Automatic Ontology Creation by Extracting Metadata from the Source code

Gopinath Ganapathy[1],S. Sagayaraj[2]

GJCST Classification
D.4.3,D.2.3,D.2.4

***Abstract*-Semantic Web can be created by developing Ontologies. For every new project Software companies are going for designing new code and components, by new developers. If the company archives the completed code and components, which can be used with no need for testing it unlike open source code and components. File metadata and file content metadata can be extracted from the Application files and folders using API's. The extracted components can be stored in the Hadoop Distributed File System along with the application environment. Extracted metadata will be in XML format. XML deals with syntactic level and the Web Ontology Language (OWL) supports semantic level for the representation of domain knowledge using classes, properties and instances. This paper converts the data model elements of XML to OWL Ontology that implements the mapping the standard XML technology *XSLT*.**

Keywords-Metadata, Parsing, Extensible Markup Language, Hadoop Distributed File System, Hbase, Web Ontology Language and Ontology.

I. INTRODUCTION

Most of the today's Web content is suitable for human consumption. An alternative approach is to represent Web content in a form that is more easily machine-processable by using intelligent techniques. The machine processable Web is called the Semantic Web. Semantic Web will not be a new global information highway parallel to the existing World Wide Web; instead it will gradually evolve out of the existing Web [1]. Ontologies are built in order to represent generic knowledge about a target world [2]. Ontology is more than a rule base of terminological problems and is worth to consider a promising methodology in the Semantic Web. In the semantic web, ontologies can be used to encode meaning into a web page, so that intelligent agents can understand what the web page is about. Ontologies increase the efficiency and consistency of describing resources, by enabling more sophisticated functionalities in development of knowledge management and information retrieval applications. Knowledge management concerns itself with acquiring, accessing, and maintaining knowledge within an organization. It has emerged as a key activity of large businesses because they view internal knowledge as an intellectual asset from which they can draw greater productivity, create new value and increase their competitiveness. From the knowledge management perspective, the current technology suffers in searching, extracting, maintaining and viewing information. The aim of the Semantic Web is to allow much more advanced knowledge management system.Software firms work on various projects for many years with little amount of archiving. If the projects are achieved properly the reusability of the available components will make the software development much easier and faster. This paper proposing an automatic generation of knowledge management by extracting metadata from the source code of the project files by using the Source code Documenter. The output will be converted into XML. Today XML has reached a wide acceptance as data exchange format. The aim of this paper is to bridge the gap from metadata, XML and OWL. A strategy for how OWL ontologies may be generated automatically out of existing XML data is proposed. This has to be done by establishing suitable mappings between the different data model elements of XML and OWL. All the project files, XML and OWL files are stored in Hbase repository. By storing code components the new project development can search for component in the OWL ontology and retrieve from the repository and using that will reduce the cost, manpower, time, testing, etc., in the software development. The paper begins with the discussion on the related technologies for this paper in Section 2. Then, detailed features and framework for the automatic metadata extraction is found in Section 3. The conversion from XML to OWL Ontology implements the mapping in the standard XML technology XSLT is presented in section 4. The implementation scenario is presented in Section 5. Section 6 deals with the findings of the paper.

II. RELATED WORK

1) *Metadata*

Metadata is defined as data about data" or descriptions of stored data. Metadata definition is about defining, creating, updating, transforming, and migrating all types of metadata that are relevant and important to a user's objectives. Some metadata can be seen easily by users, such as file dates and file sizes, while other metadata can be hidden. Metadata standards include not only those for modeling and exchanging metadata, but also the vocabulary and knowledge for ontology [3]. A lot of efforts have been made to standardize the metadata but all these efforts belong to some specific group or class. The Dublin Core Metadata Initiative (DCMI) [4] is perhaps the largest candidate in

