XSD2RDFS and XML2RDF Transformation: a Semantic Approach

Pham Thi Thu Thuy
Dept. of Computer Engineering,
Kyung Hee University,
Yongin-si, Republic of Korea,
tttpham@oslab.khu.ac.kr

Young-Koo Lee
Dept. of Computer Engineering,
Kyung Hee University,
Yongin-si, Republic of Korea,
yklee@ khu.ac.kr

Sungyoung Lee
Dept. of Computer Engineering,
Kyung Hee University,
Yongin-si, Republic of Korea,
sylee@oslab.khu.ac.kr

Abstract— **XML and its schema language are becoming a primary data exchange format in e-commerce. However, they mainly focus on the data structure and there is no way to describe the semantics of the document. In the vision of the Semantic web, data is not only being structured but also containing meaning and the relationship among them. Therefore the need for transforming current XML data into suitable form for the Semantic Web is very necessary. In this paper, we propose a set of notations to map XML Schema to RDF Schema and provide algorithm to interpret XML documents as RDF. The main advantage of our approach is to ensure the integrity of the structure and provide more meaning for the original XML document while automatically transforming them into RDF. This procedure can be used for any valid XML documents.**

Keywords-XSD, XML, RDF Schema; RDF; transformation

I. 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 (or called XSD). Although DTD and XSD provide the structure for XML document, many developers nowadays use XSD to create an XML document instead of DTD. XSD supports data types and namespaces. Therefore, XSD is usually 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 XSD, 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 [1]. Moreover, because XML enables users to define their own tags, an object can be described in different ways. In the Semantic Web, the operability requires not only the structured data but also the semantic content [1]. Therefore, current XML data cannot be used directly by the Semantic Web instead of interpreting them as a standard format of the Semantic web, particularly as RDF.

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 Vocabulary Description Language, RDF schema. 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 [2], while RDF is a graph-based data model which is a collection of the subject-predicate-object triples [3]. 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 RDF Schema from XSD and automatically interpret all XML elements as existing RDF triples which can be used immediately by the Semantic Web.

The remainder of the paper is organized as follows. In section 2, we briefly introduce the related work. Section 3 describes the mapping algorithm from XML Schema to RDF Schema. This will be followed by the XML transforming algorithm and the corresponding example in section 4. Finally, section 5 concludes this paper.

II. RELATED WORK

There are s number of approaches exist today to perform transformation from XML to RDF format, as well as the mapping from the XML Schema to the RDF Schema. However, there is no completed approach targeted on interpreting XML instances as RDF statements by replying on mapping from XML Schema to RDF Schema.

Melnik [4] assumes that every XML document has an RDF model and describes a mapping from XML to RDF. However, the author mainly focuses on how to transform all XML elements into RDF and does not concern about exploiting domain's information. Therefore, the issues follow the structures of XML but bear little meaning and the results do not fit well into existing RDF model instead of new RDF syntaxes. Our method aims at drawing the semantic information from XML Schema by interpreting it as RDF Schema and using available RDF/RDFS vocabularies. Therefore the mapping results still remains structure of XML document and provides more semantics for XML elements.

Another approach is presented in the C-Web project [5]. This method uses XPath to map information in XML documents to domain specific ontologies. This proposal exploits more specific meaning and structure of the XML documents. However, it requires human intervention to define

the meaning for every element in the DTD. Moreover, beside reference to XML document and its DTD, it requires referring to another resource, the specification of rules, which is not a requirement in our approach.

In another paper, Michel Klein introduces a procedure to transform XML data into RDF data by annotating the XML documents via external RDF Schema specifications [6]. This approach is close to our method. However, it does not transform all XML elements. Instead it concentrates on translating some pieces of information in the XML document. Moreover, elements in XML document are decided to be classes or properties depending upon user's opinion. Therefore, the results of this approach could be different among users' point of view.

In [7], we have proposed a procedure for transforming valid XML documents into RDF via RDF Schema. This procedure also derives classes and properties from XSD, then matches them with elements in XML documents and interprets all XML data as RDF statements. However, in order to describe the relationship between parent class and child class, we defined new RDF vocabulary, *rdfx:contain*. This definition is not recognized by the RDF evaluation tools or Semantic Web applications. Therefore, in this paper, we use existing RDF vocabularies by using *rdfs:Container*. Moreover, the result of this paper is displayed in graph and evaluated by tool recommended by the W3C.

