International Journal of Computer Applications (0975 – 8887)
Volume 61– No.14, January 2013

An Experiment in Software Component Retrieval based on Metadata and Ontology Repository

Shekhar Singh
Assistant Professor
Department of Computer Science & Engineering
Panipat Institute of Engineering and Technology, Samalkha, Panipat, India

ABSTRACT

Software component reuse is the use of existing software components to build a new software system. Effective storage and retrieval of software components is much essential in software components reuse process. The researchers have developed a number of software components reuse techniques for storage and retrieval of software components. No one technique is complete in its own; every technique has its own merits and demerits. This paper presents a meta-data model and faceted classification for storage and retrieval of software components that considers domain semantic information based on ontologies and texonomies. In contrast to most existing repositories, which only retrieve a limited set of components, the proposed meta-data model makes possible the recommendation of interrelated components, as ontology and taxonomies characteristics were incorporated. The software component retrieval based on facet classification is a method which has been widely applied in software component retrieval, but the precision of software component retrieval is poor as a result of subjective factor in faceted classification retrieval. The architecture of software component retrieval system and the model of software component retrieval system were designed, the corresponding match algorithm was provided. According to the relation of facet and term space, meta-data repository was established and abstracted from domain knowledge which formed coherent retrieval in the domain and was applied to software component retrieval process. These terms in the meta-data repository were then used to match software components which described in the software component description repository with facet classification, related software components were retrieved from the software component repository. The results of application show that the new software component retrieval method can evidently improve the component retrieval precision and take care of the full-scale of the searching results.

General Terms

Software reuse, software component, Metadata, component retrieval, Component based engineering, Appropriate Components, General query terms, accurate describing terms

Keywords

Metadata repository, Search Engine, faceted classification, component model, heuristic algorithm, ontology, accurate query terms, ontology repository, component repository

1. INTRODUCTION

Software component reuse is an important concept to software development, as it reduces software development effort, time and cost and increase reliability and flexibility. Software component-Based Software Engineering proposes the reuse of software components, which can be retrieved and assembled into applications of specific domains [1]. In order to build these applications successfully, it is fundamental to choose appropriated software components from a collection of available software components. Thus, it is desirable to have a repository that supports the storage, query and retrieval of software components and makes reuse possible. Most existing software component repositories only retrieve a limited set of Software components and some do not satisfy user queries. Interrelated software components may exist and would be useful, but the user either does not know about them or is unable to retrieve them because the query is defined too narrowly [2]. The schema of the repository itself often does not consider semantic relationships among software components and thus omits important component retrieval information. A technique to software component repositories is needed that provides the retrieval and recommendation of semantically interrelated software components. This paper presents ontology and faceted classification based meta-data repository and component repository for storage and retrieval of software components.

The method of faceted classification and retrieval is most extensive [2]. A term is putted into stated language context and is classified by specific angle of view (is called facet) which reflect essential characteristic of a software component in faceted classification [3][4], a facet is a basic characteristic which is described in a domain. A software component is classified by each facet from different profiles, a component can be described by many facets and many terms in a facet, different facet can describe a component from different angle of views. There are a set of terms in a facet, structured term space is formed by common and special relation. The value of a term can be only attained from given facet. It is helpful to understand correlative domain for the reused that travel in term space, the term space can be evolved. The method of faceted classification is most accurate to express information of a software component and can be easily understood by users in various methods of software component retrieval, therefore, if the method of faceted classification can be provided in some software component meta-data and component repositories which include many methods of software component retrieval, then it will achieve the best effect that the method of faceted classification is used [5]. But the type of software components and the requirement of organizations and user are different, the models of faceted classification are different too, in other words, the condition of retrieval for target software component is quite other, a user wish search appropriate software components from a component repository, the model of faceted classification must be understood and the condition of retrieval must be constructed, these manmade and subjective factors lead to the retrieval precision is low, when the main information of software component retrieval is provided, a user must make the most of generic terms or accepted terms. The metadata repository integrates expert knowledge of correlative domains and generalizes crucial concepts and relations among concepts in these domains [6] [7]. These query terms which are formed in virtue of metadata knowledge can improve the software component retrieval precision.

