DTD2OWL: Automatic Transforming XML Documents into OWL Ontology

Pham Thi Thu Thuy
Ubiquitous Computing Lab
KyungHee University
Yongin, Korea
+82-31-201-2493
tttpham@oslab.khu.ac.kr

Young-Koo Lee
Ubiquitous Computing Lab
KyungHee University
Yongin, Korea
+82-31-201-3732
yklee@khu.ac.kr

SungYoung Lee
Ubiquitous Computing Lab
KyungHee University
Yongin, Korea
+82-31-201-2514
sylee@oslab.khu.ac.kr

ABSTRACT
DTD and its instance have been considered the standard for data representation and information exchange format on the current web. However, when coming to the next generation of web, the Semantic Web, the drawbacks of XML and its schema are appeared. They mainly focus on the structure level and lack support for data representation. Meanwhile, some Semantic Web applications such as intelligent information services and semantic search engines require not only the syntactic format of the data, but also the semantic content. These requirements are supported by the Web Ontology Language (OWL), which is one of the recent W3C recommendation. But nowadays the amount of data presented in OWL is small in compare with XML data. Therefore, finding a way to utilize the available XML documents for the Semantic Web is a current challenge research. In this work we present an effective solution for transforming XML document into OWL domain knowledge. While keeping the original structure, our work also adds more semantics for the XML document. Moreover, whole of the transformation processes are done automatically without any outside intervention. Further, unlike previous approaches which focus on the schema level, we also extend our methodology for the data level by transforming specific XML instances into OWL individuals. The results in existing OWL syntaxes help them to be loaded immediately by the Semantic Web applications.

Categories and Subject Descriptors
D.2.12 [**Software Engineering**]: Interoperability – *Data mapping.* F.3.2 [**Logics and meanings of programs**]: Semantics of programming languages – *Process models, Program analysics.*

General Terms
Languages, Algorithms, Experimentation.

Keywords
OWL, DTD, XML, transformation, semantics.

ICIS 2009, November 24-26, 2009 Seoul, Korea

1. INTRODUCTION
In the recent years, XML (eXtensible Markup Language) has reached a wide acceptance as the relevant standardization for storing and exchanging data on the Web. When two participants agree on the XML data format, they begin to transfer and receive data from each others. To support for this trading, XML documents are often built based on their given schemas, which are expressed in DTD (Document Type Definition) or XML Schema. Actually, a DTD or XML Schema contains the knowledge of a structure, data type and relationship among elements in XML documents. In comparing to XML Schema, DTD is the earlier schema language for XML. It is more compact and higher readable than XML Schema [1]. Our method targets on DTD, utilizes its declarations to produce suitable mapping rules. Before deepening the mapping and transforming processes, let review the advantages as well as drawbacks of XML data and the reasons why it is necessary to convert XML data.

The first advantage of XML is its language representation. It uses human language (not computer), thus its language is readable and easy to understand. Second, its language is compatible with common programming languages such as Java, C++, therefore any application that can process XML can retrieve XML information. Third benefit is its flexibility. It allows anyone to describe any content easily by creating their own tags. However, this freedom can cause lack of understanding between a document's author and its consumer. Since an object can be expressed by different vocabularies, it is hard for computer to recognize and differentiate their meaning. For instance, "author" can be described as "creator", "inventor", etc. according to users' opinions. Furthermore, XML exposes disadvantages when coming to the semantic interoperability. XML mainly focuses on the grammar but there is no way to describe the semantics of a document [2]. Therefore, problem happens when software agents would like to understand and reason about these XML data.

To solve this problem, transforming XML documents into domain knowledge such as OWL is received high interest. Since a DTD document can be developed to a lot of XML documents, it is necessary to map DTD to OWL model to create a general structure for resulted ontology. Thus many approaches have been produced to solve schema mapping problem. However, most of them are replied on extending OWL syntaxes, even they produce a set of rules for each schema element. There are also few studies target on conversing XML Schema to existing OWL ontology. However, most approaches stop at schema mapping step, very few of them perform the transformation of XML instances into OWL individuals.

