To Generate the Ontology from Java Source Code

OWL Creation

Gopinath Ganapathy [1]
[1] Department of Computer Science,
Bharathidasan University,
Trichy, India.
gganapathy@gmail.com

S. Sagayaraj [2]
[2] Department of Computer Science,
Sacred Heart College,
Tirupattur, India
sagisara@gmail.com

***Abstract*—Software development teams design new components and code by employing new developers for every new project. If the company archives the completed code and components, they can be reused with no further testing unlike the open source code and components. Program File components can be extracted from the Application files and folders using API's. The proposed framework extracts the metadata from the source code using QDox code generators and stores it in the OWL using Jena framework automatically. The source code will be stored in the HDFS repository. Code stored in the repository can be reused for software development. By Archiving all the project files in to one ontology will enable the developers to reuse the code efficiently.**

Keywords- component: Metadata; QDox, Parser, Jena, Ontology, Web Ontology Language and Hadoop Distributed File System;.

I. INTRODUCTION

Today's Web content is huge and not well-suited 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]. In the semantic web, ontologies can be used to encode meaning into a web page, which will enable the intelligent agents to understand the contents of the web page. Ontologies increase the efficiency and consistency of describing resources, by enabling more sophisticated functionalities in development of knowledge management and information retrieval applications. 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.

For every new project, Software teams design new components and code by employing new developers. If the company archives the completed code and components, it can be used with no further testing unlike open source code and components. File content metadata can be extracted from the Application files and folders using API's. During the development each developer follows one's own methods and logic to perform a task. So there will be different types of codes for the same functionalities. For instance to calculate the factorial, the code can be with recursive, non-recursive process and with different logic. In organizational level a lot of time is spent in re-doing the same work that had been done already. This has a recursive effect on the time of development, testing, deployment and developers. So there is a base necessity to create system that will minimize these factors.

Code re-usability is the only solution for this problem. This will reduce the development of an existing work and testing. As the developed code has undergone the rigorous software development life cycle, it will be robust and error free. There is no need to re-invent the wheel. Code reusability was covered in more than two decades. But still it is of syntactic nature. The aim of this paper is to extract the methods of a project and store the metadata about the methods in the OWL. OWL stores the structure of the methods in it. Then the code will be stored in the distributed environment so that the software company located in various geographical areas can access. To reuse the code, a tool can be created that can extract the metadata such as function, definition, type, arguments, brief description, author, and so on from the source code and store them in OWL. This source code can be stored in the HDFS repository. For a new project, the development can search for components in the OWL and retrieve them at ease[3].

The paper begins with a note on the related technology required in Section 2. The detailed features and framework for source code extractor is found in Section 3. The metadata extraction from the source code is in section 4. The metadata extracted is stored in OWL using Jena framework is in section 5. The implementation scenario is in Section 6. Section 7 deals with the findings and future work of the paper.

II. RELATED WORK

A. 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 [4]. 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) [5] is perhaps the largest candidate in 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 [6], is a metadata standard for Learning Objects. It has approximately 100 fields to define any learning object. Medical Core Metadata (MCM) [7] is a Standard Metadata Scheme for Health Resources. MPEG-7 [8] 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.

B. Hadoop & HDFS

The Hadoop project promotes the development of open source software and it supplies a framework for the development of highly scalable distributed computing applications [9]. Hadoop is a free, Java-based programming framework that supports the processing of large data sets in a distributed computing environment and it also supports data intensive distributed application. Hadoop is designed to efficiently process large volumes of information[10]. It connects many commodity computers so that they could work in parallel. Hadoop ties smaller and low-priced machines into a compute cluster. It is a simplified programming model which allows the user to write and test distributed systems quickly. It is an efficient, automatic distribution of data and it works across machines and in turn it utilizes the underlying parallelism of the CPU cores.

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. In addition to this, each part is replicated across several machines, so that a single machine failure does not lead to non-availability of any data. The monitoring system then re-replicates the data in response to system failures which can result in partial storage. Even though the file parts are replicated and distributed across several machines, they form a single namespace, so their contents are universally accessible. Map Reduce [11] is a functional abstraction which provides an easy-to-understand model for designing scalable, distributed algorithms.