2. SOFTWARE COMPONENT STORING AND RETRIEVAL SYSTEM

The function of a software component retrieval storing system is that construct the model of software component retrieval, in the model, functions, applied domains, work environments, working , static and dynamic behaviors of a software component can be accurately expressed, the software component can be store, searched and reused [8]. A software component includes the entity, describing and metadata information in a software component repository. The three can be stored together or discretely. The discrete scheme is adopted so that reduce burthen, improve openness and is convenient for upgrade and maintenance, a component repository is divided into a describing repository and an entity repository. The software component retrieval system is based on meta-data, ontology faceted classification and adopts the model of three layers (view layer, application layer and data layer), the architecture is shown in Figure 1. The view layer is web form, the layer provides searching interfaces for software component users and library (repository) administration interfaces for administrators and knowledge experts. The application layer answer for describing component, classification, administration, feedback, authority and log, the layer realized by the view layer. There are four databases in data layer: a describing repository, component repository, a Meta data repository and ontology based component repository. The metadata repository stores information in special domains, provide accurate query terms, eliminate some phenomena such as same meanings with different names and same names with different meanings. According to describing facets, the describing repository can provide some information such as interfaces, functions, administrative levels, applied domains, developed languages, applied environments, editions and so on so that search software components[9]. The component repository store components and provide some services such as download and so on.

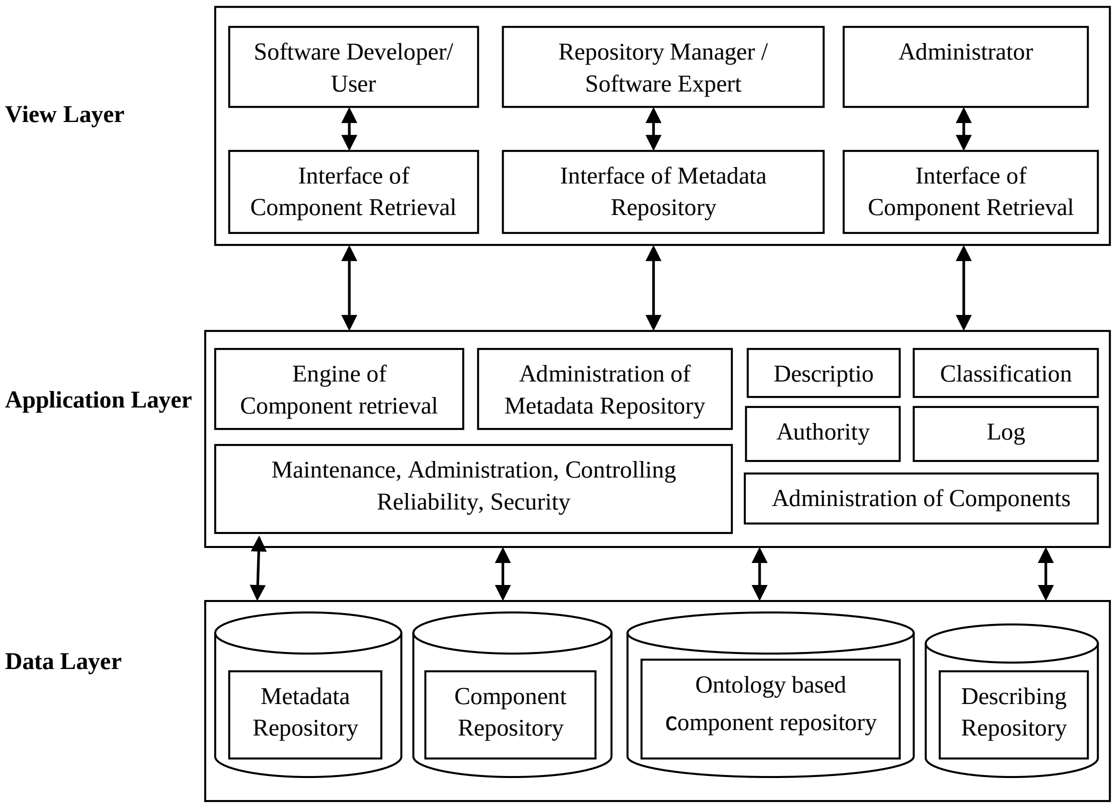
View Layer
Software Developer/ User
Repository Manager / Software Expert
Administrator
Interface of Component Retrieval
Interface of Metadata Repository
Interface of Component Retrieval
Application Layer
Engine of Component retrieval
Administration of Metadata Repository
Descriptio
Classification
Authority
Log
Maintenance, Administration, Controlling Reliability, Security
Administration of Components
Data Layer
Metadata Repository
Component Repository
Ontology based Component repository
Describing Repository

