Sri Lanka Association for Artificial Intelligence (SLAAI)
Proceeding of the sixth Annual Sessions
30th October 2009 – Colombo

Ontology Based XML Schema Generation

Kanishka Jayawardene kanishka_kan@yahoo.com
Kapila Ponnamperuma kapila@kln.ac.lk
Shantha Jayalal shantha@kln.ac.lk

Department of Industrial Management
University of Kelaniya, Sri Lanka

Abstract :There are many commercial and non commercial tools available for XML Schema generation for XML instance documents. However most of the tools produce the schema by guessing the metadata from the instance documents. Therefore lots of manual works are needed to fit the generated schema to match with the real requirements. This paper presents a different approach to generate XML schema using domain ontology with very high accuracy and completeness.

Keywords: XML, Ontology, XML Schema.

1. Introduction

EXtendible Markup Language (XML) is a simple and flexible language to represent data and was found in 1996 [3]. XML inherits the qualities of both HTML and SGML (Standard Generalized Markup Language). World Wide Web Consortium (W3C) ratified the XML in 1998 [7]. Since then XML has been evolved and been used in many areas such as data storage, messaging, interfacing, etc. Even though XML is a simple language, now it has a huge impact on the growth of the Web. XML is the basis for several next generation web technologies. Such as XHTML, RSS (Blogs), AJAX, Web Services [6].

This has lead to the creation of a huge amount of both on and off line XML documents for many purposes such as exchanging data or messages between different applications. These applications also require corresponding XML schema definitions with each of these XML instance documents for the purpose of validation [4]. Generating these XML schemas for each XML instance is a laborious task which needs a lot of human expert time [6].

2. Related Works

There are a number of tools available to create XML Schemas from XML documents. Tools such as Visual Studio XMLSpy [12], Stylus Studio [11] etc. are capable of generating XML Schemas automatically. Though different tools are able to generate XML schemas automatically, still those tools lack the capability of producing schema with high accuracy and with matching requirements [4]. It will be correct for the document but there might be instances where the document can vary while still remaining valid. And almost all available tools often generate a best-guessed schema from any XML document.

These tools make certain assumptions about the structure of the xml document, based on the data found in the XML document. For example, it will always set `minOccurs` to `1` and `maxOccurs` to `unbounded` for each element. It will also always use the `xs:sequence` compositor for lists of elements, even if the XML document has elements in various orders. For these reasons, it is not possible to take the schema generation granted. It has to be edited always to make sure it will fit in to the real requirements.

Method used in [4] uses a domain model by means of database tables. However, the main problem with the database approach is that though it works fine for a single enterprise, database structure may have to be changed for a different enterprise.

3. Objectives of the Research

The purpose of this research is to develop an automatic schema generator for XML documents using domain ontology. The ontology is used in order to provide the domain knowledge such as terminology, restrictions and data types. The XML schema generator can be used in any domains provided that the ontology is available for that domain. This will have a great impact on applications such as communication systems where XML is widely used. The manual work will be reduced in business organizations. Once the ontology is created for an organization, it is possible to refer the terms used in the ontology in XML instance documents and get the schema generated automatically.

4. Methodology

Most existing tools are focused on predicting the accurate XML schema depending on the data within the XML file. Though the XML schemas can be predicted, it is not the most accurate schema. The reason for this is mainly due to the absence of the domain knowledge in which these XML documents are created.

We focus our research to overcome these problems by proposing a general ontology [9] as the domain model. Given the ontology, XML schema can be generated for any XML document created within the assumed domain model.