The authors of [8] propose mappings from XML to RDF and from XML Schema to OWL ontology. However, the generated results from XML Schema may not suit to OWL model. Furthermore, the mapping from XML to RDF just concentrates on how to translate all XML elements into RDF and does not focus on meaning of elements. Therefore, this drawback is the same to [4].

Besides, there are several other approaches creating new OWL ontology for XML Schema [9, 10]. They produce a new ontology from an XML Schema and interpret instances of the XML Schema as instances of the generated ontology. The authors in [11] propose a mapping notation for every XML Schema and transform XML documents into existing OWL. This approach provides 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 Semantic Web.

This paper proposes a strategy to map XML Schema to the ontology (with considering the proper functions of classes and properties) and automatically interpret valid XML data (of that XML Schema) as RDF statements. 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.

III. MAPPING FROM XSD TO RDF SCHEMA

The goal of our approach is to provide a set of mapping rules that allows any XML Schema (XSD) to be converted to an appropriate RDF Schema. Then, the XML instances are transformed into a valid RDF document. Our defined rules ensure the structure of the original XML document, and provide domain and resource for each XML element.

The role of XML Schema is to provide syntaxes and structures for XML documents which differentiates from that of RDF, to model the semantic relationships of a domain. However, there is an overlap between them. Both have an object oriented foundation and their purpose is to define a general vocabularies and structures for exchanging information on the web. 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 general idea of this step is as follows:

- The <*schema*> element is the root element of the XML Schema. It can contain some attributes. The attribute *xmlns*: is interpreted as namespace.
- The first element is declared by element name is the root-class of document.
- For each XML Schema *xs:complexType* which is described using *sequence*, *all* and *choic*e, we map to a *rdf:class*. Every element or attribute declared within it is mapped to a contained class or sub-property.
- For each sub-element (elements in brackets or following the first element), we decide whether they are subclasses or properties of the class.
- For elements of *xs:simpleType*, it is mapped to a *rdf:Property*.
- Global element and attribute definitions are mapped similarly to the local ones.

By observing the XSD and its corresponding XML instances, we recognized that only XSD definitions, such as *xs:element* and *xs:attribute*, appear in the XML document. It means that XML instances contain two main components, that are elements and attributes. Therefore, our goal concentrates on transforming XML constructs that are related to these components.

We consider five main definitions in an XML Schema: element name, element ref, attribute name, attribute ref, and datatype.

An element definition has two following syntaxes: <*xs:element name =”xxx”*> or <*xs:element ref =”xxx”*>, where *xxx* is the given name of the element. The prefix *xs* can be changed depending on the namespace declaration.

Because <*xs:element name*> is used to describe elements of a document and each element can contain children elements [12], the function of these elements is like a class in a structure program, therefore, we treat element-name as a name of the class in our procedure. However, two cases are considered for <*xs:element name*> definition. If <*xs:element name*> contains another <*xs:element name*>, which has not only literal, we can assume that is a “part-of” relationship. Our procedure considers it as a class which is a subclass of the previous class (element-name). Contrary, if an element of the source tree with declaration <*xs:element name*> is always a leaf (containing only a literal and no attributes), this element is mapped to a property (despite it is defined by <*xs:element name*>).

Moreover, there are two kinds of class definition in XML Schema, *<xs:element name>* and *<xs:element ref>*. The second one refers to the name of another element. This reference to an element or an attribute is comparable to cloning an object. Therefore, our procedure considers *<xs:element ref>* as *<xs:element name>* and both are applied the same mapping rules. If an element has element ID, our procedure considers it as unique identifier for this element. Hence, if there are other elements with the same name and same level, we use this ID instead of creating new attribute for class element.

Furthermore, because attribute provides extra information about elements, its function is to describe a property of the class, we consider it as a property of the class. Therefore, for attribute definition with the declarations as *<xs:attribute name ="xxx">* or *<xs:attribute ref ="xxx">*, the second one refers to another attribute (similar to *<xs:element ref>*), our procedure maps it to a property of a corresponding class.

