Australian Journal of Basic and Applied Sciences, 8(16) October 2014, Pages: 413-418

AENSI Journals

Australian Journal of Basic and Applied Sciences

ISSN:1991-8178

Journal home page: www.ajbasweb.com

Ontology Based Software Storage Repository to Enhance Software Reuse

[1]R. Jayasudha and [2]S. Subramanian

[1]*Department of Computer Applications Sri Krishna College of Engineering And Technology Coimbatore, India.*
[2]*Vice Chancellor Karpagam University Coimbatore, India.*

ARTICLE INFO

Article history:
Received 20 August 2014
Received in revised form
19 September 2014
Accepted 23 October 2014
Available online 16 November 2014

Keywords:
Software Reuse, Semantic Web, Ontology, Software Engineering

ABSTRACT

Software Reuse plays an important role in the world of software development. This leads to better development time and cost. Software Reuse is not used much because of unavailability of needed components, Unknown available component, Unstructured storage of relevant component, improper retrieval of components, unwillingness of developer etc. Unstructured Storage of Component is taken as a problem and an attempt is made to create a structured repository. This paper proposes a work of Semantic based Structured Repository by creating an Ontology based index for the final data in the hard disk. This index helps us to identify the needed record from the disk. Ontology based repository reduces the duplicate record retrieval, improves the precision and recall. The main goal of this proposed work is to develop a methodology that can simplify and automate the software reuse.

To Cite This Article: R. Jayasudha and S. Subramanian, Ontology Based Software Storage Repository to Enhance Software Reuse. ***Aust. J. Basic & Appl. Sci.,*** *8(16): 413-418, 2014*

INTRODUCTION

Software reuse is a very old concept emerged during 1960's at the NATO conference on Software Engineering. Software Reuse is the concept of creating new software from the existing software instead of building complete new software. This can also be defined as creating existing software by using the already existing software development knowledge (SDK). SDK is nothing but various phases in the software life cycle like Software Requirement Specification (SRS), Software Design (SD), Software Coding (SC), Software Testing (ST), and Test Cases (TC). SDK refer to a piece of Knowledge that can apply to the development process of the software.

Software Reuse is a method for developing new component adding some extra functionality to the existing ones. Reuse helps to reduce cost and to improve quality and time-to-market. The main problem with the reuse process is increased maintenance cost, Lack of tool support, Not-invented-here syndrome. The third problem can be solved only by changing the attitude of the software developers. The second problem can be solved by creating a development environment where the developer gets all the needed reusable components as a preview as he enters the requirement. Reuse enables developers to leverage past accomplishments and can facilitate significant improvements in software productivity and quality, shorten the development cycle and reduce development costs.

The Reuse life cycle RLC for reusable SDK can be generally characterized into following six steps:

1. List the requirements.
2. Search for SDK.
3. Evaluate the retrieved SDK.
4. Select the suitable SDK
5. Adapt the SDK for the Current Project.
5. Integrate the SDK.

The proposed work will try to improve the second, third and fourth stage of the reuse life cycle by creating Ontology based Reuse Repository.

I. A.Ontology:

Ontology is a formal, explicit specification of a shared conceptualization. (Gruber, 1993; Borst, 1997). A "conceptualization" is an abstract model of a phenomenon, created by identification of the relevant concepts of the phenomenon. The concepts, the relations between them and the constraints on their use are explicitly defined.

Corresponding Author: R. Jayasudha, Department of Computer Applications Sri Krishna College Of Engineering And Technology Coimbatore, India.
E-mail: jayasudhasubburaj@gmail.com

"Formal" means that Ontology is machine-readable and excludes the use of natural languages. For example, in medical domains, the concepts are diseases and symptoms, the relations between them are causal and a constraint is that a disease cannot cause itself. Ontology captures knowledge independently of its use and in a way that can be shared universally, but practically different tasks and uses call for different representations of the knowledge in Ontology.

Ontology is sometimes confused with taxonomy, which is a classification of the data in a domain. The difference between them is in two important contexts:

1. Ontology has a richer internal structure as it includes relations and constraints between the concepts.
2. Ontology claims to represent a certain consensus about the knowledge in the domain.