Fig 1: Software Component Storing and Retrieval System

3. ONTOLOGY –BASED META-MODEL FOR STORAGE AND RETRIEVAL OF SOFTWARE COMPONENTS

From an analysis of studies related to ontologies [36-41], it was observed that some ontology characteristics are suitable for software component retrieval, as they allow capturing domain semantics and recommending interrelated software components. Thus, ontology-based metadata were incorporated. The incorporated metadata were identified from elements belonging to the ontology creation language Web Ontology Language [39, 40] and were also based on the domain layer model of the ODEd ontology editor [37, 38]. Both ODEd and Web Ontology Language (OWL) support basic ontology elements and allow the definition of formal axioms that provide richer semantics to ontologies. The ontology principles that were considered relevant to storage and retrieval of software components were modeled. As the meta-model shows, a Domain has usual attributes, name and description, and also a modeling that graphically describes how the domain is organized according to the elements belonging to the meta-model. A domain is composed of Entities; metadata attributes which refer to the main concepts of the knowledge domain. In order to provide relationships with richer semantics, axioms that would contribute toward software component retrieval were investigated. In [38] [40] [41] a series of axioms are presented, some of which are considered relevant, namely, generalization/specialization, disjunction, inverse and whole-part associations. Thus, entities can have super-entities and sub-entities, and can be disjoint with other entities. Inverse associations (inverse of) indicate whether the relationship is bi-directional, allowing navigation in both directions. Whole part associations include axioms such as irreflection, ant symmetry and transitivity, and are classified as Aggregations (parts compose the whole, but not exclusively) and Compositions (parts exclusively compose the whole). Through these axioms, it is possible to present more information on the domain semantics and also infer knowledge in order to recommend interrelated software components. The captured domain information should be related to the software components through an analysis of their purposes and functionalities. Thus, it is possible to relate software components to correspondent associations and entities in domain semantics. Therefore, the elements belonging to the meta-model permit retrieving and recommending components based on the analysis of semantic information.

4. SOFTWARE COMPONENT RETRIEVAL PROCESS

The software component retrieval is implemented based on the architecture of the software component retrieval system that is shown in Figure 1. A user input query terms with the interface of software component retrieval, these terms match terms in the metadata repository, and the fittest describing terms are chosen to feed back (if these terms cannot strictly match terms in the meta data repository, the thesauruses are chosen from the metadata repository by a heuristic algorithm [10] [11]), these terms are further filtered and refined by users so that accurate query describing terms is formed. An accurate requirement of users is reflected to a describing repository of software component based on faceted classification by a module of accurate query processing, appropriate software components will be searched by a fixed retrieval algorithm; users filter appropriate software components and download from the component repository of component. The whole retrieval process is shown in Figure 2. The component retrieval model is based on Meta data, faceted classification and ontology. The module of accurate query processing is given by the server of the describing repository.