For data type definitions, such as type= "xs:string/date/…", they are mapped to *rdfs:datatype*. This is used to connect a data type with its property. Moreover, in order to link between this property and its value, we use *rdfs:literal*. For connecting value of a class, *rdf:value* is added.

On the other hand, due to limited expressions in RDF and the inappropriate mapping between RDF and XML Schema, we have to skip some unnecessary definitions in XML Schema, such as definitions for element attributes as substitutionGroup, default, fixed, form, maxOccurs, minOccurss, abstract, block, final, identity-constraints, and the complexType declared within the simpeType.

Creating RDF Schema includes two main steps:

a) Class description: containing *rdfs:comment* (class name + "class") – human readable description of the resource- and rdfs:Container (describing the resource is a subclass of a class).

b) Property description: holding *rdfs:domain* – indicates the class which this property is described for – and *rdfs:range* – indicates a class which values of the property must be members or a data type.

For instance, an XML Schema of the following link http://www.onjava.com/pub/a/onjava/2004/09/15/schema-validation.html is look like 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="journal" 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="journal">
 <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" type="xs:string" />
  </xs:schema>
```

In the above XSD document, root class is *catalog*, that contains information of *journal* element. Element *journal* contains three properties, *title*, *publisher* and *date*, and a class *article*. A class *article* includes the *title* and *author* elements. Although *title* and *author* are defined by *xs:element name*, but they do not contain any other elements, we consider them as attributes.

Since there are two elements that have the same name, *title*, the second repeated name is renamed by adding its parent name in front of its name.

By using all above notations we harvest RDF Schema as following:

```
<?xml version="1.0" ?>
  <rdf:RDF xmlns:rdf=
  "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs=
  "http://www.w3.org/2000/01/rdf-schema#">
  <rdfs:Class rdf:ID="Catalog">
    <rdfs:comment>catalog Class</rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="journal">
    <rdfs:comment>journal Class</rdfs:comment>
    <rdfs:Container rdf:resource="#catalog" />
  </rdfs:Class>
  <rdfs:Class rdf:ID="article">
    <rdfs:comment>article Class</rdfs:comment>
    <rdfs:Container rdf:resource="#journal" />
  </rdfs:Class>
  <rdf:Property rdf:ID="title">
    <rdfs:domain rdf:resource="#journal" />
     <rdfs:range rdf:resource=
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal" />
 </rdf:Property>
  <rdf:Property rdf:ID="publisher">
    <rdfs:domain rdf:resource="#journal" />
     <rdfs:range rdf:resource=
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal" />
    </rdf:Property>
  <rdf:Property rdf:ID="date">
    <rdfs:domain rdf:resource="#journal" />
     <rdfs:range rdf:resource=
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal" />
   </rdf:Property>
  <rdf:Property rdf:ID="article_title">
    <rdfs:domain rdf:resource="#article" />
     <rdfs:range rdf:resource=
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal" />
   </rdf:Property>
  <rdf:Property rdf:ID="author">
    <rdfs:domain rdf:resource="#article" />
     <rdfs:range rdf:resource=
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal" />
    </rdf:Property>
    </rdf:RDF>
```

Clearly, in the above RDF Schema document, there are three classes, *catalog*, *journal* and *article*. Each class is added a description by using *rdfs:comment.* The nested class is described by using *rdfs:Container*. The attribute *title* of the class *article* is changed to *article_title.* Each property is supported by *rdfs:domain* and *rdfs:range* which restrict the anterior and posterior values of a property. Since all of attributes in the given example do not contain any child element, the *rdfs:subPropertyOf* is not used.

IV. XML TRANSFORMING

A. Algorithm

After deriving RDF Schema from an XML Schema, we continue to examine the valid XML document. The result is RDF triples to interpret these XML data based on the generated ontology. The URI of the XML document will be the URI of each class. The algorithm starts traversing from the beginning of the XML document and finishes when it meets the close tag of root element. The comments are skipped during the transformation process.

Since during the mapping step, we have changed some names of the XSD elements, in this transformation step, we must update the changed element in the XML instances too.

The algorithm can be express by pseudo-code as below:

