Implementation of attribute value & faceted value classification scheme for constructing Reuse Repository

Gowtham Gajala [#1]

[#]*Assistant Professor, Department of Information Technology*
Kakatiya Institute of Technology & Science,
Warangal, Andhra Pradesh, INDIA - 506015,

***Abstract*— Software reuse demands that existing components must be readily incorporated into new products. To be able to reuse software components, it is necessary to locate the component that can be reused. Locating components, or even realizing that they exist, can be quite difficult in a large collection of components. These components need to be suitably classified and stored in a repository to enable efficient retrieval. Four schemes have been previously employed, Free Text, Enumerated, Attribute Value and Faceted classification. But we try to develop only with two schemes which are attribute value scheme and faceted value scheme. This paper titled "Implementation of attribute value and faceted value classification for constructing reuse repository" is aimed to develop a classification scheme which uses an attribute value and faceted value of existing classification that can be used by people to retrieve the reusable components. This approach serves as an effective means to categorize components and to retrieve the relevant components efficiently to improve retrieval efficiency.**

***Keywords*— Software reuse, repository, attribute, classification techniques.**

I. INTRODUCTION

Software is rarely built completely from scratch. To a great extent, existing software documents (source code, design documents, etc.) are copied and adapted to fit new requirements. Yet we are far from the goal of making reuse the standard approach to software development.

Software reuse is the process of creating software systems from existing software rather than building them from scratch. Software reuse is still an emerging discipline. It appears in many different forms from ad-hoc reuse to systematic reuse, and from white-box reuse to black-box reuse. Many different products for reuse range from ideas and algorithms to any documents that are created during the software life cycle. Source code is most commonly reused; thus many people misconceive software reuse as the reuse of source code alone. Recently source code and design reuse have become popular with (object-oriented) class libraries, application frameworks, and design patterns.

Software components provide a vehicle for planned and systematic reuse. The software community does not yet agree on what a software component is exactly. Nowadays, the term component is used as a synonym for object most of the time, but it also stands for module or function. Recently the term component-based or component oriented software development has become popular. In this context components are defined as objects plus some-thing. What something is exactly, or has to be for effective software development, remains yet to be seen. However, systems and models are emerging to support that notion.

Systematic software reuse and the reuse of components influence almost the whole software engineering process (independent of what a component is). Software process models were developed to provide guidance in the creation of high-quality software systems by teams at predictable costs. The original models were based on the (mis)conception that systems are built from scratch according to stable requirements. Software process models have been adapted since based on experience, and several changes and improvements have been suggested since the classic waterfall model. With increasing reuse of software, new models for software engineering are emerging. New models are based on systematic reuse of well-defined components that have been developed in various projects.

Developing software with reuse requires planning for reuse, developing for reuse and with reuse, and providing documentation for reuse. The priority of documentation in software projects has traditionally been low. However, proper documentation is a necessity for the systematic reuse of components. If we continue to neglect documentation we will not be able to increase productivity through the reuse of components. Detailed information about components is indispensable.

The ability to develop new applications (In particular Web-based applications) in a short time is crucial to the success of software companies that need to compete aggressively in today's market. Considering the fact that software technologies emerge very fast, change on a daily basis, this becomes an even more complicated task. For this reason it is vital to share and reuse the knowledge and the programming experiences in an efficient and productive manner.

Software is rarely built completely from scratch. To a great extent, existing software documents (source code, design documents, etc.) are copied and adapted to fit new requirements. Software reuse is an important area of software

engineering research that promises significant improvements in software productivity and quality .Software reuse is the use of existing software or software knowledge to construct new software. It is of interest because people want to build systems that are bigger and more complex, more reliable, less expensive and that are delivered on time. They have found traditional software engineering methods inadequate, and feel that software reuse can provide a better way of doing software engineering.

There are two basic technical approaches to reuse: Parts-based and Formal language-based. The parts-based approach assumes a human programmer integrating software parts into an application by hand. In the formal language-based approach, domain knowledge is encoded into an application generator or a programming language. The research reported here focuses on the parts-based approach.