About[1]- F.A. Author is Professor & Head with the Department of Computer Science, Bharathidasan University, Trichy, Tamil Nadu, India (e-mail: gganapathy@gmail.com)
About[2]- S.B Author is Associate Professor with the Department of Computer Science, Sacred Heart College, Tirupattur, Vellore, Tamil Nadu, India(Mobile:+91 9443035624; fax:+91 4179 220553; e-mail: sagi_sara@yahoo.com)

defining the Metadata. It is simple yet effective element set for describing a wide range of networked resources and comprises 15 elements. Dublin Core is more suitable for document-like objects. IEEE LOM [5], is a metadata standard for Learning Objects. It has approximately 100 fields to define any learning object. Medical Core Metadata (MCM) [6] is a Standard Metadata Scheme for Health Resources. MPEG-7 [7] multimedia description schemes provide metadata structures for describing and annotating multimedia content. Standard knowledge ontology is also needed to organize such types of metadata as content metadata and data usage metadata. These standards may be adopted in full or in part. Further, appropriate procedures need to be defined and followed within the enterprise in documenting the capture, update, transformation, migration, replication of metadata and relevant transformation rules.

2) *HDFS & HBASE*

The Hadoop project promotes the development of open source software and it supplies a framework for the development of highly scalable distributed computing applications [8]. Hadoop is designed in such a way to efficiently process large volumes of information. It connects many commodity computers together so that they could work in parallel. It is a simplified programming model which allows the user to write and test distributed systems quickly. In a Hadoop cluster even while, the data is being loaded in, it is distributed to all the nodes of the cluster. The Hadoop Distributed File System (HDFS) will break large data files into smaller parts which are managed by different nodes in the cluster. Each part is replicated across several machines, so that a single machine failure does not lead to non-availability of any data. Even though the file parts are replicated and distributed across several machines, they form a single namespace, so their contents are universally accessible. MapReduce [9] is a functional abstraction which provides an easy-to-understand model for designing scalable, distributed algorithms.HDFS. Hbase is the Hadoop application to use when real-time random accesses to very large datasets are required. It supports both batch-style computations using MapReduce and point queries. It is built from the ground-up to scale linearly just by adding nodes. Hbase is not relational and does not support SQL, but given the proper problem space it is able to do what an RDBMS cannot. The reason for using HDFS and Hbase in this paper is that the software size in a project is growing exponentially and also takes care of volumes data for processing. To handle data and code is stored in Hbase and it is distributed using HDFS. Also the size of a project and the number of project for a three decade company needs terabytes of volume to be stored which needs Hbase.

3) *Ontology*

The key component of the Semantic Web is the collections of information called ontologies. Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. Gruber defined ontology as a specification of a conceptualization [10].Ontology defines the basic terms and their relationships comprising the vocabulary of an application domain and the axioms for constraining the relationships among terms [11]. This definition explains what an ontology looks like [12].The most typical kind of ontology for the Web has taxonomy and a set of inference rules. The taxonomy defines classes of objects and relations among them. Classes, subclasses and relations among entities are a very powerful tool for Web use.A large number of relations among entities can be expressed by assigning properties to classes and allowing subclasses to inherit such properties. Inference rules in ontologies supply further power. Ontology may express rules on the classes and relations in such a way that a machine can deduce some conclusions. The computer does not truly ―understand" any of this information, but it can now manipulate the terms much more effectively in ways that are useful and meaningful to the human user. More advanced applications will use ontologies to relate the information on a page to the associated knowledge structures and inference rules.

III. EXTRACTION FRAMEWORK

After the completion of a project all the project files are sent to the metadata extraction frame work. The files of a project can be asked to give in form of zip file which contains various types of files. The framework will extract files and give it to corresponding Document Generator tools and it will convert the file metadata into XML file. This paper deals with java files only. To extract file content metadata the source code is processed by Javadoc. The retrieved components will be stored in the HBase, which is the subproject of Hadoop. The project may contain files in various languages and tools. Similarly there are many parser tools available to parse the source code from various languages and tools to either HTML or XML output. So the frame work will indentify the type of project file and supply the file to the corresponding parser. Two types of metadata will be extracted. First is the file metadata and it contains file attributes, file size, parent, path, etc. The second type of metadata contains the contents of the source file. The file content metadata will be different for different types of languages and tools. Here the frame work presents the general form for all type of project files. The frame work output shown is for the extracted metadata into XML file. The available Document Generator converting programs from one language or tool to HTML or XML file is presented in Table 1. But the frame work takes care of the languages and tools that support XML format. The framework needs only few tools which will be available in the HDFS to support to handle various types of files. After the converting the source files in to XML, files they are stored in HBase along with the Project files. The purpose of storing the XML files, source file and file attributes and file components in Hbase is to retrieve the code components and to reduce the software development cost. The components will be stored in Hbase by creating database design with the fields project name, project leader, period of project and code components. Several Sample Document Generator