```
Read the description of root-class in the XML document to
draw its properties if they are available.
For 1 to total number of child-node (of the XML document)
 Begin
 Create the namespace for XML document;
 For each complex element (class)
  Begin
   Generate an RDF description for each class;
   Create a resource for class  (URI=baseURI +
                        resourceName+#class+number)
  //For nested class, resource name is the path specifying this
class
  //For root class, there is no number, only class name
    For each attribute in the class
      begin
        Create tag "namespace:attribute name";
        Copy attribute values;
      end;

   Else (being a class)
   Repeated as child-node
  End;
 End;
```

On the contrast to XML instances, which allow same element names to be appeared within the document, valid RDF requires that each class has a unique resource. Therefore, when we transform a class element, we assign a resource for each class. Our approach defines the resource by concatenating the URI of the XML document with the class name, following with the number of appeared times.

In the case that property has more than one value, these values can be stored by RDF container (*rdf:Bag*). For instance, if there are three authors for one article, the RDF statements are as following:

```
<rdf:Bag>
 <rdf:li>Author name 1</rdf:li>
 <rdf:li>Author name 2</rdf:li>
 <rdf:li>Author name 3</rdf:li>
<rdf:/Bag>
```

B. Example

In this section, we illustrate the transforming from XML document into RDF. The XML document is also taken on the same website with XML Schema.

Here is XML document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- A OnJava Journal Catalog  -->
<catalog xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=
                    "file://c:/Schemas/catalog.xsd"
 title="OnJava.com" publisher="O'Reilly">
<journal date="April 2004">
<article>
  <title>Declarative Programming in Java</title>
  <author>Narayanan Jayaratchagan</author>
  </article>
  </journal>
<journal date="January 2004">
<article>
  <title>Data Binding with XMLBeans</title>
  <author>Daniel Steinberg</author>
 </article>
 </journal>
</catalog>
```

Our algorithm traverses from the beginning of the XML document and finish until meeting close tag of the root class. When it meets an element, it will compare this element to definition in the RDF Schema to decide whether it is a class or a property. If it is a property, it will drag this value and tag from XML document to RDF. Otherwise (class), it creates *rdf:Description* and describes new resource for that class.

In this document, there are two "journal" nodes, each node includes another "article" node. Therefore, our procedure creates four resources for each node.

Because the value of property in this XML document is always single value, we do not use the RDF container element in the result.

Our corresponding RDF data for above XML document is as following:

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ca="http://www.recshop.fake/Catalog#">
<rdf:Description rdf:about="http://www.recshop.fake/ca/Catalog">
  <ca:title>OnJava.com</ca:title>
  <ca:publisher>O'Reilly</ca:publisher>
  <ca:journal>
<rdf:Description
rdf:about="http://www.recshop.fake/ca/Catalog/journal1">
  <ca:date>April 2004</ca:date>
  <ca:article>
<rdf:Description
rdf:about="http://www.recshop.fake/ca/Catalog/journal1/article1">
```