In this paper, we provide not only the DTD mapping process but also the XML document transformation. Compared to related approaches, our DTD mapping process is the new methodology supporting for conversing schema automatically. Moreover, we extends our procedure to transform XML instances into OWL individuals which clearly supports the solution for duplicated data. Our approach results in the existing OWL vocabularies, hence it is easily loaded by Semantic Web applications, and we can save a vast amount of memory in programming.

The next of the paper is organized as follows. In section 2, we overview some requirements of transformation and previous approaches for schema mapping and transforming XML data into OWL ontology, and then we analyze strengths as well as weaknesses of them. Section 3 describes our transformation framework, the mapping notations from DTD to OWL ontology, together with the detail descriptions of transformation XML documents into OWL file and illustrated examples. Section 4 gives experiment and discussion. Finally section 5 concludes and mentions about future directions of this research.

2. TRANSFORMATION REQUIREMENTS AND RELATED WORK

2.1 Transformation requirements

All XML transformations into OWL ontology should be satisfied the following demands:

1) **Data integrity:** The transformation output should adequately describe the original data.
2) **Semantics integrity:** In some circumstances, the transformation output (ontology) does not really convey the sense of information that described in an XML document. Therefore, the quality of the transformation should be analyzed.
3) **Maintaining original structures:** Besides the mapping of elements and attributes, the procedure should support the relationship mapping of these nodes.
4) **Data-type mappings:** An attribute should be mapped together with its data type.
5) **Use existing OWL syntaxes:** In some previous proposals, the ontology outputs are in strange or in extended ontology language so that they cannot be loaded directly by other Semantic Web applications or ontology editors. In this case, the transformation suffers from the applicability problem. Therefore, adding more semantics for original data by using existing destination ontology syntaxes is our target.
6) **Automatic transformation:** The process should be done automatically without any interference. Procedure satisfied this requirement can be used to transform arbitrary XML documents.

Our approach tries to meet these requirements during the mapping and the transforming data into OWL individual steps. By replying on the DTD descriptions, our method maintains the structure as well as the meaning of XML data. Moreover, we provide more semantics for XML instances via adding more definitions for elements and their relationships in OWL ontology by using existing OWL vocabularies. Therefore our results can be utilized immediately without any modification. On the other hand, with the automatic mechanism, our procedure can be applied to convert arbitrary XML data.

2.2 Related Work

Several approaches related to schema mapping and XML transforming have been proposed. The majority of these proposals concentrate on matching XML Schema or defining mapping rules from DTD but lack support for XML data transforming. In this section, we summarize these approaches as well as analyze their strength and weakness. Based on such examining, we propose a more comprehensive and efficient solution for DTD mapping and automatic XML transforming.

One of typical approaches which aims at OWL ontology is proposed by Ferdinand et al.[3]. These authors describe mappings from XML to RDF as well as from XML Schema to OWL, but the mapping results are independent of each other. The OWL instances may not suit the OWL model, because elements in XML documents are mapped to different OWL domain knowledge depending users' opinion. Moreover, this approach does not tackle the question how to create the OWL model, if no XML Schema is available.

Other complete approaches on transforming XML Schema to OWL ontology are [4, 6, 11, 13]. Their transformations are developed in XSLT by automatically mapping each definition in XML Schema to corresponding OWL domain knowledge. They have more advantages than [4], because if there is no XML Schema available, their procedures are still able to generate an OWL ontology. However, they also stop at describing the mapping notations from XML Schema to OWL ontology and do not execute the transforming from XML instances into OWL individuals. Therefore, they do not recognize the problems happened when transforming XML data, such as, the next element in XML documents is the same with the previous one, or the values of the next element are same with the current one.

Also focusing on OWL target, but authors of [5, 7, 10, 12] intend to define a set of mapping rules from each schema to OWL ontology. These methods add more semantics for existing XML Schema but these rules are too complicated. They are produced by authors. Therefore, this set of rules may be different to each other even though they describe for a same schema. Moreover, it is impractical to define a set of mapping rules for each schema available on the internet, especially when it has large size.

In our previous work [8, 9], we concentrate on transforming XML document into RDF based on mapping DTD and XML Schema to RDF Schema. However, we extract classes and subclasses from the given schema based on the definitions and our adjustments. Therefore, human intervention is required during the first mapping step. On the contrary, our current approach considers subclasses as object properties of their parent class and executes the mapping and transforming steps automatically.