Ontology aim to represent a form of common agreement regarding the knowledge they represent, they are often created in a cooperative process involving different people, sometimes at different places. Ontology is divided to types in accord with the degree of generality of the principles they contain.

Literature Survey:

After completion of every Software Project the details about the project are stored in the repository. We can classify the details into various categories such as Project Source code, Code Management Systems (CMS), Defect tracking systems, Communication between developers and users, Meta-data about the projects.

The following are some of the existing Software Repository FLOSSMole, FLOSSMetrics, PROMISE (PRedictOr Models In Software Engineer-ing), Qualitas Corpus (QC), Sourcerer Project, Ultimate Debian Database (UDD), Bug Prediction Dataset (BPD), The International Software Benchmarking Standards Group (ISBSG), Eclipse Bug Data (EBD), Software-artifact Infrastructure Repository (SIR), ohloh, SourceForge Research Data Archive (SRDA).

These repositories are analysed and the problem is classified into two. The one related to the extraction of data from the repository and the other related to the actual storage of data. There is large variability in the formats and tools standards, etc., Replicability, Outliers, Missing values and inconsistencies, Redundant and irrelevant attributes and instances, Overlapping or class separability, Data shifting, Imbalance, Metrics, Evaluation metrics and the evaluation of models.

II. A. Different types of Repository Classifier:

1) Free text classification:

Free text retrieval performs searches using the text contained within documents. The retrieval system is typically based upon a keyword search. All of the document indexes are searched to try to find an appropriate entry for the required keyword. An obvious flaw with this method is the ambiguous nature of the keywords used. Another disadvantage is that a search may result in many irrelevant components. This type of classification generates large overheads in the time taken to index the material, and the time taken to make a query. All the relevant text (usually file headers) in each of the documents relating to the components is index, which must then be searched from beginning to end when a query is made.

1) Enumerated classification:

Enumerated classification uses a set of mutually exclusive classes, which are all within a hierarchy of a single dimension. A prime illustration of this system is classification of books in a library. Each subject area, e.g., Biology, Chemistry etc, has its own classifying code. As a sub code of this is a specialist subject area within the main subject. These codes can again be sub coded by author. This classification method has advantages and disadvantages pivoted around the concepts of a unique classification for each item. The classification scheme will allow a user to find more than one item that is classified within the same section/subsection assuming that if more than one exists. For example, there may be more than one book concerning a given subject, each written by a different author. This type of classification schemes is one dimensional, and will not allow flexible classification of components into more than one place. As such, enumerated classification by itself does not provide a good classification scheme for reusable software components.

2) Attribute value:

The attribute value classification schemes use a set of attributes to classify a component. For example, a book has many attributes such as the author, the publisher, it's ISBN number and it's classification code in the Dewey Decimal system. These are only example of the possible attributes. Depending upon who wants information about a book, the attributes could be concerned with the number of pages, the size of the paper used, the type of print face, the publishing date, etc. Clearly, the attributes relating to a book can be: · Multidimensional. The book can be classified in different places using different attributes· Bulky. All possible variations of attributes could run into many tens, which may not be known at the time of classification.

3) Faceted Classification:

Faceted classification schemes are attracting the most attention within the software reuse community. Like the attribute classification method, various facets classify components; however, there are usually a lot fewer facets than there are potential attributes. Ruben Prieto-Diaz has proposed a faceted scheme that uses six facets. · The functional facets are: Function, Objects and Medium · The environmental facets are: System type, Functional area, Setting Each of the facets has to have values assigned at the time the component is classified. The individual components can then be uniquely identified by a tuple, for example. <add, arrays, buffer, database manager, billing, book store> Clearly, it can be sent that each facet is ordered within the system. The facets furthest to the left of the tuple have the highest significance, whilst those to the right have a lower significance to the intended component. When a query is made for a suitable component, the query will consist of a tuple similar to the classification one, although certain fields may be omitted if desired. For example: <add, arrays, buffer, database manager, *,*>The most appropriate component can be selected from those returned since the more of the facets from the left that match the original query, the better the match will be. Frakes and Pole conducted an investigation as to the most favourable of the above classification methods (Hans-Jörg Happel, 2001). The investigation found no statistical evidence of any differences between the four different classification schemes, however, the following about each classification method was noted. (Guru Rao, 2008)

Enumerated classification Fastest method, difficult to expand · Faceted classification Easily expandable, most flexible · Free text classification Ambiguous, indexing costs · Attribute value classification Slowest method, no ordering, number of attributes.