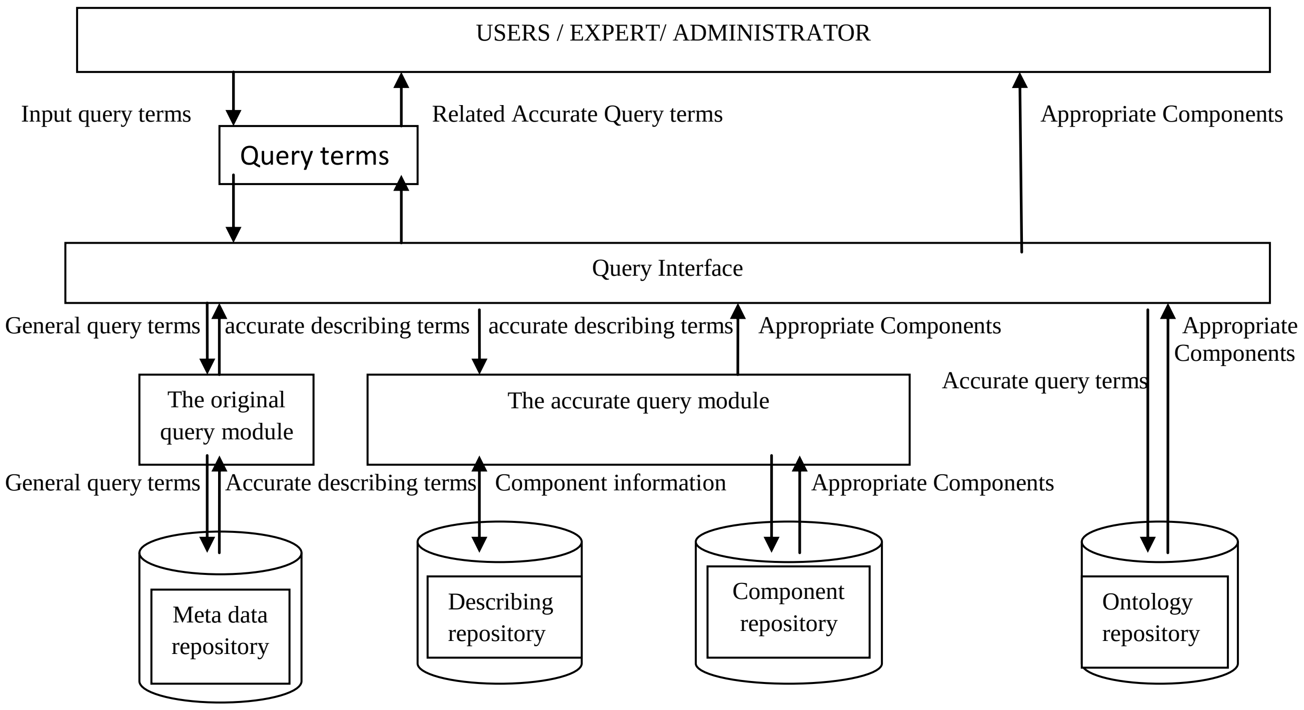

Fig 2: Software Component Retrieval Process

5. CONSTRUCTION OF AN METADATA REPOSITORY

The construction of a metadata repository comes down to two crucial problems: Meta knowledge representation and knowledge reasoning. Knowledge representation is chiefly solved; knowledge should be expressed by the method which a computer can understand, at the same time, and result should be told to users by the method which people can understand [12] [13].

5.1 The design of a repository

Much domain knowledge is stored in a repository; knowledge is expressed by rules which show implicit relations [14], these rules are defined as follows:
If *A* comes into existence, then *B* can be concluded, the confidence degree is *CL*, marked up *A* = (*B*, *CL*), hereinto, *A* is called antecedent which is combination of a series of conditions, i.e. $A = A1 \Lambda A2 \Lambda A3 \ldots\ldots\ldots \Lambda An$, is used to express preconditions; *B* is called consequent which express a conclusion. A Meta data repository includes relation as follows:

Table 1: A Metadata Repository includes Relation

Rule	
Rule_Number	Varchar (10)
Rule_Name	Long varchar
Antecedent	Long varchar
Consequent	Long varchar
Confidence	Float
Precondition	Long varchar
Category	Long varchar
Conclusion	Long varchar

When a expert system is used, new Meta data knowledge need be added, old meta data knowledge need be amended or deleted, for the sake of administration of various rules in an metadata repository, these rules need be classified and the above relation need be standardized. Finally, the whole metadata repository is composed of five relations tables.

Table 2: Precondition relation table

Rule_Precondition	
Rule_Number	Varchar(10)
Rule_Name	Long varchar
Precondition	Long varchar

Table 3: Antecedent relation table

Rule_Antecedent	
Antecedent_number	Varchar (10)
Antecedent_name	Long varchar
Antecedent_capacity	integer

Table 4: category relation table

Rule_category	
Category_number	Varchar (10)
Category_name	Long varchar
Rule_number	Varchar (10)
Rule_name	Long varchar

Table 5: Conclusion relation table

Rule_conclusion	
Rule_number	Varchar(10)
Rule_name	Long varchar
Antecedent_capacity	Integer
Consequent_name	Long varchar
Confidence	Float
Category_number	Varchar (10)
Category_name	Long varchar

Table 6: Consequent relation table

Rule_consequent	
Consequent_number	Varchar (10)
Consequent_name	Long varchar

In these relations, when the confidence degree is null, knowledge is full; the antecedent capacity is amount of conditions in precondition, its function will be explained in the design of inference engine, the content of Rule_Precondition and Rule_Conclusion expresses complete rules (i.e.knowledge) together.

5.2 The design of an inference engine

The design of an inference engine is directly concerned with the structure of a metadata repository, because the metadata repository is created with relational schema, the inference engine can be designed with SQL.

Definition: Set R*A*1 =Select Rule_Consequent. Consequent_name from Rule_Conclusion, Rule_Precondition, Rule_Consequent Where Rule_Precondition.Precondition= *A1* AND Rule_Precondition.Rule_number=Rule_Conclusion. Rule_number AND Rule_Conclusion.
Consequent_name =Rule_Consequent.Consequent_name, R*A*1 is called set based on *A1*.
The design of an inference engine is that gain set R*Ai* based on *Ai* (*i* =1,2,……. ,*n*) according to preconditions *A1* , *A2* ,….. , A*n* of rule *A*, the algorithm is designed as follows:

1. The precondition $A = A1\Lambda\ A2\Lambda \ldots\ldots.\Lambda An$ is put forward.
2. R*A*1 is solved, i.e. *S* =“Declare Cur Cursor for Select Rule_Consequent.Consequent_name from Rule_Conclusion.Rule_Precondition.Rule_Consequent Where Rule_Precondition.Precondition= *A1* AND Rule_Precondition.Rule_number=Rule_Conclusion. Rule_number AND Rule_Conclusion.Consequent_name= Rule_Consequent. Consequent_name”
3. The derivation sentence of $A2 \Lambda A3 \ldots\ldots. \Lambda An$ is constructed as follows:
   For *i* =2 to *n*
   S = *S* +“intersection Select Rule_Consequent. Consequent_name from Rule_Conclusion,Rule_Precondition,Rule_Consequent Where Rule_Precondition.Precondition= *Ai* AND Rule_Precondition.Rule_numbe=Rule_Conclusion .Rule_numbe AND Rule_Conclusion.Consequent_name= Rule_Consequent.Consequent_name” End For
4. The metadata repository is connected
5. The derivation sentence of *S* is executed

```
6. Fetch S Into A
7. Output A
8. Close S
```

In above algorithm, the operation of set is non backtracking, the whole derivation process can be rapidly completed by SQL, however, when the precondition such as " *A1 = B, A1 Λ A2 = C* " is included in an metadata repository, a problem will appear, therefore, the antecedent capacity is introduced into an Metadata repository, "AND Rule_Conclusion.Antecedent_capacity = *n"* is joined the clause "Where" of sentence "Select" so that solve the problem.