Component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard. General examples of concrete components include interface, computational, memory, manager, controller components and Web services. Components may come from many domains, in many languages and design notations. Also versions of components may also exist. Due to this large number of components, we think that a component management system is needed in order to keep track of the properties of all the components which are available.

To incorporate reusable components into systems, programmers must be able to find and understand them. If this process fails, then reuse cannot happen. Thus, how to index and represent these components so that they can be found and understood are two important issues in creating a reuse tool. Classifying software allows re-users to organize collections of components into structures that they can search easily.

II. Existing Techniques

A. 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. The major drawback with this method is the ambiguous nature of the keywords used. Another disadvantage is that a search my result in many irrelevant components. A typical example of free text retrieval is the 'grep' utility used by the UNIX manual system. 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 are index, which must then be searched from beginning to end when a query is made.

B. 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 is the Dewey Decimal system used to classify books in a library. Each subject area, for example, 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.

C. Attribute value

The attribute value classification scheme uses a set of attributes to classify a component [6]. For example, a book has many attributes such as the author, the publisher, a unique ISBN number and 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:

1) Multidimensional. The book can be classified in different places using different attributes.
2) Bulky. All possible variations of attributes could run into many tens, which may not be known at the time of classification.

Each attribute has the same weighting as the rest, the implications being that it is very difficult to determine how close a retrieved component is to the intended requirements, without visually inspecting the contents.

D. 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 (at most, 7). Ruben Prieto-Diaz has proposed a faceted scheme that uses six facets. He proposed three functional and three environmental facets.

1) The Functional Facets are: Function, Objects, and Medium.
2) 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 seen 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. The investigation found no statistical evidence of any differences between the four different classification schemes; however, the following about each classification method was noted:

- 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.

III. Proposed System

Existing software components for reuse can be directly classified in the classification scheme into one among the above specified classifications presented in the previous section and stored in the reuse repository. Sometimes they need to be adapted according to the requirements. As classification scheme relies on one of the techniques discussed in the previous section which shall inherently affect the classification efficiency. New designs of software components for reuse are also subject to classified to classification scheme before storing them in the reuse repository. User will retrieve his desired component with required attributes from reuse repositories. The architecture of proposed system is shown in the figure1.

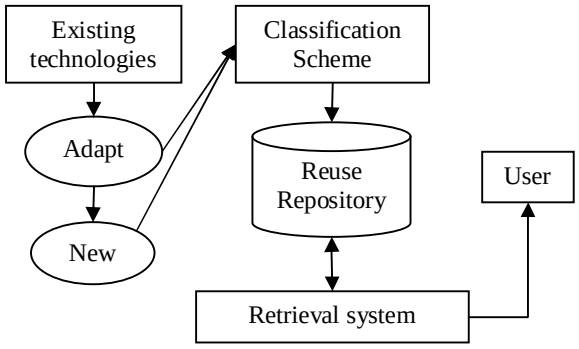

Fig.1 Proposed System Architecture

A. Reuse Environment

Reuse environment should include the following elements.

- User: A registered user who want to reuse the components.
- Library or Repository: Is capable of storing software components and classification information to allow their retrieval.
- Retrieval System: Enables client software to retrieve components and services from library server.
- Matching Details: Is a mechanism carried out in an effective search for the components.

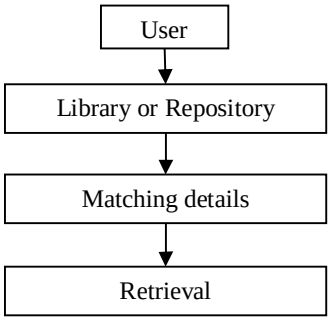

Fig. 2 Reuse environment

B. Component Classification

The generic term for a passive reusable software item is a component. Components can consist of, but are not restricted to ideas, designs, source code, linkable libraries and testing strategies. The developer needs to specify what components or type of components they require.