III. Proposed Work:

The main idea of the proposed work is the creation of the well organized repository. The ontology based repository is created by the following ways

A) Extraction of the metadata from the Source code in the disk:

All the project folders are stored in the form of rar or Zip files in the hard disk for the future reference. The name of the file does not indicate the content of the file. The first step in the repository creation is the rar file extraction and the source code file is opened for extraction of the knowledge from the file. Here only the java code file is used where the Package name, Class Name, Method Name, Parameters used, return data type is extracted. The methods of any class will be the actual action done in the project. The extracted method is mainly reused.

B) Creation of Ontology Software Repository Index (OSRI) using TOVE Method:

All the extracted Package, Class, Method, Parameter, Return type are stored in the form of ontology. The Ontology is developed using TOVE methontology. The TOVE Based on experiences in the development of TOVE (Toronto Virtual Enterprise) the following approach to engineering ontologies is developed

(1) Motivating scenarios: the start point is a set of problems encountered in a particular enterprise, which are often in the form of story problems or examples.

(2) Informal competency questions: requirements of the ontology, based on the motivating scenario, described as informal questions that an ontology must be able to answer; this phase acts as an evaluation on the ontological commitments made in the previous stage.

(3) Terminology specification: the objects, attributes and relations of the ontology are formally specified (usually in first order logic).

(4) Formal competency questions: the requirements of the ontology are formalised in terms of the formally defined terminology (see especially.

(5) Axiom specification: axioms that specify the definition of terms and constraints on their interpretations are given in first -order logic, guided by the formal competency questions as the axioms must be necessary and sufficient to express the competency questions and their solutions.

(6) Completeness theorems: an evaluation stage which assesses the competency of the ontology by defining the conditions under which the solutions to the competency questions are complete.

The initial description of the tasks to be supported by an ontology in terms of motivating scenarios seems to result from the methodology being abstracted from the development process undertaken in the TOVE project. Motivating scenarios are only one of many ways in which such tasks could be described and in order to arrive at a more comprehensive methodology, it would probably be necessary to incorporate other types of representation.

The TOVE approach is most interesting for the emphasis on ontology evaluation, especially as a means of performing this evaluation is provided in the form of completeness theorems. These theorems are useful in an umber of ontology maintenance tasks, e.g. assessing the extendibility of an ontology - any extension must be able to preserve the validity of the completeness theorems - or to provide a benchmark for ontology (Jones, 1998).

C) Steps involved in the creation of the Ontology file:

1) Creation of folder.owl file:

This owl file gives the detail about the folder in which the needed data is stored. The details such as the Main Folder, Domain Folder, Platform Folder, and Version Folder are maintained. The Main folder will have all the project in the form of Zip File. The Domain Folder will have the details like various domain specific projects, Platform folder consist of the details like the same domain project which is done in different platform has been identified and the details are maintained. The Version folder consist of the details like the different Versions of the same project has been maintained in this folder. The information about the actual storage path of the particular need software can be identified with the help of this Ontology.

- Folder
- Main
- Domain
- Platform
- Version

2) Creation of the file.owl file:

The File.owl is the Ontology file contains details about the files related to the same project. Some of the files are Requirement File (SRS), Design Document, Coding File, Testing Document, Maintenance manuals are maintained in this ontology file . This Owl file provides the complete details about the project and their corresponding files.

- File
- SRS
- Design
- Coding
- Testing
- Maintenance

3) Creation of the Coding.owl file:

This Ontology file consists of the details about the Coding document. The Coding Folder is extracted and the meta details like Package Name, Class Name, Methods, Parameters and the Return Types are extracted and maintained in this file. Thus this ontology will give us the details of the full coding file which helps in complete reuse. The methods identified are the most important which leads to reuse.