C. 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 [12].Ontology defines the basic terms and their relationships comprising the vocabulary of an application domain and the axioms for constraining the relationships among terms [13]. This definition explains what an ontology looks like [14].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. SOURCE CODE EXTRACTOR FRAMEWORK

After the completion of a project, all the project files are sent to Source code extraction framework that extracts metadata from the source code. Only java projects are used for this framework. The java source file or folder that consists of java files is passed as input along with project information like description of the project, version of the project. The framework extracts the metadata from the source code using QDox code generators and stores it in the OWL using Jena framework. The source code is stored in the Hadoop's HDFS. A sketch of the source code extractor tool is shown in "Fig. 1".

Source code extraction framework performs two processes: Extracting Meta data from the source code using QDox and storing the meta-data in to OWL using Jena. Both the operations are performed by API's. This source code extractor will integrate these two operations in a sequenced manner. The given pseudo code describes the entire process of the framework.

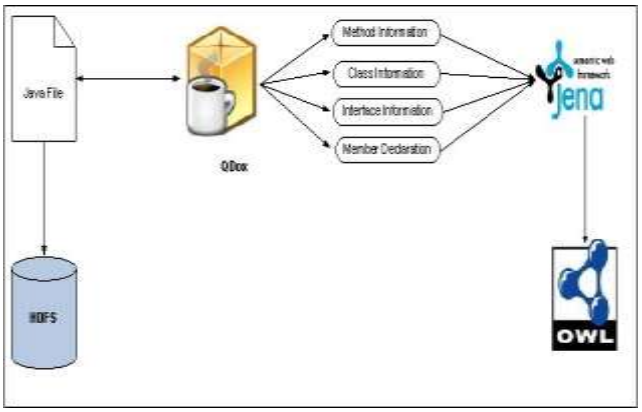

Figure 1. The process of Semantic Stimulus Tool

The framework takes project folder as input and counts the number of packages. Each package information is stored in the OWL. Each package contains various classes and each class has many methods. The class and method information is stored in the OWL. For each of method, the information such as return type, parameters and parameter type information are stored in the OWL. The framework which places all the information in the persistence model and it is stored in the OWL file.

1. Get package count by passing the file path.
2. Initialize packageCounter to zero
3. While the package count equal to packageCounter
3.1 Store the package[packageCounter] Information into OWL model
3.2 Initilalize classCounter value is equal to zero.
3.3 Get the no of class count
3.4 While class count equal to classCounter
3.4.1 Store the class[ClassCounter] Information into OWL model
3.4.2 Initialize methodCounter to zero
3.4.3 Get no of method of the class[packageCounter]
3.4.4 While no of method count is equal to zero
3.4.4.1 Store the method[methodCounter] information into OWL.
3.4.4.2 Store the modifier informaton of the method [classCounter]
3.4.4.3 Store the return type of the method[classCounter]
3.4.4.4 Initialize the parmCounter to zero
3.4.4.5 Get the no of parameters of method[methodCounter]
3.4.4.6 While no of paramerters count is equal to zero
3.4.4.6.1 Store the parameter[paramCounter] information into OWL model
3.4.4.6.2 Increase paramCounter by one
3.4.4.7 Increase methodCounter by one
3.4.5 Increase class Counter by one
3.5 Increase package Counter by one
4. Write the OWL model in the OWL File.

IV. Extracting Metadata

QDox is a high speed small footprint parser for extracting classes, interfaces, and method definitions from the source code. It is designed to be used by active code generators or documentation tools. This tool extracts the metadata from the given java source code. To extract the meta-data of the source, the given order has to be followed. When the java source file or folder that has the java source file is loaded to QDox, it automatically performs the iteration. The loaded information is stored in the JavaBuilder object. From the java builder object the list of packages, as an array of string, are returned. This package list has to be looped to get the class information. From the class information, the method information is extracted. It returns the array of JavaMethod. Out of these methods, the information like scope of the method, name of method, return type of the method and parameter information is extracted.