```
  <ca:article_title>Declarative Programming in Java
   </ca:article_title>
  <ca:author>Narayanan Jayarachagan</ca:author>
</rdf:Description>
  </ca:article>
</rdf:Description>
  </ca:journal>
 <ca:journal>
<rdf:Description
rdf:about="http://www.recshop.fake/ca/Catalog/journal2">
  <ca:date>January 2004</ca:date>
  <ca:article>
<rdf:Description
rdf:about="http://www.recshop.fake/ca/Catalog/journal1/article2">
  <ca:article_title>Data Binding with XMLBeans
   </ca:article_title>
  <ca:author>Daniel Steinberg</ca:author>
</rdf:Description>
</ca:article>
</rdf:Description>
</ca:journal>
</rdf:Description>
</rdf:RDF>
```

xmlns:ca is a namespace, it specifies that elements with the prefix *ca* are from the namespace http://www.recshop.fake/Catalog#. Every class is described within element *rdf:Description* that contains the information of the resource identified by the *rdf:about* attribute.

Our procedure automatically generates RDF data, it does not requires any human intervention so the result is independent from every user. The RDF result obeys RDF syntaxes, so it does not requires any change in order to be used by the Semantic Web. This procedure also can be applied for scalable XML documents which exist enormously on the current web.

C. Discussion, implementation and evaluation

In this section, we discuss the reason why we choose the RDF for the destination transforming. Of course, other onlotogy languages than RDF can be used to described the meaning of the XML, too. However, we target on the RDF Schema and RDF instance since it is currently the foundation ontology language for the Semantic Web. Moreover, currently there are some tools supporting for it are available, such as Protégé, Altova, and some other reasoning tools.

Our approach is notably different from other related approaches. First, we translate between the schemas and update the changing element during mapping step in the original XML document. While mapping, we use the existing RDF and RDF Schema vocabularies, especially to express the relationship among nesting classes. Second, we transform the XML instances with namespace supports. Since our approach is based on the XSD definitions and exploits the RDF syntaxes, our transformation process is done automatically without any user intervention. Moreover, our result is a valid RDF document which is very important for applying directly on the web.

The program transforms valid XML document into RDF data including two files *.rdfs and *.rdf. File .rdfs stores descriptions of classes and the relationships between properties and classes as well as the data-types of these properties. The transformation does not depend on XML data, it can be used to transform arbitrary XML documents. The main function of our program is to interpret XML data as RDF data by travelling from the beginning of the XML document until matching close tag of root-class.

The program language to be used is C# with the help from the library .Net 2.0. We choose C# because it is a strong language supported for building application related to data processing and windows interface.

In order to validate our RDF output result, we use the RDF Validation Service of W3C at http://www.w3.org/RDF/Validator/. This RDF validation service is based on Another RDF Parser (ARP). It currently uses version 2-alpha-1. ARP was created and is maintained by Jeremy Carroll at HP-Labs in Bristol. This W3C service was created by Nokia's Art Barstow (a former W3C Team member). The service now supports the Last Call Working Draft specifications issued by the RDF Core Working Group, including datatypes. It no longer supports deprecated elements and attributes of the standard RDF Model and Syntax Specification and will issue warnings or errors when encountering them. In order to use this service, we only need to copy and paste the RDF file to the input window. When parsing large RDF files, requesting Triples Only instead of Triples and Graph will significantly shorten the response time of this service.

