Reusable Software Component Retrieval System

Suresh Chand Gupta¹, Prof. Ashok Kumar²

¹PIET, Samalkha, Panipat
²MMICT & BM, MMU, Mullana

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 taxonomies. 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.

Keywords: Metadata repository, Search Engine, faceted classification, component model, heuristic algorithm, ontology, Software reuse, software component, Metadata, component retrieval, Component based engineering

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 reduceburthen, 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.

![Software Component Storing and Retrieval System](image)
3. ONTOLOGY–BASED META-MODEL FOR STORAGE AND RETRIEVAL OF COMPONENTS

From an analysis of studies related to ontologies [10-15], 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 (OWL) [14] and were also based on the domain layer model of the ODEd ontology editor [12]. Both ODEd and 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. This modeling can be graphically represented by an UML Class Diagram. A domain is composed of Entities, which refer to the main concepts of the knowledge domain. In order to provide relationships with richer semantics, axioms that would contribute toward component retrieval were investigated. In [12] [14] [15] 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 expert 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](image-url)
5. CONSTRUCTION OF AN EXPERT REPOSITORY

The construction of an expert 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. $A = A1 A2 A3 ......... An$, 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

<table>
<thead>
<tr>
<th>Rule</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule_Number</td>
<td>Varchar (10)</td>
</tr>
<tr>
<td>Rule_Name</td>
<td>Long varchar</td>
</tr>
<tr>
<td>Antecedent</td>
<td>Long varchar</td>
</tr>
<tr>
<td>Consequent</td>
<td>Long varchar</td>
</tr>
<tr>
<td>Confidence</td>
<td>Float</td>
</tr>
<tr>
<td>Precondition</td>
<td>Long varchar</td>
</tr>
<tr>
<td>Category</td>
<td>Long varchar</td>
</tr>
<tr>
<td>Conclusion</td>
<td>Long varchar</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Rule_Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule_Number</td>
</tr>
<tr>
<td>Rule_Name</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
</tbody>
</table>

Table 3: Antecedent relation table

<table>
<thead>
<tr>
<th>Rule_Antecedent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antecedent_number</td>
</tr>
<tr>
<td>Antecedent_name</td>
</tr>
<tr>
<td>Antecedent_capacity</td>
</tr>
</tbody>
</table>

Table 4: category relation table

<table>
<thead>
<tr>
<th>Rule_category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category_number</td>
</tr>
<tr>
<td>Category_name</td>
</tr>
<tr>
<td>Rule_number</td>
</tr>
<tr>
<td>Rule_name</td>
</tr>
</tbody>
</table>

Table 5: Conclusion relation table

<table>
<thead>
<tr>
<th>Rule_conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule_number</td>
</tr>
<tr>
<td>Rule_name</td>
</tr>
<tr>
<td>Antecedent_capacity</td>
</tr>
<tr>
<td>Consequent_name</td>
</tr>
<tr>
<td>Confidence</td>
</tr>
<tr>
<td>Category_number</td>
</tr>
<tr>
<td>Category_name</td>
</tr>
</tbody>
</table>
In these relations, when the confidence degree is null, knowledge is full; the antecedent capacity is amount of conditions in preconditions, 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 RA1 = \(\text{Select Rule_Consequent. Consequent_name from Rule_Conclusion, Rule_Precondition, Rule_Consequent Where Rule_Precondition.Precondition= 1 A AND Rule_Precondition.Rule_number=Rule_Conclusion.Rule_number AND Rule_Conclusion.Consequent_name=Rule_Consequent. Consequent_name}\). RA1 is called set based on A1. The design of an inference engine is that gain set RAi based on A1, A2, ……, An of rule A, the algorithm is designed as follows:

1. The precondition A = A1, A2, ……, An is put forward.
3. The derivation sentence of A1, A2, ……, An is constructed as follows:
   
   ```
   For i = 2 to n
   ```
4. The expert 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 ^ A2 = C” is included in a metadata repository, a problem will appear, therefore, the antecedent capacity is introduced into a 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 Components in term of the software component matching algorithm of component retrieval, above all, the facet describing of 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 ten tuple, i.e. component=<function, applied domain, level, object, source object, middle object, interface, core algorithm, language, applied environment>.

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

\[
\text{Precision} = \frac{\text{Number of relevant components retrieved}}{\text{Total number of components retrieved}}
\]

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

\[
\text{Recall} = \frac{\text{Number of relevant component retrieved}}{\text{Total number of relevant components in the index}}
\]
Case 1: Metadata 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 1 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 1.

All the experiment results are shown in Table 7. The results that are searched by the method of traditional faceted retrieval and the above method are shown in Table 7.

**Table 7: The Experiment results of software Components retrieval**

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

8. CONCLUSION

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 components in such a library will be stored and retrieved for the purpose of reuse. 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 file 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 components at the design time only, due to which the total effort of software development is decreased.

REFERENCES

[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.