strategies for extracting metadata from source file have been proposed. Document generator which takes care of XML alone is discussed and used in this framework.

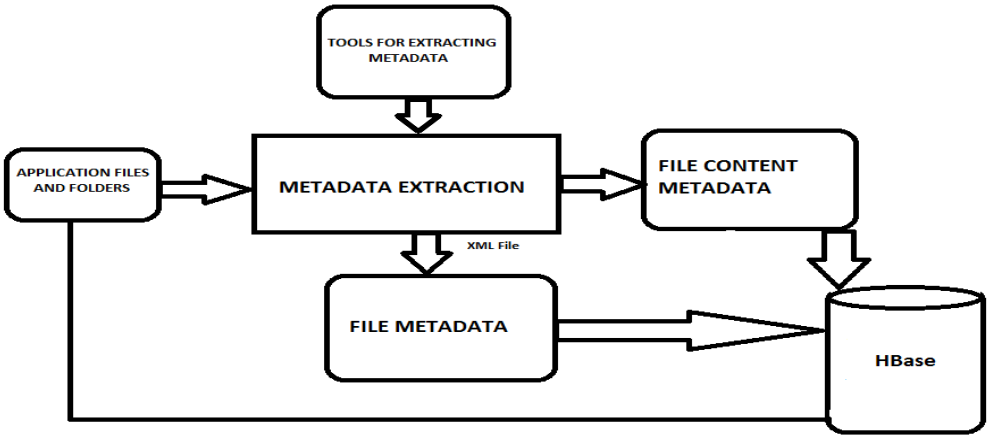

Fig.1. Metadata extraction Framework

1) *Doxygen*

Doxygen is a documentation for C++, C, Java, Objective-C, Fortran, VHDL, PHP and C# [13]. Doxygen supports the documentation tags used in the Qt toolkit and can generate output in HTML as well as in CHM, RTF, PDF, LaTeX, PostScript or man pages.

2) *Ddoc*

Ddoc is the embedded for the D programming language designed by Walter Bright. Its emphasis is on being able to write documentation in code comments in a natural style, minimizing the need for embedded markup and thus,improving the legibility of the code comments[14]. Code comments are associated with symbols in the code, and Ddoc uses the semantic and syntactic information available from the D compiler to fill in routine information such as parameters and return types automatically. Ddoc can generate output in XML and XHTML.

3) *FPDoc*

FPDoc is a tool that combines a Pascal unit file and a description file in XML format and produces reference documentation for the unit. The reference documentation contains documentation for all of the identifiers found in the unit‘s interface section[15]. FPDoc does not require the presence of formatted comments in the source code. It takes a source file and a documentation file (in XML format) and merges these two together to a full documentation of the source. This means that the code doesn‘t get obfuscated with large pieces of comment, making it hard to read and understand.

4) *JSDoc*

JSDoc is syntax for adding inline API documentation to JavaScript source code. The JSDoc syntax is similar to the

Javadoc syntax, used for documenting Java code, but is specialized to work with JavaScript's more dynamic syntax and therefore unique, as it is not completely compatible with Javadoc[16]. However, like Javadoc, JSDoc allows the programmer to create doclets and tags which can then be translated into published output, like HTML or RTF.