The QDox process uses its own methods to extract various metadata from the source code. The getPackage() method lists all the available packages for a given source. The getClasses() method lists all the available classes in the package. The getMethods() method lists all the available methods in a class. The getReturns() method returns the return type of the method. The getParameters() method lists all the parameters available for the method. The getType() method returns the type of the method. And when the getComment() method is used with packages, classes and methods, it returns the appropriate comments. Using the above methods the project informations such as package, class, method, retune type of the method, parameters of the method, method type and comments are extracted by the QDox. These metadata are passed to the next section for storing in the OWL.

V. Storing Metadata in OWL

To store the metadata extracted by QDox, the Jena framework is used. Jena is a Java framework for manipulating ontologies defined in RDFS and OWL Lite [15]. Jena is a leading Semantic Web toolkit [16] for Java programmers. Jena1 and Jena2 are released in 2000 and August 2003 respectively. The main contribution of Jena1 was the rich *Model* API. Around this API, Jena1 provided various tools, including I/O modules for: RDF/XML [17], [18], N3 [19], and N-triple [20]; and the query language RDQL [21]. In response to these issues, Jena2 has a more decoupled architecture than Jena1. Jena2 provides inference support for both the RDF semantics [22] and the OWL semantics [23].

Jena contains many APIs out of which only few are used for this framework like addProperty(), createIndividual() and write methods. The addProperty() method is to store data and object property in the OWL Ontology. CreateIndividual() creates the individual of the particular concepts. Jena uses in-memory model to hold the persistent data. So this has to be written in to OWL Ontology using write() method.

The OWL construction is done with Protégé. Protégé is an open source tool for managing and manipulating OWL[24]. Protégé [25] is the most complete, supported and used framework for building and analysis of ontologies [26, 27, 28]. The result generated in Protégé is a static ontology definition [29] that can be analyzed by the end user. Protégé provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based API for building knowledge-based tools and applications.

Based on the java source code study the ontology domain is created with the following attributes. To store the extracted metadata, the ontology is created with project, packages, classes, methods and parameters. The project is concept that holds the information like name, project repository location, project version and the packages. The package is a concept that holds the information like name and the class. The class is a concept that holds the class informations such as author, class comment, class path, identifier, name and the methods. The method is a concept that holds the information like method name, method Comment, method identifier, isConstructor, return type, and the parameter. The parameter is a concept that holds the information like name and the data type.

Concepts/Classes provide an abstraction mechanism for grouping resources with similar characteristics. Project, package, class, method, parameter are concepts in source code extractor ontology.

Individual is an instance of the concept/ class.

Property describes the relation between concepts and objects. It is a binary relationship on individuals. Each property has domain and range. There are two types of property namely object and data property

Object Property links individuals to individuals. In source code ontology, the object properties are hasClass, hasMethod, hasPackage and hasParameter. hasClass is an object property which has domain Package and range Class. hasMethod is an object property which has domain class and range method. hasPackage is an object property which has domain Project and range Package. hasParameter is an object property which has domain method and range range.

Datatype Property links individuals to data values. Author is a dataproperty which has domain Class and the String as range. ClassComment is a data property which has domain class and string as range. DataType is a data property which has domain parameter and the range string as range. Identifier is a data property which has domain method,class and the range boolean as range. IsConstructor is a data property which has domain method and string as range. MethodComment is a data property which has domain method and string as range. Name is a data property which has domain project, package, class, method, parameter and string as range. Project_Date is a data property which has domain project and string as range. Project_Description is a data property which has domain project and string as range. Project_Repository_Location is a data property which has domain project and string as range. Project_Description is a data property which has domain project and string as range. Project_Version is a data property which has domain project and string as range. Returns is a data property which has domain method and string as range.

VI. CASE STUDY

To evaluate the proposed framework the following simple java code is used.

```
package com.sourceExtractor.ontology;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import org.apache.log4j.Logger;
import org.apache.log4j.spi.RootLogger;
import com.hp.hpl.jena.ontology.DatatypeProperty;
/**   * To manage the ontology related informations
 * @author Sagayaraj  */
public class OntoManager {
                private Logger LOGGER =
RootLogger.getLogger(OntoManager.class);
            @SuppressWarnings("static-access")
            public OntModel getModel(String modelLocation) {
  OntModel ontModel = null;
   ontModel = ModelFactory.createOntologyModel();
              ontModel.read(new FileManager().get().open(modelLocation),
"");
              return ontModel;                }
            /**
             * To create Individual In OWL
             * @param model
             * @param concept
             * @param individual              */
            public void createIndividual(OntModel model, String concept,
String individual) {
                OntClass ontClass =
model.getOntClass(addNameSpace(concept, model));
            model.enterCriticalSection(Lock.WRITE);
                try {if (ontClass != null) {
            ontClass.createIndividual(addNameSpace(individual, model));
                            } else {
LOGGER.error("Direct Class is null");// todo
                            }} finally {
                            model.leaveCriticalSection();
                }               }}
```