Generally, existing schema mappings and XML transforming solutions still expose several limitations. Most of them try to narrow the gap between the XML Schema and OWL but do not solve the problems when same elements in an XML document appear. For these problems, it is essentially to propose a solution that solves more comprehensively and effectively schema mapping and XML transforming. In this paper, we design such a solution to map DTD to OWL domain knowledge and automatically transforming XML documents into existing OWL ontology which can be loaded immediately by OWL editors and other Semantic Web applications.

3. MAPPING AND XML TRANSFORMING

3.1 DTD2OWL Framework

The general architecture of DTD2OWL is shown in Figure 1. First consideration of this architecture is XML document. If it does not go with a schema, in this case is DTD schema, we generate a DTD corresponding to this XML document by using the available tool recommended by HiTSoftware on the internet[1].

When having DTD as an input, a mapping process executes the converting all of DTD components to OWL ontology which captures the semantics and maintains the structure, element's names and data types of DTD. Moreover, the OWL ontology enriches the DTD by adding definitions to describe the meaning and relationship between elements in DTD. During this stage, our mapping mechanism also checks and solves the problem whether the next element has the same name with the previous one, if it does, these elements are renamed.

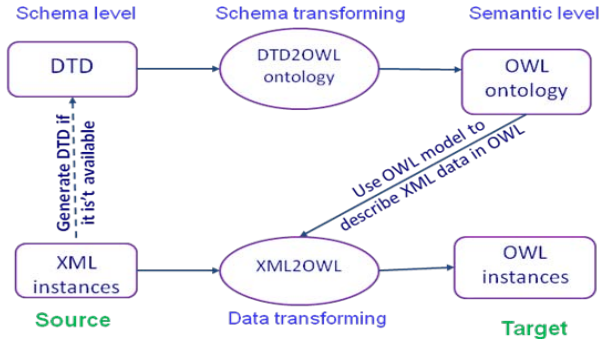

Fig. 1. XML Transforming framework into OWL data

Once these mappings are created, DTD2OWL system moves to the next step. The input of this step is an XML document together with an OWL ontology. Our procedure traverses from the root element in the XML document and ends when it meets a close-tag of the root element. If an element in XML data is matched with a node in OWL ontology, DTD2OWL system will execute the XML transformation instructions and generate an OWL document corresponding to XML data.

The details of mapping and transforming steps are presented in following sections.

3.2 Mapping DTD to OWL Ontology

In this section, we present a set of notations for the direct mapping of the DTD constructs to existing OWL concepts. Our mapping approach tends to convert every DTD nodes and attributes to a class or object property in the target ontology. The result of this mapping is an OWL ontology that maintains the structure and captures the semantics of the DTD constructs. The operation of DTD mapping is presented as following:

Root element. Element defined by <!DOCTYPE> in DTD is mapped to the root-class of OWL ontology, which is the first class declared by `owl:class`.

Classes (`owl:class`). For element definition *<!ELEMENT element-name (element-content)>*, if the *element-content* contains sequences of children, our procedure considers this element as OWL class, because it has common feature with OWL class in representing a set of individuals with the same chracteristics. Each `owl:class` is represented by a unique identifier, `rdf:ID`, and disjoints with other classes. As an example, consider the complex class in Fig.2, it is mapped to a class shown in Fig.3.

```
<!DOCTYPE catalog[
<!ELEMENT catalog ( product ) >
<!ELEMENT product ( catalog_item+ ) >
<!ELEMENT catalog_item ( item_number, price, size+ ) >
<!ELEMENT size ( color_swatch+ ) >
<!ELEMENT color_swatch ( #PCDATA ) >
<!ATTLIST color_swatch image ( black_cardigan.jpg |
burgundy_cardigan.jpg | navy_cardigan.jpg |
red_cardigan.jpg ) #REQUIRED > …
```

Fig. 2. Definition of complex classes in DTD

In Fig.2, the symbol "+" after "catalog_item" means that this element appears one or more times inside the class "product". It happens similar to the class "size" and "color_swatch".

```
<owl:Class rdf:ID="catalog">
    <owl:disjointWith>
      <owl:Class rdf:ID="product"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="color_swatch"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="size"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="catalog_item"/>
    </owl:disjointWith>
  </owl:Class>
```

Fig. 3. OWL declaration of the element definition in Fig.2

The definition `owl:Class rdf:ID="catalog"` means that class axiom "rdf:ID" declares the URI reference `#catalog` to be the name of an OWL class.