These components then need to be retrieved from a library, assessed as to their suitability, and modified if required. Once the developer is satisfied that they have retrieved a suitable component, it can then be added to the current project under development. The aim of a 'good' component retrieval system is to be able to locate either the exact component required, or the closest match, in the shortest amount of time, using a suitable query. The retrieved component(s) should then be available for examination and possible selection.

An integrated classification scheme, which employs a combination of one or more classification techniques, is proposed and likely to enhance the classification efficiency. The proposal is described in the following sub section. This had given rise to development of a software tool to classify a software component and build reuse repository.

Integrated classification scheme which combines the attribute value and faceted classification schemes to classify components with the following attributes.

- Operating system
- Language
- Keywords
- Inputs
- Outputs
- Domain
- Version
- Category

The attributes when used in query can narrow down the search space to be used while retrieval.

The proposed software tool will provide an user friendly interface for browsing, retrieving and inserting components. Two algorithms are proposed for searching and inserting components as part of this software tool.

C. Algorithm 1: Component Insert (Component facet and attributes)

Purpose: This algorithm inserts a component into the reuse repository with integrated classification scheme attributes.

Input: Component facet and attributes

Output: Component insertion is success or failure.

Variables: rrp: reuse repository array,
rp: repository pointer,
flag : boolean

```
if((rrp[i].lang<>lan) and rrp[i].fun>fun) and (rrp[i].dom<>dom) and (rrp[i].os<>os) and (rrp[i].ip<>ip) and (rrp[i].op<>op) and (rrp[i].ver<>ver))
        i++;
else
        flag = true;
        break;
if (flag)
        rrp[rp].lang = lan;
        rrp[rp].fun = fun;
        rrp[rp].os = os;
        rrp[rp].dom = dom;
        rrp[rp].ip = ip;
        rrp[rp].op = op;
        rrp[rp].ver = ver;
        return successful insertion;
else
        Component is already exists;
```

The insert algorithm stores the newly designed or adapted existing component into the reuse repository. When component attributes are compared with existing repository component attributes and determines no similar components are found then component is inserted successfully otherwise component not inserted in repository and exits giving message that component already exists.

D. Algorithm 2: Search Component (Component facet and attributes)

Purpose: This algorithm searches for relevant components with given component facet and attributes from reuse repository.

Input: Component facet and Component attributes.

Output: list of relevant components

Variables: rrp: reuse repository array
rp: repository pointer
table: result array
i.j : internal variables
flag: boolean

```
if (component facet <> null )
for ( i=1; i <= rp ; i++ )
if ((rrp[i].language = lan ) and (rrp[i].function = fun ))
        table[j].lang = rrp[j].lang
        table[j].fun = rrp[j].fun
        table[j].os = rrp[j].os
        table[j].ip = rrp[j].ip
        table[j].op = rrp[j].op
        j++;
else
        flag = 0;
if (component facet<>null) and (any of the other attributes<> null )
for (i =1;i <= rp ;i++ )
if ((rrp[i].lang = lan) and (rrp[i].fun = fun))
if((rrp[i].os = os) or (rrp[i].ip = ip) or (rrp[i].op = op) or rrp[i].dom = dom) or (rrp[i].ver = ver))
        table[j].lang = rrp[i].lang;
        table[j].fun = rrp[i].fun;
        table[j].os = rrp[i].os;
        table[j].dom = rrp[i].dom;
        table[j].ip = rrp[i].ip;
        table[j].op = rrp[i].op;
        table[j].ver = rrp[i].ver;
if(!flag )
        No component is matched with given attributes.
```

IV. RESULTS

The search algorithm accepts component facet and attribute values from user and retrieves relevant components from reuse repository.

The proposed software tool is developed by implementing the following modules.