6. SOFTWARE COMPONENT RETRIEVAL AND MATCHING

In above software component retrieval process, the module of accurate query processing searches correlative software Components in term of the software component matching algorithm of component retrieval, above all, the facet describing of software component must be given, in order to describe static characteristic of a software component, e.g. applied domains, levels of development, functions, key facets (algorithm, languages, types and so on), applied environments and so on[15], at the same time, dynamic characteristics of a component should be described too [16], the different faceted classification corresponds to different sub-domain, the accuracy rate of formal specification should be improved, the layered and synthetically representation of facet is adopted[17][18], a component is regarded as twelve tuple, i.e. component=<function, applied domain, level, object, source object, middle object, interface, relationship, data type, core algorithm, language, applied environment>.

7. THE DESIGN OF A COMPONENT REPOSITORY

Software Components are stored in the form of component files. Associated to each component file, an index table is maintained. Some accurate Describing terms related to each component are also stored in the table [10].

Component files accurate describing terms showing functionality:

1. File1 ADT11, ADT12, ADT13, ADT14
2. File2 ADT21, ADT22, ADT23, ADT24, ADT25
3. File3 ADT31, ADT32, ADT33, ADT34, ADT35, ADT36
4. File4 ADT41, ADT42, ADT43, ADT44, ADT45
5. File5 ADT51, ADT52, ADT53, ADT54

……………………………………………………………..……
…………………………………………………………………
……………………………………………………………..……
…………………………………………………………………

For searching, a search function based on accurate Describing terms is used to retrieve the required component. Input to this function is a specification given by the users. Search function returns the component file name from the table. A link is established from returned file name to component file in the library. A user can decide by checking the name of the component file and select the component file by clicking on the link. The component file can be opened, checked for suitability, modified according to needs and can be saved by users at desired location.

All the components of the library are stored in the memory of the computer in folder. Name of component files and accurate Describing terms are stored in a two dimensional matrix known as index table. Accurate Describing terms of a component file are stored as a single multi word string. A delimiter can be used between two accurate Describing terms. In the below table, '*' is used as a delimiter. Corresponding to each component file, a counter is also used to keep account of accurate Describing terms matched. Initially value of each counter will be set zero.

Table 7: Repository Index Table

Sr_no #	Component File	Accurate Describing terms	Counter
1	File1	ADT11, ADT12, ADT13, ADT14	0
2	File2	ADT21, ADT22, ADT23, ADT24, ADT25	0
3	File3	ADT31, ADT32, ADT33, ADT34, ADT35, ADT36	0
4	File4	ADT41, ADT42, ADT43, ADT44, ADT45	0
5	File5	ADT51, ADT52, ADT53, ADT54	0
………. ………	……………. …………….	……………… ………………	0

7.1 Updating Software Component Library

A new software component can be added to the library by storing the component in the library, making its entry in the index table and establishing a link from index table to memory location in the library where it is actually stored. Similarly when a component is to be deleted from the library, it is removed from the physical memory along with its entry in index table and link from index table to memory location. Component can be stored anywhere in the library where free space is there. To make the insertion easy, entry of new component in index table is made at last position. This will not disturb the rest of entries in the index table and also not affect the efficiency as index table is searched linearly. But when an entry of a component is deleted from the index table, rest of the entries will have to be shifted one position above to avoid null row in the table.

Algorithm for Library Construction:

```
1. Set LIBRARY_SIZE=1000 and SR=0;
2. Declare parallel arrays
a. S_NO [LIBRARY_SIZE]; int array to store Sr#
b. STORED_COMPONENT [LIBRARY_SIZE]; string array to store file name
c. STORED_ACCURATE_DESCRIBING_TERMS [12]; string array to store accurate Describing terms associated with each component file
d. COUNTER [LIBRARY_SIZE]; int array to store counting of matches
3. Set CHARACTER='Y'
4. While (CHARACTER = = 'Y'), repeat steps from 5 to 12
5. Set SR=SR+1
6. Print 'Enter component file name'
7. Read STORED_COMPONENT [SR]
8. Print 'Enter accurate Describing terms '
```

```
9. Read STORED_ACCURATE_DESCRIBING_TERMS [SR]
10. Set COUNTER [SR] =0
11. Print 'Want to add another component? (Y/N)'
12. Read CHARACTER [End of step 4 loop]
13. Exit.
```