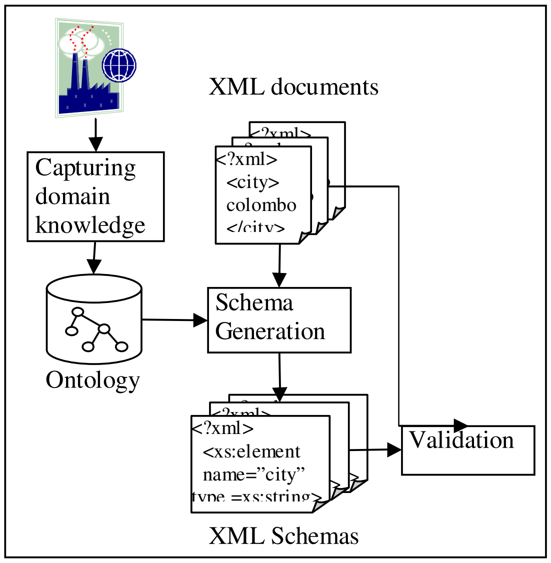


Figure 1: The conceptual model

Capturing Domain Knowledge

In order to create XML schema automatically for every XML document, it is necessary to store the domain knowledge related to the specified XML file in some machine-readable form. Ontology was selected as the means of capturing and storing the domain specific knowledge due to the flexible nature and the reasoning ability. Particularly RDF was chosen to develop the ontology as this had been used successfully in many web based applications. As this research mainly focuses on capturing terminology and related semantics used in a particular domain, methodology used in [8] has been used to define the concepts and their relationships. As the first step, the ontology should be developed, as it contains the terms used in the particular domain and its types and relationships.

Schema Generation

After defining a framework for the abstraction of the ontology, the next step was the algorithm development. The objective of the algorithm was to process any valid XML document and to produce the XML schema by obtaining necessary knowledge from the given ontology. *Appendix A* shows the abstract algorithm used in the implementation. If the XML document does not comply with the ontology, XML schema will not be produced as we use the closed world approach for ontology reasoning.

5. Implementation and Results

Protégé [2] ontology development tool was selected to develop the ontologies used in our prototype development due to its wide acceptance and the strong community support. Figure 2 shows the class hierarchy for the sample University ontology created using Protégé.

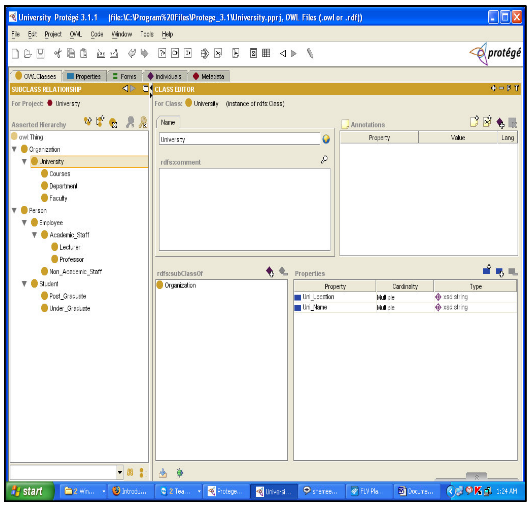
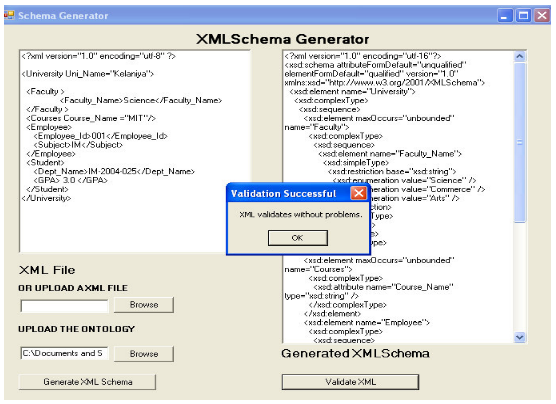


Figure 2: Class hierarchy of the university Ontology

XML Schema Generator

XML Schema Generator tool has been implemented using C#.Net. A screen shot of the tool is given in figure 3. In order to read the RDF triples written by Protégé, it was necessary to use a third party library since C#.Net does not have a precompiled library. Therefore, SemWeb [1] which is a .NET library designed for working with RDF is used. It provides methods for reading, writing, manipulating, and querying RDF.



```
<?xml version="1.0" encoding="utf-8" ?>

<University Uni_Name="Kelaniya">

 <Faculty >
          <Faculty_Name>Science</Faculty_Name>
 </Faculty >
 <Courses Course_Name ="MIT"/>
 <Employee>
   <Employee_Id>001</Employee_Id>
   <Subject>IM</Subject>
 </Employee>
 <Student>
   <Dept_Name>IM-2004-025</Dept_Name>
   <GPA> 3.0 </GPA>
 </Student>
</University>
```