In the main interface of RDF validation, there are three display formats: Triples only, Triples and Graph, and Graph only. Here we choose ‘Triples and Graph” in order to see Subject-Predicate-Object structure and the visualization of RDF data.

For testing our RDF result, we paste our RDF statements to this validator, the result is verified successfully. By pressing the button “Parse RDF”, we can see the validation result as in the figure 1. This means that our RDF document can be used directly by other RDF editors or Semantic Web applications.

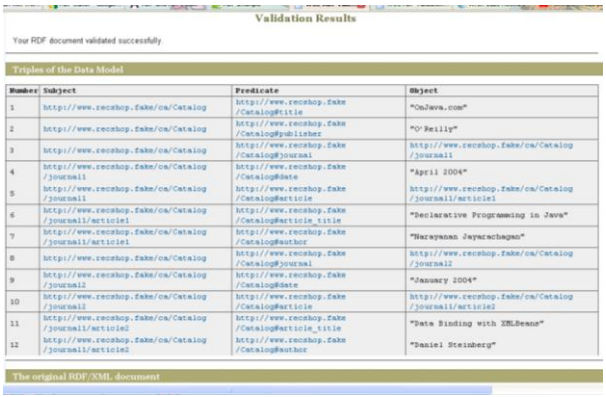

Figure 1. The validation result of our RDF document.

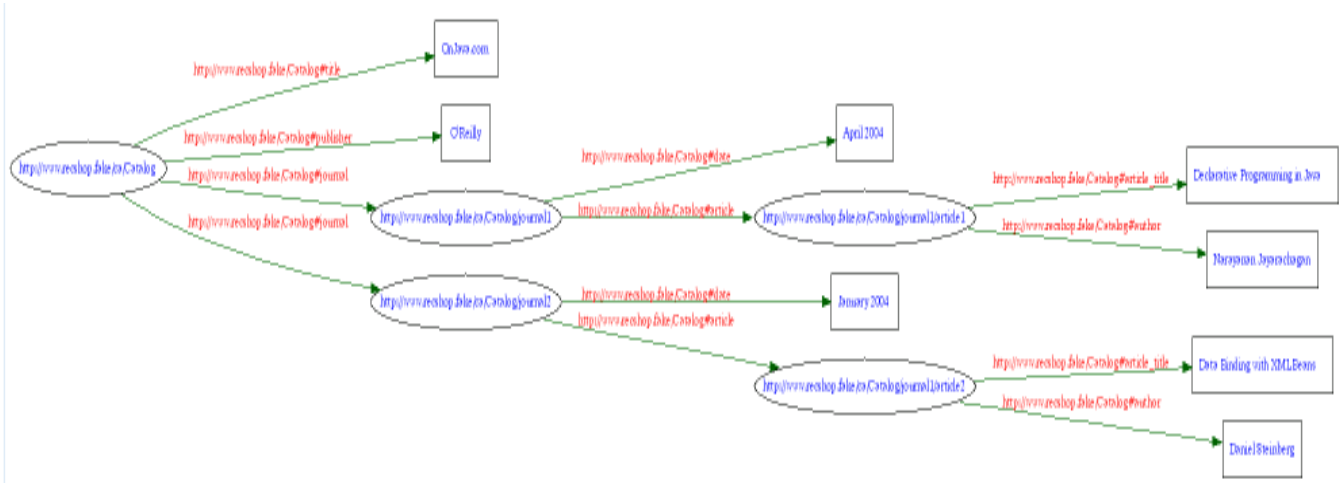

Figure 2. Graph of the RDF data

V. CONCLUSION

In this paper we have proposed a set of mapping rules to generate the ontology from the XML Schema and a procedure to transform valid XML documents into RDF statements by using the existing RDF vocabularies. These are crucial for referencing and integrating XML data into the Semantic Web.

The generated RDF statements are much more semantics than the original XML document. In general, if XML Schema is available, our procedure effectively performs all mapping and transformation automatically without any human intervention.

Our transformation is fundamental and can be applied to any XML Schema and XML instances. Therefore it can be consider as a standard for automatic transformation from XML data into RDF. The transformation is generic so that the inverse transformation can be built by defining the converted mapping rules. Moreover, the validation result shows that our RDF statements are successfully satisfied the regulations of the W3C. This means that our RDF file can be used directly by Semantic Web applications without any changes.

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 Semantic Web.

ACKNOWLEDGMENT

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2010-(C1090-1021-0003))

REFERENCES

[1] 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.

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

[3] Graham Klyne, Jeremy J. Caroll, and Brian McBride, “Resource Description Framework (RDF): Concepts and Abstract Syntax”, W3C Recommendation, 2004, available at: http://www.w3.org/TR/rdf-concepts/

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

[5] B.Amann, I.Fundulaki, M.Scholl, C.Beeri, and A-M.Vercoustre, “Mapping XML fragments to community Web ontologies”, Fourth International Workshop on the Web and Databases (WebDDB’2001).

[6] Michel Klein, “Interpreting XML via an RDF Schema”, 2002, Database and Expert Systems Applications.

[7] Pham Thi Thu Thuy, Young-Koo Lee, Sungyoung Lee, and Byeong-Soo Jeong, “Exploiting XML Schema for Interpreting XML Documents as RDF”, 2008 International Conference on Services Computing.

[8] Matthias Ferdinand, Christian Zirpins, and David Trastour, “Lifting XML Schema to OWL”, 2004, Web Engineering – 4th Int.Conference, ICWE, pp. 354–358.

[9] Roberto García, Ferran Perdrix, and Rosa Gil, “Ontological Infrastructure for a Semantic Newspaper”, 2006, Semantic Web Annotations for Multimedia Workshop, SWAMM’06, UK.

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

[11] Toni Rodrigues, Pedro Rosa, and Jorge Cardoso, “Mapping XML to Existing OWL Ontologies”, 2006, International Conference WWW/Internet.

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