Object property. For nested elements, we do not use the `rdfs:subClassOf`, which is available in OWL syntaxes. The reason is because some nested elements in DTD are not actually the sub-class of their parent element. Furthermore, we would like to map all elements in DTD to OWL concepts automatically, so that we choose a middle course: We add a new object property described by `owl:ObjectProperty` to establish the relationship between child node and parent node. The object property's name is derived from "has" concatenating the underscore symbol "_" with the child's node name. The specification of parent node and child node are represented by `rdfs:domain` and `rdfs:range` respectively. By using this method, our transformation still express the nesting structure and can also provide more semantics for XML data.

For example, the relationships of three classes "`catalog`", "`product`", and "`catalog_item`" in Fig.2 are described as OWL concepts shown in Fig.4.

```
<owl:ObjectProperty rdf:ID="has_product">
    <rdfs:domain rdf:resource="#catalog"/>
    <rdfs:range rdf:resource="#product"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_catalog_item">
    <rdfs:domain rdf:resource="#product"/>
    <rdfs:range rdf:resource="#catalog_item"/>
</owl:ObjectProperty>
```

Fig. 4. OWL declaration of the element definition in Fig.2

[1] http://www.hitsw.com/xml_utilites/

In our procedure, nesting class is described by using owl:ObjectProperty. The class "catalog" containing class "product" is portrayed by adding new element "has_product". Similarly, "has_catalog_item" is used to define the parent-child relationship between "product" and "catalog_item".

Datatype property. For the case an element in DTD is described by <!ELEMENT> tag but it only contains datatype (#PCDATA or #CDATA), it is mapped to an OWL property, defined as `owl: DatatypeProperty`. The property's domain is the parent class of this property, and its range is the data type of this property. On the other hand, because #PCDATA and #CDATA declare for character data in XML, we map them to "`String`" data type in OWL. Note that, in the Fig.2, element "`color_swatch`" has a data type. Usually, it is recognized as a property, but since it contains child element (an attribute "`image`"), it is still recognized as a class. For instance, two simple elements in Fig.5 are mapped to OWL concepts in Fig.6.

```
<!ELEMENT item_number ( #PCDATA ) >
<!ELEMENT price ( #PCDATA ) >
```

Fig. 5. Definition of complex classes in DTD

```
<owl:DatatypeProperty rdf:ID="item_number">
    <rdfs:domain rdf:resource="#catalog_item"/>
    <rdfs:range        rdf:resource=
      "http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="price">
    <rdfs:range rdf:resource=
      "http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#catalog_item"/>
  </owl:DatatypeProperty>
```

Fig. 6. OWL declaration of the element definition in Fig.2

Moreover, DTD attributes normally contain other constraints. For instance, "#FIXED" means that the value of this attribute is stuck as defined. The declaration of "ENTITY" expresses that the name of this entity is already stored its value. "REQUIRED" means that the value must be appeared. On the contrary, value of an attribute with "IMPLIED" connotes that it is not demanded. And "NOTATION" means that attribute's value is a comment. Their mappings to OWL concepts are presented in table 1.

Table 1. The mapping of DTD constraints to OWL concepts

DTD	OWL
#FIXED value <!ENTITY entity-name "entity-value">	owl:hasValue
#REQUIRED	owl:minCardinality (=1)
#IMPLIED	owl:Cardinality (=0)
NOTATION	rdfs:comment
+	owl:minCardinality (=1)
?	owl:minCardinality (=0)
*	owl:minCardinality (=0) owl:maxCardinality (=unbounded)

There is another notice that XML syntax allows elements with the same name in a document, but OWL does not. OWL requires each element has a unique identifier. Therefore, when generating OWL individuals from XML instances, if the current XML element has the same name with the previous element, our procedure renames it by adding parent node's name concatenating with the symbol "_" before this element's name.