5) *PHPDocumentor*

PHPDocumentor is an open source documentation written in PHP. It automatically parses PHP source code and produces readable API and source code documentation in a variety of formats. PHPDocumentor generates documentation based on PHPDoc-formatted comments and the structure of the source code itself. PHP Documentor can create documentation in HTML, PDF, CHM or Docbook formats. PHPDocumentor is able to parse all PHP syntax and supports PHP4 and PHP5.

IV. CONVERTING XML TO OWL

The conversion process elevates the XML Schema Definition (XSD) to the level of OWL ontology. XML documents contain relational structure and represent those using OWL classes, properties and instances. The XML data model describes a node labeled tree [17], on the other hand OWL‘s data model is based on the subject-predicate-object triples from RDF[18]. RDF-Schema defines a vocabulary for creating class hierarchies, attaching properties to classes and adding instance data [19]. The framework creates a corresponding class hierarchy by exploiting the tree structure of XML. Restrictions like cardinality constraints and properties can be expressed easily when OWL is on the top of RDF and RDFS. This enables the representation of Source code data in OWL. The mapping is implemented in XML Stylesheet Language Transformations (XSLT) and it is interoperable with different programming languages [20]. As for as XML data is concerned no XML schema is attached. The conversion process generates a suitable intermediate XML schema definition for the XML. The overall architecture of the framework is shown in Fig. 2.

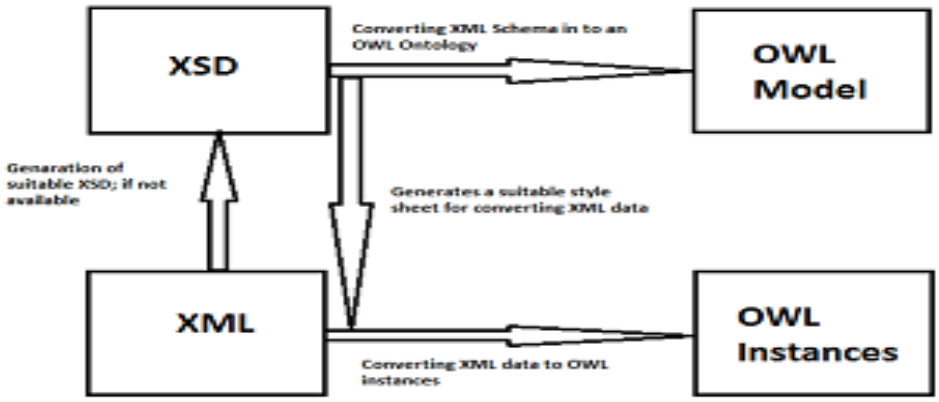

Figure.2. Conversion process from XML to OWL.

The conversion process requires at the most three steps for XML instance data and at least one step for XML Schema only. To process XML Schema only, the model of the ontology is creates along with classes and properties. Intermediate step is created for XML instance data only. The first step extracts an XML Schema out of the XML instance data. [21] states that in every XML instance document an XML Schema is exist implicitly. Since XML instance document do not contain as much information about constrains as in manually created XML Scheme, automatically generated XML Schema could be incomplete. There are also several XML Schema components and they cannot be discovered via stylesheet driven extraction. XML instance documents can contain optional elements or attributes, which are not in the sample document. Thus they are not in the generated XML Schema and OWL ontology. In XML Schema extraction a preferably representative XML instance document is needed, so that the XML Schema can serve as a good basis. This XML Schema extraction is based upon an XSLT stylesheet [22], which is extended and adapted to this framework. The majority of existing XML to OWL mapping tools use XSLT to accomplish the mapping [23].A stylesheet is created simultaneously while converting XML to OWL and this stylesheet will converts XML instance data into the instances part of the ontology. This stylesheet will be configured automatically to adjust the transformation process of the instances to the OWL model. It determines whether elements become classes or properties. That is necessary, because the XML instance data can have optional elements or attributes and the created stylesheet will be their common denominator. To support the separation of model and data, the OWL model will be stored separately from the OWL instances.The OWL instances will be connected to their model using the owl:

