal | "Budi" |
| author | rdf:Property | Lastname |
| lastname | rdf:datatype | "string" |
| lastname | rdf: literal | "Kurniawan" |

Moreover, the data types of property are only *string*, so the value of *rdf:datatype* is always *string*.

Besides, our procedure can also work well if there is no XML Schema. In that case, we draw classes and properties from element function in the XML document. If element contains only value and has no other attribute, we can consider it as a property, otherwise, it is a class. In general, when XML Schema is absent, our procedure needs some human observations. Otherwise, it can generate RDF statements automatically.

## 4. Conclusion

Our procedure outperforms the existing methods due to the following three reasons. Firstly, while transforming all the elements of an XML document into RDF, our algorithm retains the original structure and captures the implicit semantics in the structure of the XML document. Secondly, elements in XML are clarified in classes or properties based on their definition and detail descriptions in XML Schema, this makes the result independent from users' opinions. Finally, languages used in our procedure do their jobs as their original functions. XML Schema is used for defining XML structure, XML for describing data, RDF for providing triple statements about data, and RDF schema for supporting vocabularies to describe the relationship among data. We hope that the research has created a bridge to narrow the gap between the XML and RDF. If this procedure is executed, a large amount of the XML data on the current Web will be interpreted into RDF statements which are useful for the SW.

ACKNOWLEDGMENT

This work was supported by the Korea Science and Engineering Foundation (KOSEF) and the Korean Government (MOST) through the NRL Program (No. R0A-2007-000-20101-0).

## References

[1] [1] Priscilla Walmsley, "XML Schema part 0: Primer second edition", 2004, *W3C Recommendation*, available at: http://www.w3.org/TR/xmlschema-0/

[2] Frank Manola, Eric Miller, "RDF Primer", 2004, *W3C Recommendation,* http://www.w3c.org/TR/REC-rdf-syntax/

[3] S. Decker, S. Melnik, F. V. Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks, "The Semantic Web: The Roles of XML and RDF", 2000, *IEEE Internet Computing*.

[4] Bert Bos, "The XML data model", August 2005, http://www.w3.org/XML/Datamodel.html

[5] Graham Klyne, Jeremy J. Caroll, and Brian McBride, "Resource Description Framework (RDF): Concepts and Abstract Syntax", *W3C Recommendation,* 2004

[6] Sergey Melnik, "Bridging the gap between RDF and XML", 1999, http://www-db.stanford.edu/melnik/rdf/syntax.html

[7] B.Amann, I.Fundulaki, M.Scholl, C.Beeri, and A-M.Vercoustre, "Mapping XML fragments to community Web ontologies", 4th *International Workshop on the Web and Databases*, 2001.

[8] Michel Klein, "Interpreting XML via an RDF Schema", 2002, *Database and Expert Systems Applications*.

[9] Thuy Pham, Young-Koo Lee, Sungyoung Lee, and Byeong-Soo Jeong, "Transforming Valid XML Documents into RDF via RDF Schema", October 2007, *3rd International Conference on Next Generation Web Services Practices, IEEE Computer Science*.

[10] Matthias Ferdinand, Christian Zirpins, and David Trastour, "Lifting XML Schema to OWL", 2004, *Web Engineering – 4th International Conference, ICWE*, pp. 354–358.

[11] Roberto García, Ferran Perdrix, and Rosa Gil, "Ontological Infrastructure for a Semantic Newspaper", 2006, *Semantic Web Annotations for Multimedia Workshop,* SWAMM'06, UK.

[12] Hannes Bohring, and S¨oren Auer, "Mapping XML to OWL Ontologies", 2005, *Marktplatz Internet: Von e-Leanrning bis e-Payment,*LIT2005, Germany, pp. 147-156.

[13] Toni Rodrigues, Pedro Rosa, and Jorge Cardoso, "Mapping XML to Existing OWL Ontologies", 2006, *International Conference WWW/Internet*.

[14] Priscilla Walmsley, "XML Schema part 0: Primer second edition", 2004, *W3C Recommendation*, available at: http://www.w3.org/TR/xmlschema-0/

[15] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin, "Namespaces in XML 1.0", 2006, *W3C Recommendation*, http://www.w3.org/TR/REC-xml-names/

[16] Michel Klein, Dieter Fensel, F.V. Harmelen, and Ian Horrocks, "The relation between ontologies and XML schemas", 2001.