C:\Documents and S

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema attributeFormDefault="unqualified"
elementFormDefault="qualified" version="1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="University">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded"
name="Faculty">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Faculty_Name">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="Science" />
                    eration value="Commerce" />
                    eration value="Arts" />
                  ction>
                  Type>
                  >
                  e>
                  pe>
        <xsd:element maxOccurs="unbounded"
name="Courses">
          <xsd:complexType>
            <xsd:attribute name="Course_Name"
type="xsd:string" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Employee">
          <xsd:complexType>
            <xsd:sequence>
```

XML validates without problems.

Figure 3: Screen shot of XML Schema generator

XML Schema generator is capable of generating the XML schema for any given XML document. The Ontology should also be provided in order to acquire the domain knowledge. For the simplicity in this research paper, a sample ontology of a University is mentioned. Appendices *B* and *C* respectively show a sample XML document and the corresponding schema generated using the XML Schema Generator tool.

6. Evaluation

Evaluation of our model has been done in two ways. First the generated XML schema and the XML instance documents were validated using standard XML schema validation schemes. Initially, a syntax check is performed while uploading the document. If the XML file is not syntactically correct, an error is given saying the XML is not well formed. After generating the Schema for a well formed XML file, the XML document is validated against the generated schema. Secondly, human experts also used to check whether the domain requirements such as data types, cardinality constraints, etc. have been captured correctly into the XML schema. During the validation, several ontologies (sample shown in *Appendix C*) have been used and the result has been checked with the existing tools in order to assess the accuracy of the generated schema.

In order to evaluate the accuracy of our model we used several XML instance documents (sample shown in *Appendix B*). Each XML document consists of five elements and four different XML documents were considered from each ontology for the evaluation. Generated data types and the number of occurrences from each tool is compared with the expert. *Appendix D* shows the comparison between the XML Schema generation tools.

The comparison shows that there is a 30% improvement on average for the data types and 48% improvement on average for the number of occurrences feature. Therefore, there is a significant improvement in accuracy.

The existing tools are not capable of generating restrictions. But, our model is capable of generating accurate restrictions and hence shows a 100% improvement in completeness when compared to the existing tools available.

7. Conclusion

The main advantage of generating XML schema automatically is that this method can be used in applications where many XML files are generated daily and compliance schema is required. Evaluation shows that our tool can be used to generate XML schemas accurately for a domain specific XML documents. The increased accuracy of XML schemas probably makes the communication more effective. It is possible to use this in many industries without being restricted by the distance and platform.

8. References

[1] Semantic Web/RDF Library for C#/.NET By Joshua Tauberer
http://razor.occams.info/code/semweb/

[2] Getting Started with Protégé-OWL (3.x series)
http://protege.stanford.edu/

[3] W3C XML Schema Definition Language
http://www.w3.org/XML/Schema

[4] Lisa Bahler, Francesco Caruso, Josephine Micallef, *Experience with a Model-Driven Approach for Enterprise-Wide Interface Specification and XML Schema Generation*, Proceedings of the 7th International Conference on Enterprise Distributed Object Computing, September 16-19, 2003

[5] W3C Namespaces in XML
http://www.w3.org/TR/1999/REC-xml-names-19990114/

[6] W3C Extensible Markup Language (XML)
http://www.w3.org/XML/

[7] Bray T, Sperberg-McQueen CM, Paoli J, & Yergeau F. *Extensible Markup Language (XML) 1.0.* http://www.w3.org/TR/REC-xml/ . 2007.

[8] Deen SM & Ponnamperuma K. *Dynamic Ontology Integration in a Multi-agent Environment.* 20th International Conference on Advanced Information Networking and Applications (AINA 2006)., 367-372. 2006. IEEE Computer Society.

[9] Noy NF & McGuinness DL 2001, *Ontology Development 101: A Guide to Creating Your First Ontology* Stanford Knowledge Systems Laboratory Technical Report KSL-01-05.

[10] POL. Protege Ontology Library. http://protege.cim3.net/file/pub/ontologies/travel/travel.owl . 1-5-2007.

[11] Stylus Studio 2009
http://www.stylusstudio.com/

[12] XMLSpy 2009
http://www.altova.com/xml-editor/

Appendix A