import property. Therefore every OWL instance, which refers to the OWL model, will obtain an adjustable namespace prefix. A further stylesheet is generated automatically for the conversation of XML instance data to OWL instances. The framework is designed to be easily extensible, so that the support for the missing XSD components can be included and a better support for document oriented XML, can be integrated.The conversion from XML to OWL uses Classes (owl:Class) which will emerge from xsd:complexTypes and xsd:elements according to the following rules: For the case, that an element in the source XML tree is always a leaf, containing only a literal and no attributes. This element will be mapped to an owl:DatatypeProperty having as domain the class which represents the surrounding element. XML attributes will be handled equally which means it is mapped to owl:DatatypeProperties, too. XML Schema also can contain arity constraints like xsd:minOccurs or xsd:maxOccurs, which is mapped to the equivalent cardinality constraints in OWL, owl:minCardinality and owl:maxCardinality. Table 2 summarizes the mapping based on the correspondence between XML schema elements and OWL classes and properties. OWL ontology created by this process is a full proof ontology without any discrepancy for the given source code without leaving a single piece of information. The code will also be stored in the database for future code extraction. The entire conversion process uses the concepts of XSD and OWL to convert XML to XSD using a trang tool and XSD to OWL Ontology manually. Conversion of XSD to OWL Ontology is also possible to using a tool.

V. CASE STUDY

To evaluate the present frame work a simple java code is used, to extract the file and file contents metadata into a XML file.

Source code in java

```
public class Calculator
{
        public int add(int i1, int i2)
        {
          return i1 + i2;
        }
        public int subtract(int i1, int i2)
        {
          return i1 - i2;
        }
      }
```

The java sample code is given to Doxygen Document generator tool that extracts the metadata extracted from file and file content. The contents of the file metadata are Title, Author, Owner, Year, Month, Pages, Number of Bytes, File permission attributes, Date, time, etc.,. are stored in an XML file. In the same way Javadoc is used to extract the metadata from the source code contains Function, Definition, Type, Arguments, Brief Description, Member Definition, Declaration Name, Parameters, etc.,. are stored in HBase

```
<compounddef id="classCalculator"
kind="class" prot="public">

<compoundname>Calculator</compoundname
>
      <sectiondef kind="public-func">
      <memberdef kind="function"
<type>int</type>
        <definition>int
Calculator::add</definition>
        <argsstring>(int i1, int
i2)</argsstring>
        <name>add</name>
        <param>
          <type>int</type>
          <declname>i1</declname>
        </param>
        <param>
          <type>int</type>
          <declname>i2</declname>
        </param>
        <location
      </memberdef>
      <memberdef kind="function"
<type>int</type>
        <definition>int
Calculator::subtract</definition>
        <argsstring>(int i1, int
i2)</argsstring>
        <name>subtract</name>
        <param>
          <type>int</type>
          <declname>i1</declname>
        </param>
        <param>
          <type>int</type>
          <declname>i2</declname>
        </param>
        <location      </memberdef>
      </sectiondef>
          <location    <listofallmembers>

      <scope>Calculator</scope><name>add</na
      me></member>
           <member
      <scope>Calculator</scope><name>subtrac
      t</name></member>
          </listofallmembers>
        </compounddef>
      </doxygen>
```

