Facets of Software Component Repository

Vaneet Kaur
Computer Science and Engineering Department
Thapar University
Patiala, India
er.vaneetkaur@gmail.com

Shivani Goel
Computer Science and Engineering Department
Thapar University
Patiala, India
shivani@thapar.edu

***Abstract*-The software repository is used for storing, managing, and retrieving large numbers of software components. Repositories should be designed to meet the growing and changing needs of the software development organizations. Storage and representation of reusable software components in software repositories to assist retrieval is a key concern area. In this paper we have discussed various assets of the component repository like component searching mechanisms and classifications such as Free Text, Enumerated, Attribute Value, and Faceted classifications. Developers and end users can formulate high-level, aspect-based queries to retrieve components according to their needs.**

***Keywords*-Repository, Retrieval, Component, Queries**

I. INTRODUCTION

The component-based software engineering (CBSE) or component-based development (CBD) emphasizes the development of applications based on components so that the applications are easy to maintain, and extend.
We say component is an independent and self-sufficient part of a system having complete functionalities. The main idea of using component-based development is reusability. Software reuse refers to use of software modules that were developed on a earlier software projects as part of a new software development project. Software reuse is a worthwhile goal because it is used to reduce software costs and improve software quality as well as programmer productivity [5]. Also, the move toward component-based software development, which is a systematic method for using reusable components from in-house component repositories and from commercial component providers, has recently, influenced interest in software reuse research. A component-based system highlights appropriate reuse and composition of software components as its key concept. However, finding and reusing appropriate software components is often very challenging, particularly when faced with a large collection of components and little documentation about how they can and should be used[4].

II. COMPONENT REPOSITORY

An independent information repository system (or component repository system) can be abstractly thought of as secondary information storage from the perspective of computer users because information stored in this is accessible only after users have stopped working on their current tasks and switched from their workspaces. The retrieval process finds the components that match given reuse queries. An effective retrieval mechanism including a representation schema for indexing and a matching criterion between a query and a component is essential. The structure of a repository is generally regarded as key to obtaining good retrieval results. No matter how
"Intelligent" the matching algorithm, if components are indexed or otherwise structured poorly, it will be difficult to achieve good retrieval performance [6].

III. STORAGE AND RETRIEVAL OF SOFTWARE COMPONENT

Reusability is a process of utilizing and applying already developed components. So there are many work products that can be reused, for example source code, designs, specifications, architectures and documentation. Successful reuse requires having a wide variety of high quality components, proper classification and retrieval mechanisms.
Effective software reuse requires that the users of the system have access to appropriate components. Retrieval should allow users to formulate high-level queries about component capabilities and takes account of the context

in which a query is performed to assist query formulation [4]. The user must access these components accurately and quickly, and if necessary, be able to modify them. Classifying software allows reusers to organize collections of components into structures so that they can be searched easily. Most retrieval methods require some kind of classification of the components. Four different classification techniques had been previously employed to construct reuse repository namely Free Text, Enumerated, Attribute Value, and Faceted classifications.

A. Free text classification

The retrieval system of free text classification is typically based upon a keyword search. In this type of searching technique, a user inputs keywords to search and as a result a ranked list of documents is returned. We can see a number of search engines using keyword search technique to search documents on Internet. As we know search engines like Google, Yahoo, MSN search, and AltaVista are some well-known web search engines that support keyword-based searching as a basic searching technique to search documents and websites etc. Free-text classification is also referred to as uncontrolled vocabulary, consists in analyzing word frequencies in natural text [3]. Relevant keywords are derived automatically by their statistical and positional properties, thus resulting in what is called automatic indexing.

Two processes are involved in keyword search that is indexing and searching. The indexing process looks into all the available documents in different repositories or databases and creates a list of search items, which could be used for search purpose.

Keyword search gives freedom to users to freely submit any query to search engines; however this freedom may rise following two problems as:

- Recognize keywords that best explain the need of users.
- Possible ways in which a user may search.

Another problem is that it may result in irrevalent components.

B. Enumerated Classification

Enumerated classification is a single dimension classification which uses a set of mutually exclusive classes. An example of this is the Dewey Decimal system used to classify books in a library [2]. Each subject area, e.g. Physics, 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.

Major problem is that this type of classification schemes as is one dimensional will not allow flexible classification of components into more than one place for reusable software components. It does however; provide substantial support for best effort retrieval of components.

C. Attribute Value

This type of classification scheme uses a set of attributes to classify a component .For example, a book has many attributes such as the author, the publisher, title and a unique ISBN number and classification code in the Dewey Decimal system[1]. Then we see the requirement and on that basis the attributes could be concerned with the number of pages, the size of the paper used and the publishing date etc. The attributes related to a book can be:

- Multidimensional. The book can be classified in different places using different attributes.
- Bulky. All possible variations of attributes which may not be known at the time of classification[2].

D. Faceted

Faceted classification so known as faceted navigation or faceted browsing was proposed by Prieto-Diaz and Freeman in 1987[3] that relies on facets which are extracted by experts to describe features about components. Features serve as component descriptors, such as the component's functionality, how to run the component, and implementation details .Like the attribute classification method, various facets classify components however there are usually a lot fewer facets than there are potential attributes. Sometimes faceted search is also referred to explorative search and guided search, because users are given choice to select features available for search. This helps the users to accomplish his search goals rapidly and efficiently.