For example, there are two "description" attributes in Fig.7, one belongs to "product" class and another to "size" class. Because "description" of size is defined after that of "product", it is renamed to "size_description" as in Fig.8.

```
<!ATTLIST product description CDATA #REQUIRED >
<!ATTLIST size description CDATA #REQUIRED >
```

Fig. 7. Definition of complex classes in DTD

```
<owl:Class rdf:about="#size">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty
rdf:ID="size_description"/>
        </owl:onProperty>
        <owl:minCardinality rdf:datatype=
"http://www.w3.org/2001/XMLSchema#int">1
        </owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
```

Fig. 8. OWL declaration of the element definition in Fig.2

These changes are also updated in XML document for the nest step of transformation into OWL individuals.

3.3 XML Instances Transformation into OWL Individuals

The input of this step is the OWL model generated from previous step and the given XML instance. Our illustrated example is a kind of complex data represented in XML document[2]. "*ComplexData.xml*" contains the information about a *product* of a *catalog*. We choose this document because it has common characteristics of XML data on the Web.Only some of elements of XML document have unique identifiers. Moreover, there are many-to-many relationships in XML document. For instance, each *catalog_item* has different *sizes* and each *size* is specialized for many *catalog_item* elements.

This document describes a *product* of the *catalog*. *Product* contains many *catalog-item* elements, each of them includes information about clothe types. A part of that file is as below:

```
<?xml version="1.0"?>
<catalog>
   <product description="Cardigan Sweater"
                product_image="cardigan.jpg">
      <catalog_item gender="Men's">
         <item_number>QWZ5671</item_number>
         <price>39.95</price>
         <size description="Medium">
            <color_swatch
image="red_cardigan.jpg">Red</color_swatch>
            <color_swatch
image="burgundy_cardigan.jpg">Burgundy
            </color_swatch> </size>
         <size description="Large">
            <color_swatch
image="red_cardigan.jpg">Red</color_swatch>
            <color_swatch
image="burgundy_cardigan.jpg">Burgundy
            </color_swatch> </size>
      </catalog_item> …
</catalog>
```

Fig. 9. A first part of XML document

[2] http://www.service-architecture.com/object-oriented-databases/articles/xml_file_for_complex_data.html

The first importance of this transformation step is to specify the namespace for OWL document. In our approach, we use namespace "http://www.w3.org/2002/07/owl" for supporting OWL syntaxes and "http://www.w3.org/1999/02/22-rdf-syntax-ns" for providing RDF syntaxes. The next importance thing is to make sure that every source in OWL has unique identifier. Because XML document allows duplicate elements, when transforming these elements into OWL individuals, we need to give them the unique name to distinguish with others. Since our procedure starts traversing from the beginning of XML document, the first node matched with OWL ontology is added an ID. By default, the ID of the first instance node is created by adding the number 1 after the instance's name and the underscore symbol "_". Similarity, the next duplicate instance node is added the number 2, and so on. Furthermore, because this step is inherited from the previous step, the mapping step, some changes in the element names are kept. For instance, "*size_description*" replaces for "*description*" of element "*size*".

For example, the "*catalog*", "*product*", and "*catalog_item*" instance nodes in Fig.8 are added a unique identifier as in Fig.9. Because the "catalog_item" has two "size" instances, they are added the "size_1" and "size_2" identifiers respectively.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-s#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
…
<catalog rdf:ID="catalog_1">
    <has_product> <product rdf:ID="product_1">
        <description rdf:datatype=
              "http://www.w3.org/2001/XMLSchema#string">
                        Cardigan Sweater</description>
     <product_image rdf:datatype="http://www.w3.org/
          2001/XMLSchema#string"> cardigan.jpg