7.2 Component Search and Retrieval

A Component retrieval method [5, 10] can be described from three aspects: component representation, component query (user's requirements) specification, and component retrieval process. In this free-text-based retrieval method, components are represented as free-text-based documents, while a component query is described using accurate Describing terms. The retrieval process is to look up the accurate Describing terms in all component description documents. The components with most matched accurate Describing terms will be selected. Vector space and indexing technology are used to facilitate documents organizing and matching. This method has low scores on both precision and recall. Researchers and practitioners have proposed to use general thesaurus to extend accurate Describing terms, by including their synonyms and antonyms, to get more relevant component. In addition, general domain knowledge is also used to extend initial accurate Describing terms to get more semantically relevant components. However, both of these two improvements increase retrieval recall at the cost of retrieval precision. Multi word query entered by users is stored in string type array elements QUERY [1], QUERY [2] and so on. A list of common words like 'in", "on", "the", "of" etc. is stored at the time of library construction and these common words cannot become part of query.

These string type array elements are compared with accurate Describing terms of component files one by one. When QUERY[i] matches with any of the accurate Describing terms of a component file, value of its corresponding counter is incremented by 1. Fraction of match is also taken into consideration. It is possible by comparing QUERY[i] with accurate Describing terms character by character.

Let the number of character in QUERY [i] = y and number of characters matched with accurate Describing terms of particular component = x. Fraction of match (z) can be calculated as z = x / y. now the value of corresponding counter is incremented by z. First QUERY [1] is searched in the first row of accurate Describing terms. Then QUERY [2] is searched in this row of accurate Describing terms linearly and so on. After updating the value of first counter, the same procedure is applied on the second row and so on. Now the entire index table is sorted on counter column in descending order. This places the most relevant component file at first position with highest value of its counter, lesser relevant component at second place and so on. All the components with positive value of their counters are accessed and the components with zero value of their counters are discarded.

Algorithm for Searching:

```
1. Declare one dimensional array QUERY [ ] of suitable length to store the words of given query
2. Repeat for i=1 to n; n is the total number of components in the library Set COUNTER[i] = 0 [End of loop]
3. Set i=1
4. Print 'Enter your specification/ query'
5. Repeat step 6 and 7 while (Entered key =/= Return key)
6. Read QUERY[i]
7. i=i+1 [End of step 5 loop]
8. Set m=i-1; m is the number of accurate Describing terms entered by user
9. Repeat steps 10 for i=1 to n
10. Repeat step 11 to 14 for j=1 to m
11. Calculate the no. of characters in QUERY[j]; Let it be y
12. Compare QUERY[j] with STORED_ACCURATE_DESCRIBING_TERMS [i] character by character; Let no. of matched characters=X
13. Calculate float value Z = X /Y
14. COUNTER[i] = COUNTER[i] + Z [End of step 10 loop] [End of step 9 loop]
15. Sort the table on COUNTER column in descending order
16. Set i=1
17. Repeat step 18 and 19 while (COUNTER [i] =/= 0)
18. Print STORED_COMPONENT[i]
19. Calculate i = i + 1 [End of step 17 loops]
20. Exit.
```

Search mechanism described above is based on blind search. Efficiency of search mechanism can be improved by classifying the components into different categories. Required component can be searched into that particular category in spite of searching in the whole library. This approach will save time and improve efficiency of the search. Efficiency can also be improved by using fast sorting method for sorting the index table.

8. EXPERIMENT RESULTS

Precision: Precision is defined as the number of relevant components retrieved divided by the total number of components retrieved.

Precision = Number of relevant components retrieved / Total number of components retrieved

Recall: Recall is defined as the number of relevant components retrieved divided by the total number of relevant components in the index.

Recall = Number of relevant component retrieved / Total number of relevant components in the index

The Method of Retrieval	Components in Repository	Components Retrieved	Relevant Component Retrieved	Relevant Components in the Index	Precision	Recall
Traditional faceted retrieval	400	380	320	350	84%	91%
MDL File based retrieval	400	372	323	348	86%	92%
Metadata repository based retrieval	400	375	340	344	90%	97%
Metadata repository and ontology based component retrieval (proposed method)	400	392	375	380	96%	98%

Table 8: The Experiment results of software Components retrieval

Case 4: Metadata, Component and ontology repository based Search:
Total components in the repository = 400
Total number of components retrieved = 392
Total number of relevant components retrieved = 375
Total number of relevant components in the index =380
Precision =375 / 392 = 0.9566
Recall = 375/380 = 0.9868

- Attaining the precision of 0.9566 in Case 4 is considerably good which indicates that match is up to 96%.
- Recall value of 0.9868 indicates that we would have been able to retrieve 99% relevant components, in Case 4.