The goal is to transform this syntactic XML file into an OWL file by creating instances of the classes with the XSLT Stylesheet. The conversion from XML to XSD is carried out using Trang, which is open source software. The XSD for the given XML file is shown below

```
<xs:complexType>
      <xs:sequence>
        <xs:element ref="compounddef"/>
      </xs:sequence>
      <xs:attribute name="version"
use="required" type="xs:NMTOKEN"/>
      <xs:attribuy te
ref="xsi:noNamespaceSchemaLocation"
use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="compounddef">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="compoundname"/>
        <xs:element ref="sectiondef"/>
        <xs:element ref="briefdescription"/>
        <xs:element
ref="detaileddescription"/>
        <xs:element ref="location"/>
        <xs:element ref="listofallmembers"/>
      </xs:sequence>
      <xs:attribute name="id" use="required"
type="xs:NCName"/>
      <xs:attribute name="kind"
use="required" type="xs:NCName"/>
      <xs:attribute name="prot"
use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="compoundname"
type="xs:NCName"/>
  <xs:element name="sectiondef">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded"
ref="memberdef"/>
      </xs:sequence>
      <xs:attribute name="kind"
use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="memberdef">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="type"/>
        <xs:element ref="definition"/>
        <xs:element ref="argsstring"/>
        <xs:element ref="name"/>
```

```
        <xs:element maxOccurs="unbounded"
ref="param"/>
        <xs:element ref="briefdescription"/>
        <xs:element
ref="detaileddescription"/>
        <xs:element ref="inbodydescription"/>
        <xs:element ref="location"/>
      </xs:sequence>
      <xs:attribute name="const"
use="required" type="xs:NCName"/>
      <xs:attribute name="explicit"
use="required" type="xs:NCName"/>
      <xs:attribute name="id"
use="required"/>
      <xs:attribute name="inline"
use="required" type="xs:NCName"/>
      <xs:attribute name="kind"
use="required" type="xs:NCName"/>
      <xs:attribute name="prot"
use="required" type="xs:NCName"/>
      <xs:attribute name="static"
use="required" type="xs:NCName"/>
      <xs:attribute name="virt"
use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="definition"
type="xs:string"/>
  <xs:element name="argsstring"
type="xs:string"/>
  <xs:element name="param">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="type"/>
        <xs:element ref="declname"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="declname"
type="xs:NCName"/>
  <xs:element name="inbodydescription">
    <xs:complexType/>
  </xs:element>
  <xs:element name="listofallmembers">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded"
ref="member"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="member">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="scope"/>
        <xs:element ref="name"/>
      </xs:sequence>
      <xs:attribute name="prot"
use="required" type="xs:NCName"/>
      <xs:attribute name="refid"
use="required"/>
      <xs:attribute name="virt"
use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="scope" type="xs:NCName"/>
  <xs:element name="type" type="xs:NCName"/>
  <xs:element name="name" type="xs:NCName"/>
  <xs:element name="briefdescription">
    <xs:complexType/>
  </xs:element>
  <xs:element name="detaileddescription">
    <xs:complexType/>
  </xs:element>
  <xs:element name="location">
    <xs:complexType>
      <xs:attribute name="bodyend"
use="required" type="xs:integer"/>
      <xs:attribute name="bodyfile"
use="required"/>
      <xs:attribute name="bodystart"
use="required" type="xs:integer"/>
      <xs:attribute name="file"
use="required"/>
      <xs:attribute name="line"
use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The implementation transforms XML to OWL using XSLT [24]. Applying the previous XSD Stylesheet to the XML file, the OWL Ontology presented below is created manually

```
  <Project rdf:ID="CalPrj">
<hasPackages>
      <Package rdf:ID="default">
<hasClasses>
          <Class rdf:ID="Calculator">
            <hasMethods>
<Method rdf:ID="add">
<hasParameters>
                  <Parameter rdf:ID="i1">
                    <Name >i1</Name>
<DataType   >int</DataType>
</Parameter>
</hasParameters>
<hasParameters>
<Parameter rdf:ID="i2">
<DataType >int</DataType>
<Name >i2</Name>
</Parameter>
</hasParameters>
<Identifier >public</Identifier>
<Name >add</Name>
</Method>
</hasMethods>
<hasMethods
<Method rdf:ID="subtract">
<Name >subtract</Name>
<hasParameters rdf:resource="#i1"/>
<hasParameters rdf:resource="#i2"/>
<Returns >int</Returns>
<Identifier >public</Identifier>
</Method>
</hasMethods>
<Name
>Calculator</Name>
<Identifier
>public</Identifier>
```