- Coding
- Package
- Class
- Method
- Parameter
- ReturnType

RESULT AND DISCUSSIONS

The usage of this Ontology based Repository helps to reduce the Retrieval time. Precision and recall are the basic measures used in evaluating search strategies. RECALL is the ratio of the number of relevant records retrieved to the total number of relevant records in the database. It is usually expressed as a percentage. PRECISION is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved. It is usually expressed as a percentage. A measure that combines precision and recall is the harmonic mean of precision and recall, the traditional F-measure or balanced F-score:

$$F = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

The Table-1 shows the different values of Precision, Recall and F-Measure for various queries.

Table 1: Measuring Values For The Search In The Osri

Query	Appearance (in Time New Roman or Times)					
	Precision		Recall		F-Measure	
	OSRI	Disk	OSRI	Disk	OSRI	Disk
Query 1	0.5	0.3	0.8	0.5	0.6	0.4
Query 2	0.7	0.3	0.4	0.2	0.5	0.2
Query 3	0.2	0.2	0.9	0.5	0.3	0.3
Query 4	0.8	0.6	0.3	0.1	0.4	0.2
Query 5	0.9	0.6	0.3	0.3	0.5	0.4

The Fig.1 shows the comparison between the Precision, when the queries are used for the search of software components in the Disk directly and the search using the Ontology based repository that is OSRI. The graph shows that the Precision that is relevant components are retrieved more in the OSRI Search than the disk search. Thus Ontology improves the Precision.

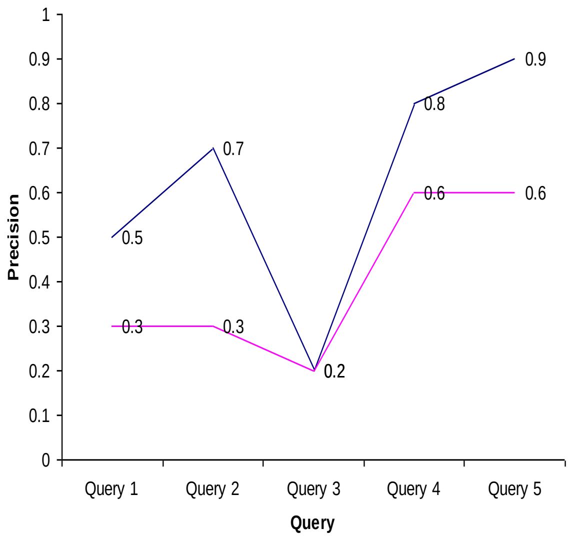

Fig. 1: A Comparison of Precision for Various Queries between OSRI & Disk Search.

The Fig.2 shows the Comparison of Recall between the normal disk search and the OSRI Search. It is understood from the graph the recall is more in the OSRI Search that is most of the relevant component are retrieved in the Ontology Based Repository Search than the disk Search.

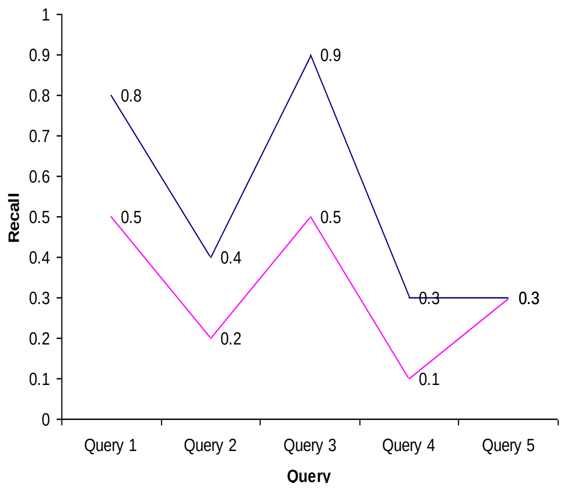

Fig. 2: A Comparison of Recall for Various Queries between OSRI & Disk Search.