     </product_image> <has_catalog_item>
          <catalog_item rdf:ID="catalog_item_1">
            <gender rdf:datatype="http://www.w3.org/
              2001/XMLSchema#string">Men's</gender>
         <item_number
              rdf:datatype="http://www.w3.org/2001/
              XMLSchema#string">QWZ5671</item_number>
            <price rdf:datatype="http://www.w3.org/2001/
             XMLSchema#string">39.95</price>
            <has_size><size rdf:ID="size_1">
      <size_description rdf:datatype="http://www.w3.org/
       2001/XMLSchema#string">Medium</size_description>
        <has_color_swatch rdf:resource="#color_swatch_1"/>
        </size> </has_size><has_size>
        <size rdf:ID="size_2"> <has_color_swatch>
        <color_swatch rdf:ID="color_swatch_1">
        <contain_image rdf:datatype="http://www.w3.org/
       2001/XMLSchema#string">Burgundy</contain_image>
        <contain_image rdf:datatype="http://www.w3.org/
         2001/XMLSchema#string">Red</contain_image>
        <image rdf:datatype="http://www.w3.org/2001/
         XMLSchema#string">burgundy_cardigan.jpg</image>
        <image rdf:datatype="http://www.w3.org/2001/
         XMLSchema#string">red_cardigan.jpg</image>
        </color_swatch></has_color_swatch>
        <size_description rdf:datatype="http://www.w3.org/
         2001/XMLSchema#string">Large</size_description>
        </size> </has_size></catalog_item>
        </has_catalog_item></product></has_product>
  </catalog> …
</rdf:RDF>
```

Fig. 10. A first part of XML document

The unique ID also has a good advantage in inheritance. For example, the instance "*color_swatch*", which contains pictures of sweater, can be recalled by "*size*". In large XML documents, we can save a large amount of memory with this mechanism.

4. EXPERIMENTS AND DISCUSSIONS

4.1 Experiments

We carry out our experiment on AMD Turion 64x2, 2.00 GHz CPU 2GB memory with Visual C++ windows XP machine.

The whole DTD2OWL framework combines of two sub projects: mapping DTD to OWL ontology and transforming XML document into OWL instances. In the first mapping step, if our procedure finds the duplicate elements, it will rename that element (section 3.2). In this case, we also update the changes in XML document. Therefore, an intermediate procedure is also produced to modify these names in XML document.

We have implemented three procedures in XML stylesheet language transformations so that they can be easily integrated with other programming languages. These stylesheets are based on the XSLT stylesheets in [9], but we have extended and adjusted them to adapt to our framework. In [9], authors mainly extract XML Schema from given XML document which differs from our purposes, we derive DTD from XML document and use it to support the XML transformation. Moreover, we create a new procedure to update the name changes in XML file. Especially, our main stylesheet is totally different from [9], because our algorithm is different. We utilize the object oriented characteristics that each instance is given an ID expression so that it can be inherited from other instances.

4.2 Discussions

Our method almost satisfies the transformation requirements in section 2.1.

Fig. 11. A sample of DTD tree

In Fig.12, except for the added "*owl:thing*" element and changes element name "*description*" of "*size*", the OWL tree has the same structure with DTD tree. Since every OWL document usually contains "*owl:thing*" as the root class, our results only change the duplicate names. Hence our approach satisfies the 3rd requirement. On the other hand, our procedure provides more semantics for existing data by adding more vocabularies to describe the relationship among classes and between class and its properties so it meets the 5th rule. Further, with the detail descriptions in section 2.2 and 2.3, our approach obliges the 1st, 2nd and 4th requirements. Finally, because our procedure replies on the similar definitions between DTD and OWL to perform the mapping step, it does not need any user's interference to accomplish the whole transforming framework. Thus, the last requirement is also satisfied.

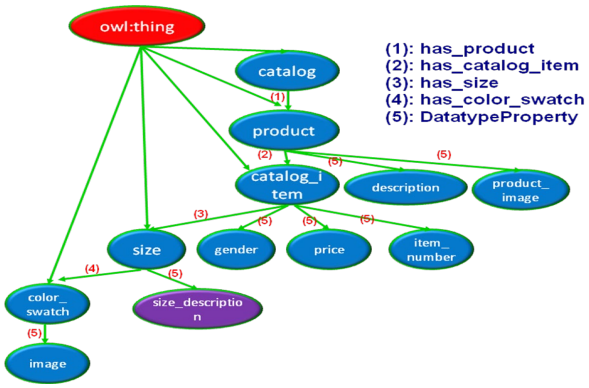

Fig. 12. The result OWL tree