```
</Class>
</hasClasses>
<Name >default</Name>
</Package>
</hasPackages>
</Project>
</rdf:RDF>
```

The code components stored in the HBase will be linked to the method signature in the OWL ontology for retrieval purpose. The components will be reused for the new project appropriately. The obtained OWL Ontology successfully loads on both Protégé Editor and Altova Semantics Works.

VI. Conclusion

The paper presents an approach for generating ontologies automatically out of existing XML data from source code. This approach helps to integrate conventional XML and source code into the Semantic Web. OWL is semantically much more expressive than needed for the results of our mapping. With these sample tests the paper argues that it is indeed possible to transform XML structures into OWL ones using XSLT. As the source XML structure and Ontology complexity increases, the XSLT stylesheet will get more and more complex. The purpose of the paper is to achieve the code reusability for the software development. By creating OWL ontology for the source code the future will be to search and extract the code and components and reuse to shorten the software development life cycle. Open source code can also used to create OWL Ontology so that there will be huge number of components which can be reused for the development.

VII. References

1) Grigoris Antoniou and Frank van Harmelen, A Semantic Web Primer", PHI Learning Private Limited, New Delhi, 2010, pp 1-3
2) Bung. M, Treatise on Basic Philosophy. Ontology I. The Furniture of the World. Vol. 3, Boston: Reidel.
3) Won Kim: On Metadata Management Technology Status and Issues", in *Journal of Object Technology*, vol. 4, no. 2, 2005, pp. 41-47
4) Dublin Core Metadata Initiative. < http://dublincore.org/documents/>,2002.
5) IEEE Learning Technology Standards Committee, http://ltsc.ieee.org/wg12, IEEE Standards for Learning Object Metadata (1484.12.1)
6) Darmoni, Thirion, Metadata Scheme for Health Resources" *AmericanMedical Informatics Association, 2000 Jan–Feb; 7(1): 108–109.*
7) MPEG-7 Overview: ISO/IEC JTC1/SC29/WG11 N4980, Kla-genfurt, July 2002.
8) Jason Venner, Pro Hadoop : Build Scalable, distributed applications in the cloud, Apress, 2009.
9) Tom White, Hadoop: The Definitive Guide, O'Reilly Media, Inc., 2009.
10) Gruber, T. What is an Ontology? (September, 2005): http://www.ksl-stanford.edu/kst/what-is-an-ontology.html.
11) Yang, X. Ontologies and How to Build Them. 2001. (March, 2006): http://www.ics.uci.edu/~xwy/publications/area-exam.ps.
12) Bugaite, D., O. Vasilecas. Ontology-Based Elicitation of Business Rules. In A. G. Nilsson, R. Gustas, W. Wojtkowski, W. G. Wojtkowski, S. Wrycza, J. Zupancic Information Systems Development: Proc. of the ISD'2004. Springer- Verlag, Sweden, 2006, pp. 795-806.
13) Peter Toft, Introduction to Doxygen 1994
14) Gray Davis, REVISED PRESIDING MEMBERS PROPOSED DECISION,2000
15) Michaël Van Canneyt, Free Pascal code documenter: Reference manual, 2000
16) Bastian Feder, The Beauty and the Beast, 2009
17) Bert Bos. The XML data model. http://www.w3.org/XML/Datamodel.html, 1997.
18) Graham Klyne, Jeremy J. Caroll, and Brian McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, W3C, http://www.w3.org/TR/2002/WD-rdf-concepts-20021108/, 2002.
19) Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, W3C,http://www.w3.org/TR/2002/WD-rdf-schema-20021112/, 2002.
20) James Clark. XSL Transformations (XSLT). Technical report, W3C, http://www.w3.org/TR/xslt, 1999.
21) Stefan Mintert. Schluesselqualifikation; XML jenseits des Mainstreams. iX, 8:48–51, 2005.
22) Charlie Halpern-Hamu. Transform a sample instance to a schema. http://incrementaldevelopment.com/papers/xsltrick/ ,1999.
23) XSLT Quickly, Bob DuCharme, Manning Publications, ISBN 1- 930110-11-1.
24) Mastering XML Transformations, Doug Tidwell, O'REILLY, ISBN 0596000537.