9. CONCLUSION

Software component reuse, as in other engineering disciplines, also evolved with fruitful results in case of software components reuse. The basic step in reusing already developed software artifacts is to build a library of such components. Such library is not just a collection of software artifacts but it is built with the objective in mind that the software components in such a library will be stored and retrieved for the purpose of software components reuse. Software components to be stored are developed such that these become more and more reusable. Making such a reuse library requires some different mechanism for storage and retrieval of software components. One such approach based on metadata, Component and Ontology repository searching was described in this paper with algorithm for building library and searching mechanisms. Fraction of match is also taken into consideration to make the retrieval mechanism more relevant. Combining software reuse with component and metadata is a new emerging trend in software development process. Combining these technologies helps the software development process by locating pre-existing software components at the design time only, due to which the total effort of software development is decreased.

10. REFERENCES

[1] K. Feiyui and W. Zhijian, "A Concept Model of Web Components", Proceedings of 2004 IEEE International Conference on Services Computing, pp.464-475, 2004.

[2] Y. Haining and L. Etzkorn, "Towards a semantic-based approach for software reusable component classification and retrieval", Proceedings of the 42nd Annual Southeast Regional Conference, New York: ACM, pp.110-115, 2004.

[3] F. Gibb, C. Mccartan and O. DonnellR, "The Integration of Information Retrieval Techniques within a Software Reuse Environment", Journal of Information Science, vol. 26, no. 4, pp.520- 539, 2000.

[4] W. Yuanfeng, Z. Yong and R.Hongmin, "Retrieving Components Based on Faceted Classification", Journal of Software, vol. 13, no. 8, pp.1546-1550, 2002.

[5] W. Yuanfeng, "Research on retrieving components classified in faceted schem", Fudan University, 2002.

[6] Lina and Z. Shijie, "Progress and prospects of expert system", Application Research of Computers, vol. 24, no. 12, pp.1-5, 2007.

[7] Z. Zipeng and L. Longshu, "Construct the expert system knowledge base with XML", Computer Technology and Development, vol. 17, no. 7, pp.31-34, 2007.

[8] D. Hemer, "Specification-based retrieval strategies for component architectures", Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05), pp.233-242, 2005.

[9] R. Giliane, S. Luciana and H. Peter, "A Reference Model for Reusable Components Description", Proceedings of the 38th Annual Hawaii International Conference on Systems Sciences, Los Alamitos: IEEE Computer Society, pp.282-283, 2005.

[10] H . Wang, Y. Feng and C. David, "Verifying the Reusability of Software Component Specification Framwork and Algorithms", Information Science, vol. 112, no. 12, pp.169-197, 1998.

[11] E.Gamma, R. Helm and P. Johnson, "Design Patterns: Elements of Reusable Object Oriented Software", Beijing: China Machine Press, 2002.

[12] Y. Wensheng, T. Pinghui and C. Xiuguo, "Problem Oriented Analysis and Decision Expert System with Large Capacity Knowledge-base", Proceedings of 2008 International Conference on Intelligent System and Knowledge Engineering, China, pp.32-37, 2008.

[13] Q. Yu and X. Li, "An expert system for real-time fault diagnosis of complex chemical processes", Expert Systems with Applications, vol. 24, no. 4, pp.425-432, 2006.

[14] P. Vitharana, F. Zahedi and H. Jain, "Knowledge-Based Repository Scheme for Storing and Retrieving Business Components: A Theoretical Design and an Empirical Analysis", IEEE Transactions on Software Engineering, vol. 29, no. 7, pp.649-664, 2003.

[15] Hiroyuki Kanazawa, Naoki Onishi, Yuri Mizusawa, Takahiro Tsunekawa, Hitohide Usami, "Application Hosting Services for Research Community on Multiple Grid Environments", JCIT: Journal of Convergence Information Technology, vol. 5, no. 4, pp.152-163, 2010.

[16] F.Yuku, "Dynamic Behavior Specification of Web Component Based on Logic Programming", Proceeding of the First International Multi-Symposiums on Computer and Computational Sciences, pp.480-482, 2006.

[17] Chuang Chih-Feng, Cheng Chao-Jen, "A study of institutional repository service quality and users' loyalty to college libraries in Taiwan: The mediating & moderating effects", JCIT: Journal of Convergence Information Technology, vol. 5, no. 8, pp.10, 2010.

[18] K. Wen, Research of Component Reuse in CAPP Domain, Nanjing University of Aeronautics and Astronautics, 2003.

[19] L. Xiaobo, M. Huaikou and L. Jing, "Components Matching Based on Formal Specifications", Computer Applications and Software, vol. 23, no. 10, pp.10-12, 2006.