```
 Load the Required Ontology to the memory
 Extract the Root element from the XML file
   CreateComplexType (XmlElement)       {
         Create an Array List
         Check the Element for Child nodes
          For each Child Node {
              If the node is an Element                    {
          If no existing Node as the same name in the Array List
              {
              Add the new node to the Array List
              Sort the Array
              //Node contains Children or Attributes
Create a Complex type container
              If (Node has ChildNodes OR Attributes)      {
                   CreateComplexType (Node)
                   }
                   Else{
                        //No complex Type
                   Needed
                   Create a XMLSchema Element
with the Node name
                   }
              Check the Ontology for the maxOcuurance
              Set the Xml Schema Element
maxOccurance
              }
              }
         }
         If the element has Attributes {
              For each Attribute {
                   Create XML Schema Attribute
              }
         }
         If the Element is Complex Type {
              Add Complex Type Name
         }
              Else {
              Add a Simple Type
              Check the Ontology for Simple Type
Restriction
              Check the Ontology for Data Type
         }
           Return XML Schema Element
        }
 Create the XML Schema
```

Appendix B

```
<?xml version="1.0" encoding="utf-8" ?>
<University Uni_Name="Kelaniya">
  <Faculty >
<Faculty_Name>Science</Faculty_Name>
  </Faculty >
  <Courses Course_Name ="MIT"/>
  <Student>
    < Student_Name>Kanishka</Student_Name>
    <GPA> 3.0 </GPA>
  </Student>
</University>
```

Appendix C

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema          attributeFormDefault="unqualified"
elementFormDefault="qualified"              version="1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="University">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded"
name="Faculty">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Faculty_Name">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="Science" />
                    <xsd:enumeration value="Commerce" />
                    <xsd:enumeration value="Arts" />
                  </xsd:restriction>
                </xsd:simpleType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element maxOccurs="unbounded"
name="Courses">
          <xsd:complexType>
            <xsd:attribute name="Course_Name"
type="xsd:string" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Student">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Student_Name"
type="xsd:string" />
              <xsd:element minOccurs="0" name="GPA"
type="xsd:float" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="Uni_Name" type="xsd:string" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Appendix D

		No of occurrences				Improvement
Ontology		Visual Studio	XML Spy	Schema Gen	Expert	
University	Test case1	3	3	5	**5**	
	Test case2	3	3	5	**5**	
	Test case3	3	3	5	**5**	
	Test case4	2	2	5	**5**	
Match with the Expected		**11**	**11**	**25**		**56%**
Vehicle	Test case5	3	3	5	**5**	
	Test case6	4	4	5	**5**	
	Test case7	4	4	5	**5**	
	Test case8	4	4	5	**5**	
Match with the Expected		**15**	**15**	**25**		**40%**

		Data type				Improvement
Ontology		Visual Studio	XML Spy	SchemaGen	Expert	
University	Test case1	5	5	5	**5**	
	Test case2	4	4	5	**5**	
	Test case3	3	3	5	**5**	
	Test case4	3	3	5	**5**	
Match with the Expected		**20**	**20**	**25**		**20%**
Vehicle	Test case5	3	3	5	**5**	
	Test case6	3	3	5	**5**	
	Test case7	3	3	5	**5**	
	Test case8	3	3	5	**5**	
Match with the Expected		**15**	**15**	**25**		**40%**