The sample java code is given as input to QDox document generator through the Graphical User Interface (GUI) provided in the "Fig. 2".

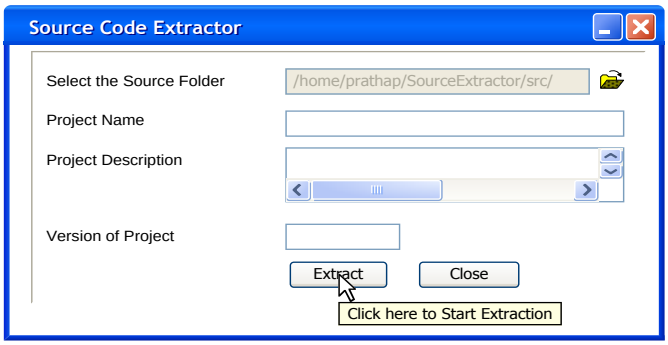

Figure 2. GUI for locating folder

Using the QDox API's metadata is extracted as given in the Table 1. The output of the QDox stores metadata in the form of strings. To store the metadata the OWL ontology, template is created using Protégé. The strings are passed to the Jena framework and the APIs place the metadata in to the OWL Ontology. The entire project folder, stored in the HDFS, is 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. The sample OWL file is given below as the output of the framework.

```
 <owl:Ontology                          rdf:about="http://www.owl-
ontologies.com/SourceExtractorj.owl#"/>
 <owl:Class                             rdf:about="http://www.owl-
ontologies.com/SourceExtractorj.owl#Package"/>
 <owl:Class                             rdf:about="http://www.owl-
ontologies.com/SourceExtractorj.owl#Project"/>
 <owl:Class                             rdf:about="http://www.owl-
ontologies.com/SourceExtractorj.owl#Class"/>
 <owl:Class                             rdf:about="http://www.owl-
ontologies.com/SourceExtractorj.owl#Method"/>
 <owl:Class                             rdf:about="http://www.owl-
ontologies.com/SourceExtractorj.owl#Parameter"/>
 <owl:ObjectProperty                    rdf:about="http://www.owl-
ontologies.com/SourceExtractorj.owl#hasPackage">
  <rdfs:domain                          rdf:resource="http://www.owl-
ontologies.com/SourceExtractorj.owl#Project"/>
  <rdfs:range                           rdf:resource="http://www.owl-
ontologies.com/SourceExtractorj.owl#Package"/>
```

```
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="http://www.owl-ontologies.com/SourceExtractorj.owl#hasParameter">
   <rdfs:domain rdf:resource="http://www.owl-ontologies.com/SourceExtractorj.owl#Method"/>
   <rdfs:range rdf:resource="http://www.owl-ontologies.com/SourceExtractorj.owl#Parameter"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="http://www.owl-ontologies.com/SourceExtractorj.owl#hasClass">
   <rdfs:domain rdf:resource="http://www.owl-ontologies.com/SourceExtractorj.owl#Package"/>
   <rdfs:range rdf:resource="http://www.owl-ontologies.com/SourceExtractorj.owl#Class"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="http://www.owl-ontologies.com/SourceExtractorj.owl#hasMethod">
   <rdfs:range rdf:resource="http://www.owl-ontologies.com/SourceExtractorj.owl#Method"/>
   <rdfs:domain rdf:resource="http://www.owl-ontologies.com/SourceExtractorj.owl#Class"/>
 </owl:ObjectProperty>  <owl:DatatypeProperty rdf:about="http://www.owl-ontologies.com/SourceExtractorj.owl#Project_Date"/>
 <owl:DatatypeProperty rdf:about="http://www.owl-ontologies.com/SourceExtractorj.owl#Identifier">
   <rdfs:domain>
    <owl:Class>
     <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="http://www.owl-ontologies.com/SourceExtractorj.owl#Method"/>
      <owl:Class rdf:about="http://www.owl-ontologies.com/SourceExtractorj.owl#Class"/>
     </owl:unionOf>
    </owl:Class>
   </rdfs:domain>
   <rdfs:range>
    <owl:Class>
     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DataRange"/>
     <owl:oneOf rdf:parseType="Resource">
      <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >public</rdf:first>
      <rdf:rest rdf:parseType="Resource">
       <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >private</rdf:first>
       <rdf:rest rdf:parseType="Resource">
        <rdf:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
         >protected</rdf:first>
         <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
       </rdf:rest>
      </rdf:rest>
     </owl:oneOf>
    </owl:Class>
   </rdfs:range>
 </owl:DatatypeProperty>
```