[20] Guo, Jiang, "A pull-push combined architecture for federating information spaces", AISS: Advances in Information Sciences and Service Sciences, vol. 3, no. 3, pp.19-24, 2011.

[21] L.Yuhua, Z. Bandar and D. Mclean, “An approach for measuring semantic similarity between words using multiple information sources”, IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 4, pp.871-882, 2003.

[22] Yong-liu, Aiguang-yang; Research and Application of Software-reuse; Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/ Distributed Computing, IEEE, 2007, pp. 588-593.

[23] Prieto-Diaz, Ruben, Freeman, Peter; Classifying Software for Reuse; IEEE Software, 1987, vol. 4, no. 1, pp. 6-16.

[24] Haining Yao, Letha Etzkorn; Towards A Semanticbased Approach for Software Reusable Component Classification and Retrieval; ACM Software Engineering, 2004.

[25] Ostertag, Eduardo, Hendler, James, Prieto-Diaz, Ruben, Braun, Christine; Computing Similarity in a Reuse Library System an AI-based Approach; ACM Transaction on Software Engineering and Methodology, 1992, vol. 1, no. 3, pp. 205-228.

[26] Mili, Rym, Mili, Ali, Mittermeir, Roland T.; Storing and retrieving software components: a refinement based system; IEEE Transaction on Software Engineering, 1997, vol. 23, no. 7, pp. 445-460.

[27] Vitharana, Padmal, Zahedi, Fatemeh M., Jain, Hemant; Knowledge-based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis; IEEE Transactions on Software Engineering, 2003, vol. 29, no. 7, pp. 649-664.

[28] Sugumaran, Vijayan, Storey, Veda C.; A Semantic-Based Approach to Component Retrieval; The DATA BASE for Advances in Information Systems – Summer 2003, Vol. 34, No. 3, pp. 8-24.

[29] Rajender Nath, Harish Kumar; Building Software Reuse Library; 3rd International Conference on Advanced Computing and Communication Technology- ICACCT-08; Asia Pacific Institute of Information Technology, Panipat , India; November 08-09, 2008, pp. 585-587.

[30] Mili and Edward Addy, Reuse Based Software Engineering (A Wiley-Interscience Publication, John Wiley and Sons, Inc.2002) .

[31] Rajesh K Bhatia, Mayank Dave, R.C Joshi, “Retrieval of most relevant reusable Component using genetic algorithms”, Software Engineering Research and Practice 2006, 151-155.

[32] Rajesh K Bhatia, Mayank Dave, R.C Joshi, “A Hybrid Technique for Searching a Reusable Component from Software Libraries”, DESIDOC Bulletin of Information Technology, Vol.27, No.5, September 2007, pp. 27-34.

[33] Rajesh K Bhatia, Mayank Dave, R.C Joshi, “Ant Colony Based Rule Generation for Reusable Software Component Retrieval”, Proceedings of the 1st Conference on India Software Engineering Conference, pp 129-130, Feb 19-22, 2008, Hyderabad, India.

[34] Rajiv D. Banker, Robert J Kauffman and Dani Zweig, “Repository Evaluation of Software reuse”, IEEE Transactions on Software Engineering, Vol. 19, No 4, April 1993.

[35] Rym Mili, Ali Mili and R.T.Mittermeir, “Storing and Retrieving Software Components: A Refinement Based System”, IEEE Transactions on Software Engineering, Vol.23, No 7, July 1997.

[36] S. Araban, “A Two level Matching Mechanism for Object-Oriented Class libraries”, Ada-Europe 1998: Uppsala, Sweden, pp 188-200, no.1, Jan 1993. [10] Noy, N.F. and C.D. Hafner, The State of the Art in Ontology Design, AI Magazine. 1997. p. 53-74.

[37] Guarino, N. Formal Ontology and Information Systems. In International Conference on Formal Ontologies in Information Systems. 1998. Trento, Italy.

[38] Mian, P.G. and R.A. Falbo, Supporting Ontology Development with ODEd. Journal of the Brazilian Computer Society, 2003. 9(2): p. 57-76.

[39] Prieto-Díaz, R. A faceted approach to building ontologies. In IEEE International Conference on Information Reuse and Integration. 2003. Las Vegas, USA.

[40] Smith, M.K., C. Welty, and D.L. McGuiness, W3C Proposed Recomendation: OWL Web Ontology Language Guide. 2004. Available in <www.w3.org/TR/2004/RECowl- guide-20040210>. Accessed in May 2004.

[41] Staab, S. and A. Maedche. Ontology Engineering beyond the Modeling of Concepts and Relations. In ECAI'2000 Workshop on on Applications of Ontologies and Problem-Solving Methods. 2000. Berlin, Germany.