In compare with other related works, our approach appears more advantages. First, we exploit DTD document of a given XML file, instead of XML Schema document in other works. Except for Bernd et al.[13], they also start at DTD file, but they create a mapping rule for each element in DTD. This method is said to provide more details for original data, but it is impractical because with a large amount of XML documents on the web, we cannot manually define the rules for each element. Therefore, to the best of our knowledge, our approach is a new method in mapping DTD to OWL ontology. Moreover, most of related work stops at the mapping process, except for Toni et al. [5]. However, Toni's method considers instances of XML document as a node, so to differentiate the duplicate data, it attaches the specified prefix to each datum. This method results in very large OWL instances and does not solve the repeated element names. For example, for the imitated "*description*" described in section 3.2, they have no solution. Our technique not only gives the answer for imitated element names but also reduces the consumed memory by inheriting the previous objects.

5. CONCLUSIONS

The DTD2OWL framework presented in this paper allows the automatic mapping DTD to OWL domain knowledge and transforming XML instances into OWL individuals. Our procedure outperforms the existing methods due to the following five reasons. Firstly, while transforming all the elements of an XML document into OWL, our algorithm retains the original structure and captures the implicit semantics expressed in the XML document. Secondly, components in DTD are considered as classes or properties or data types based on their definitions and detail descriptions, this makes the result be independent from users' opinions. Thirdly, languages used in our procedure do their jobs as their original functions. DTD is used for defining XML structure, XML for describing data, OWL for providing definitions and relationships between data. Fourthly, our approach provides new method for transforming XML instances into existing OWL individuals without any user intervention. That method makes many XML documents to be converted to the OWL formats. Finally, during the transformation process, our procedure not only solves the duplicate element problems but also provides the inheritance mechanisms which help reducing the consumed memory. We hope that our research has created a bridge to narrow the gap between the XML data and OWL ontology. If this procedure is executed, a large amount of the XML data on the current Web will be interpreted into OWL ontology which is useful for the Semantic Web applications.

Further improvement to our work may be focused on the transforming XML Schema into OWL ontology by giving more semantics than current approaches. Moreover, in order to prove the quality of our transformation, we are going to extend our work to the semantics measurement. The structure and semantic similarity of XML and OWL documents will be computed and compared to other methods.

6. ACKNOWLEDGMENTS

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement)" (IITA-2009-(C1090-0902-0002)). This work also, was supported by the Korea Science & Engineering Foundation(KOSEF) grant funded by the Korea government(MEST) (No. 2008-1342), and was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2009-0076798)

7. REFERENCES

[1] XML Schema Language Comparison, Wikipedia, 2009 http://en.wikipedia.org/wiki/XMLSchemaLanguage Comparison

[2] Hawke, S.: XML with Relational Semantics: Bridging the Gap to RDF and the Semantic Web. W3C, http://www.w3.org/2001/05/xmlrs (2001)

[3] M. Ferdinand, C. Zirpins, D. Trastour: Lifting XML Schema to OWL, 4th International Conference, ICWE 2004, pp. 354--358. Springer Heidelberg, Germany, 2004

[4] H. Bohring, S. Auer: Mapping XML to OWL Ontologies. LIT2005, pp. 147-156, Germany. 2005

[5] Toni Rodrigues, Pedro Rosa, Jorge Cardoso: Mapping XML to Existing OWL Ontologies. International Conference WWW/Internet 2006, pp. 72-77

[6] 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, pp. 137-146 (2007)

[7] C. Cruz, C. Nicolle: Ontology Enrichment and Automatic Population from XML Data, 4th Int. VLDB Workshop on Ontology-based Techniques, ODBIS 2008.

[8] P.T.T. Thuy, Young-Koo Lee, Sungyoung Lee, and Byeong-Soo Jeong, "Transforming Valid XML Documents into RDF via RDF Schema", International Conference on Next Generation Web Services Practices, IEEE, October 2007.

[9] P.T.T. Thuy, Young-Koo Lee, Sungyoung Lee and Byeong-Soo Jeong, "Exploiting XML Schema for Interpreting XML Documents as RDF", International Conference on Services Computing 2008, SCC'08, IEEE, Hawaii, July 2008.

[10] N. Kobeissy, M. G. Genet and D. Zeghlache, "Mapping XML to OWL for seamless information retrieval in context-aware environments", pp.349-354, IEEE, 2007.

[11] Chrisa Tsinaraki and Stavros Christodoulakis, "Interoperability of XML Schema Applications with OWL Domain Knowledge and Semantic Web Tools", pp. 850–869, OTM Conference, Springer-Verlag , 2007.

[12] P. Kunfermann, and C. Drumm, "Lifting XML Schemas to Ontologies – The concept finder algorithm", Mediate 2005.

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