VII. Conclusion and Future Work

This paper presents an approach for generating ontologies using the source code extractor tool from source code. This approach helps to integrate 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 source code in to OWL using this Source Code Extractor framework. The framework created OWL which will increase the efficiency and consistency in development of knowledge management and information retrieval applications. The purpose of the paper is to achieve the code re-usability for the software development. By creating OWL 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 be used to create OWL so that there will be huge number of components which can be reused for the development. By storing the projects in the OWL and the HDFS the corporate knowledge grows and the developers will use more of reuse code than developing themselves. Using the reuse code the development cost will come down, development time will become shorter, resource utilization will be less and quality will go up.

After developing OWL and storing the source code in the HDFS, the code components can be reused. The future work can take off in two ways. One can take a design document from the user as input, then extract the method signature and try to search and match in the OWL. If the user is satisfied with the method definition, it can be retrieved from the HDFS where the source code is stored. Second one can take the project specification as input and text mining can be performed to extract the keywords as classes and the process as methods. The method prototype can be used to search and match with the OWL and the required method definition can be retrieved from the HDFS. The purpose of storing the metadata in OWL is to minimize the factors like time of development, time of testing, time of deployment and developers. Creating OWL using this framework can reduce these factors.

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] Gopinath Ganapathy and S. Sagayaraj, “Automatic Ontology Creation by Extracting Metadata from the Source code ”, in Global Journal of Computer Science and Technology,Vol.10, Issue 14(Ver.1.0) Nov. 2010. pp.310-314.

[4] Won Kim: “On Metadata Management Technology Status and Issues”, in Journal of Object Technology, vol. 4, no.2, 2005, pp. 41-47.

[5] Dublin Core Metadata Initiative. < http://dublincore.org/documents/>,2002.

[6] IEEE Learning Technology Standards Committee, http://ltsc.ieee.org/wg12, IEEE Standards for Learning Object Metadata (1484.12.1)

[7] Darmoni, Thirion, “Metadata Scheme for Health Resources” American Medical Infor. Association, 2000 Jan–Feb; 7(1): 108–109.

[8] MPEG-7 Overview: ISO/IEC JTC1/SC29/WG11 N4980, Kla-genfurt, July 2002.

[9] Jason Venner, “Pro Hadoop : Build Scalable, Distributed Applications”, in the cloud, Apress, 2009.

[10] Gopinath Ganapathy and S. Sagayaraj, “Circumventing Picture Archiving and Communication Systems Server with Hadoop Framework in Health Care Services”, in Journal of Social Science, Science Publication 6 (3) : pp.310-314.

[11] Tom White, “Hadoop: The Definitive Guide”, O’Reilly Media, Inc., 2009.

[12] Gruber, T. “What is an Ontology?” ,September, 2005: http://www.ksl- stanford.edu/kst/what-is-an-ontology.html.