Faceted classification and retrieval has proven to be very effective in retrieving reuse component from repositories, but the approach is labor intensive.

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 [1].

Each of the facets has to have values assigned at the time the component is classified.

IV. QUERYING AND BROWSING COMPONENT

Querying and browsing are the two major information access mechanisms for most users. Querying is direct that users formulate a query and the system returns information matching the query. However to formulate a query is a quiet difficult task because users have to overcome the gap from the situational model to the system model. In browsing, users determine the usefulness or relevance of the information currently being displayed in terms of their task and traverse its associated links. Mili et al.[8] claims that browsing is the most predominant pattern of component repository usage because most programmers often cannot formulate clearly-defined requirements for reusable components so they rely on browsing to get familiar with available reusable components in the repository.

V. EFFECTIVE RETRIEVAL MECHANISM IN RETRIEVING

Several retrieval techniques are produced till now. There are three major approaches in retrieval mechanisms text-based, descriptor-based and formal specification-based. In text-based approach a component is represented by their textual documents and information retrieval technology is used to match components to queries. In descriptor-based approach, components are represented by a set of selected descriptors [7]. The semantic relationships among those descriptors are captured in a predetermined structure that can be specified by a semantic network. In specification-based approach, components are represented with formal specification languages, and specification refinement systems are used to determine whether a component matches a query written in formal specification languages or not. Retrieval mechanisms play an important role in locating reusable components that match reuse queries.

VI. EVALUATION

To check whether component repository works effectively we evaluate the correctness and effectiveness so the recall and precision measures.

Precision and recall are two concepts that have traditionally been used to evaluate the method of retrieval software components.

Recall means to get all the relevant components. Precision means that all the retrieved components are exact as per query submitted by a user.

$$\text{RECALL} = \frac{\text{Ratio of relevant retrieved assets}}{\text{Total number of relevant assets in the library}}$$

This is the number that ranges between 0 and1.Under the hypothesis that all the library assets are visited, we get perfect recall (=1) whenever the relevance criterion logically implies the matching condition. This can be achieved in particular by letting the matching condition be true which means that all library assets are returned.

$$\text{PRECISION} = \frac{\text{Ratio of relevant retrieved assets}}{\text{Total number of retrieved assets}}$$

This is a number that ranges between 0 and 1.Under the hypothesis that all the library assets are visited, we get perfect precision (=1) whenever the matching condition logically implies the relevance criterion. This can be achieved in particular by letting the matching condition be false which means no assets are returned.

It is very difficult to get both high precision and high recall using the classical approach. For effective use, the keywords have to be independent, they have to span the range of the users need, and the users must be able to pick the right keywords.

VII. SUMMARIES

There are various shortcomings of existing approaches that include the need to use various queries like low-level, service-based queries, lack of high-level description of component capabilities, lack of validation or checking of retrieved component suitability, and lack of use of the context for which queries are being performed by the retrieval tool. Formal specification or execution based retrieval mechanisms repositories generally bear from a need to thoroughly, formally specify parts of component services, with queries requiring formal specification techniques that may be difficult for many end users and developers to use. Classification

that is keyword and facet-based search are efficient to search and retrieve components. Enumerated classification is the Fast method but is difficult to enlarge. Faceted classification can be easily enlarged and is most flexible. Free text classification is uncertain and consists of indexing costs. Attribute value classification is slowest method and has no ordering.

Future work involved with this classification scheme will be to refine the scheme for Multi-Tired or Multimedia presentation of components. Another critical aim of the repository system is to build capabilities to support concept generation. Also various ways of optimizing system speed and efficiency must be explored in order to keep the repository as an effective design tool.

REFERENCES

[1] P.Niranjan, Dr. C.V.Guru Rao 'A mock- up tool for software component reuse repository', International journal of software engineering and applications (IJSEA), Vol.1, No.2, April 2010.

[2] E. Smith, A.Al-Yasiri and M. Merabti, 'A multitiered classification scheme for component retrieval'. Proc. 24th Euro micro Conf., 1998, pp. 882–889.

[3] Ruben Prieto-Diaz Software Production Consortium, Herndon, VA,' Implementing faceted classification for software reuse', ACM Press New York, NY, USA; Pages: 88 – 97: 1991.

[4] J.C. Grundy, 'Storage and retrieval of Software Components using Aspects', in Proc. of the 2000 Australasian Computer Science Conference, Canberra, Australia ,IEEE CS Press, pp 95-103.

[5] H. Yao and L. Etzkorn: 2004, 'Towards a semantic-based approach for software reusable component classification and retrieval'. In: Proceedings of the 42nd annual southeast regional conference. pp. 110–115, ACM Press.

[6] S. Henniger, (1996), 'Supporting the construction and evolution of component repositories', Proceedings of the 18th International Conference on Software Engineering (ICSE'96), Berlin, Germany.

[7] W.B. Frakes, & T.P. Pole, (1994), 'An Empirical Study of Representation Methods for Reusable Software Components', IEEE Transactions on Software Engineering 20(8), 617–630.

[8] A. Mili, S. Yacoub, E. Addy, & M. Hafedh, (1999), 'Toward an Engineering Discipline of Software Reuse', IEEE Software 16(5), 22–31.