The Fig.3 shows the F-Measure comparison between the two type of search. F-Measure completely depends on the Recall and Precision. Since the Recall and Precision favours the OSRI Search thus F-Measure also favours the OSRI Search than the normal disk search.

Conclusion:

The Ontology based Software Reuse helps the programmer to retrieve the needed components in a timely way. The availability of the component increases the software reuse. Software Reuse increases if the availability increases. Set of 50 project dataset is used for testing this process. Each project is stored in the disk in the form of Zip file and finally OSRI Ontology based reference index is created for each project. Semantic based Reuse plays major role in increasing the usage of the existing components. In future this method may extended by storing the data in cloud environment for distributed usage of the components.

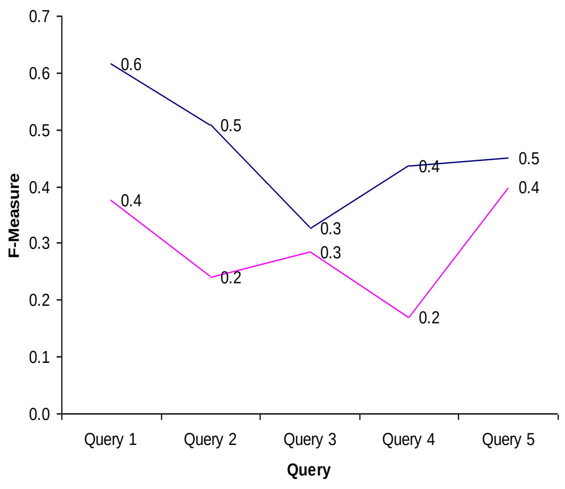

Fig. 3: A Comparison of F-Measure for Various Queries between OSRI & Disk Search.

REFERENCES

Amit Kumar, 2013. ”Software Reuse Libraries Based Proposed Classification for Efficient Retrieval of Components, International Journal of Advanced Research in Computer Science and Software Engineering, 3(6): 2277 128X.

Daniel Rodriguez, Israel Herraiz, Rachel Harrison, 2012.On Software Engineering Repositories and Their Open Problems.

Gohgfpinath Ganapathy, S. Sagayaraj, 2011. To Generate the Ontology from Java Source Code OWL Creation,(IJACSA) International Journal of Advanced Computer Science and Applications, 2: 2.

Gopinath Ganapathy, S. Sagayaraj, 2011. Extracting Code Resource from OWL by Matching Method Signatures using UML Design Document UML Extractor, (IJACSA) International Journal of Advanced Computer Science and Applications, 2: 2.

Gowtham Gajala, M.V. Phanindra, A Firm Retrieval of Software Reusable Component Based On Component Classification, International Journal of Computational Engineering Research, 03: 5.

Guru Rao, C.V. and P. Niranjan, 2008. An Integrated Classification Scheme for Efficient Retrieval of Components, Journal of Computer Science, 4(10): 821-825, 2008 ISSN 1549-3636 © 2008 Science Publications.

Hans-Jörg Happel, 2001. Stefan Seedorf, Applications of Ontologies in Software Engineering, SWESE. VaneetKaur and Shivani Goel, Facets of Software Component Repository, International Journal of Computer Science and Engineering(IJCSE).

http://www.iqlue.com/Ontology.jsp

Jones, Dean, Trevor Bench-Capon, and Pepijn Visser, 1998. Methodologies for ontology development. Proc. IT&KNOWS Conference of the 15th IFIP World Computer Congress.

Marzanah, A. Jabar, 2014. ”Meta-Analysis of Ontology Software Development Process”,IRECOS,Praise Worthy Prize.

Nagarajan, G., K.K. Thyagharajan, 2014. “Rule Based Semantic Content Extaction in Image using Fuzzy Ontology”, IRECOS,Praise Worthy Prize.

Ontology – The Review Document

Wongthongtham, P., E. Chang, T. Dillon and Sommerville, 2007. Software Engineering Ontology – the Instance Knowledge(Part II), IJCSNS International Journal of Computer Science and Network Security,” 7: 2.