[13] Yang, X. “Ontologies and How to Build Them”,(March, 2006):

http://www.ics.uci.edu/~xwy/publications/area-exam.ps.
[14] 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.
[15] McCarthy, P," Introduction to Jena": www-106.ibm.com/developerworks/java/library/j-jena/, , 22.02.2005.
[16] B. McBride," *Jena* IEEE Internet Computing", July/August, 2002.
[17] J.J. Carroll." CoParsing of RDF & XML", HP Labs Technical Report, HPL-2001-292, 2001
[18] J.J. Carroll, "Unparsing RDF/XML,WWW2002": http://www.hpl.hp.com/techreports/2001/HPL-2001- 292.html
[19] T. Berners-Lee et al," Primer: Getting into RDF & Semantic Web using N3", http://www.w3.org/2000/10/swap/Primer.html
[20] J. Grant, D. Beckett, "*RDF Test Cases"*, 2004, W3C6.
[21] L. Miller, A. Seaborne, and A. Reggiori,"Three Implementations of SquishQL, a Simple RDF Query Language"**,** 2002, p 423.
[22] P. Hayes, "RDF Semantics*"*, 2004, W3C.
[23] P.F. Patel-Schneider, P. Hayes, I. Horrocks," OWL Semantics & Abstract Syntax", 2004, W3C.
[24] Protégé Semantic Web Framework, http://protege.stanford.edu/overview/protege-owl.html, accessed 16th October 2010.
[25] Protégé. – http://protege.stanford.edu/ontologies/ontologyOfScience.
[26] 9th Intl. Protégé Conference - July 23-26, 2006 – Stanford, Californiahttp://protege.stanford.edu/conference/2006.
[27] 10th Intl. Protégé Conference - July 15-18, 2007 – Budapest, Hungaryhttp://protege.stanford.edu/conference/2007.
[28] 11th Intl. Protégé Conference - June 23-26, 2009 – Amsterdam, Netherlandshttp://protege.stanford.edu/conference/2009.
[29] Hai H. Wang, Natasha Noy, Alan Rector, Mark Musen, Timothy Redmond, Daniel Rubin, Samson Tu, Tania Tudorache, Nick Drummond, Matthew Horridge, and Julian Sedenberg," Frames and OWL side by side". In 10th International Protégé Conference,Budapest, Hungary, July 2007.

AUTHORS PROFILE

Gopinath Ganapathy is the Professor & Head, Department of Computer Science and Engineering in Bharathidasan University, India. He obtained his under graduation and post-graduation from Bharathidhasan University, India in 1986 and 1988 respectively. He submitted his Ph.D in 1996 in Maduari Kamaraj University, India. Received Young Scientist Fellow Award for the year 1994 and eventually did the research work at IIT Madras. He published around 20 research papers. He is a member of IEEE, ACM, CSI, and ISTE. He was a Consultant for a 8.5 years in the international firms in the USA and the UK, including IBM, Lucent Technologies (Bell Labs) and Toyota. His research interests include Semantic Web, NLP, Ontology, and Text Mining.

S. Sagayaraj is the Associate professor in the Department of Computer Science, Sacred Heart College, Tirupattur, India. He did his Bachelor Degree in Mathematic in Madras University, India in 1985. He completed his Master of Computer Applications in Bharadhidhasan University, India in 1988. He obtained Master of Philosophy in Computer Science from Bharathiar University, India in 2001. He registered for Ph.D. programme in Bharathidhasan University, India in 2008. His Research interests include Data Mining, Ontologies and Semantic Web.

TABLE I. METADATA EXTRACTED FROM THE SAMPLE CODE

Project	Project Name	Ontology_Learn
	Project Version	1.0.0
	Project Date	10/10/10
	Repository Location	/opt/SourceCodeExtrctor/
	HasPackage	com.sourceExtractor.ontology
Package	Name	com.sourceExtractor.ontology
	HasClass	OntoManager
Class	Name	OntoManager
	Class Comment	It manage the ontology operation
	Class Path	/SampleOntology/com/sourceExtractor/ontology/OntoManager.java
	Author	Sagayaraj
	Identifier	Public
	HasMethod	getModel createIndividual
Method	Name	getModel createIndividual
	Identifier	Public Public
	Returns	OntoModel Void
	Method Comment	-undefined- To add the data property in owl file
	IsConstructor	FALSE FALSE
	HasParameter	modelLocation Individual model Concept
Parameter	Name	modelLocation
	Data Type	java.lang.String
	Name	Individual
	Data Type	java.lang.String
	Name	Model
	Data Type	OntModel
	Name	Concept
	Data Type	java.lang.String