1) User Interface: The user must be able to insert and search the components in the reuse repository. A user friendly interface is designed to select relevant attributes.
2) Query Formation: The user when desirous of searching a component may enter some keywords. He may also select some list of attributes from the interface. The query formation module should accept all the keywords entered and form the query using those keywords.
3) Query Execution: When user sends a query to retrieve component by query execution on all the components which satisfy the criteria that is specified by user in advanced search of user interface.
4) Formatting Results and Presentation: The results obtained in the previous module are formatted so that the user can clearly understand the functionality of component before choosing one.

The search performance is evaluated with different test results and compared with existing schemes.

Search effectiveness refers to how well a given method supports finding relevant items in given database. This may be number of relevant items retrieved over the total number of items retrieved.

Faceted classification scheme marked highest performance of search among all the existing classification schemes. Keyword classification scheme registered the lowest performance. Whereas our proposed integrated classification scheme out performed to retrieve more relevant items in comparison to all those existing schemes.

V. CONCLUSION AND FUTURE SCOPE

This paper "Implementation of attribute value & faceted value classification scheme for constructing reuse repository" was deeply studied and analyzed to design the code and implement with various testing methods was done. The solution developed is free from all the bugs and executable with all different modules to the utmost satisfaction of the client.

The main criteria are whenever an admin or a registered user uploads a component into repository, an auto generated mail is sent to the admin email so that the admin can update the (temp) repository along with validation of uploaded component and to apply a multimedia affect like audio output for the searched components.

In addition to the retrieval of relevant component, auto generation of mail and also multimedia effect like audio output, we can still work on applying more multimedia effects like adding video output for the searched output so as to make the registered user more comfortable in selecting and downloading the searched component.

ACKNOWLEDGMENT

This is my pleasure to express my deep sense of profound gratitude to my guide Sri P. Niranjan Reddy, Head of Department of Computer Science & Engineering, Kakatiya Institute of Technology & Science, Warangal. Under his guidance and supervision this work has been accomplished. His keen interest, constant inspiration and constructive criticism have been of great help to me.

REFERENCES

[1] S.Henninger, "An Evolutionary Approach to Constructing Effective Software Reuse Repositories", ACM Transactions on Software Engineering Methodology, no 2, 1997, pp. 111-150

[2] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", Communication of the ACM, Vol. 34, No.5, May 1991

[3] Gerald Kotonya, Ian Sommerville and Steve Hall, "Towards A Classification Model for Component Based Software Engineering Research", Proceeding of the of the 29th EUROMICRO Conference © 2003 IEEE

[4] William B. Frakes and Thomas. P.Pole, "An Empirical Study of Representation Methods for Reusable Software Components", IEEE Transactions on Software Engineering vol.20, no.8, Aug. 1994, pp.617-630.

[5] Lars Sivert Sorumgard Guttorm Sindre and Frode Stokke, "Experiences from Application of a Faceted Classification Scheme" © 1993 IEEE, pp 116-124.

[6] Jeffrey S. Poulin and Kathryn P.Yglesias "Experiences with a faceted Classification Scheme in a Large Reusable Software Library (RSL)", In The Seventh Annual International Computer Software and Applications Conference (COMPSAC'93), 1993, pp.90-99

[7] Vicente Ferreira de Lucena Jr., "Facet-Based Classification Scheme for Industrial Automation Software Components"

[8] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse" © 1990 IEEE, pp.300-304

[9] Klement J. Fellner and Klaus Turowski, "Classification Framework for Business Components", Proceedings of the 33 rd Hawaii International conference on system Sciences- 2000, 0-7695-0493-0/00 © 2000 IEEE

[10] Vitharana, Fatemeh, 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.

[11] William B.Frakes and Kyo Knag, "Software Reuse Research: Status and Future", IEEE transactions on Software Engineering, VOL.31 NO.7, JULY 2005

[12] R.Prieto-Diaz and P.Freeman, "Classifying Software for Reuse", IEEE Software, 1987, Vol.4, No.1, pp.6-16.

[13] Rym Mili, Ali Mili, and Roland T.Mittermeir, "Storing and Retrieving Software Components a Refinement Based System", IEEE Transactions of Software Engineering, 1997, Vol.23, No.7, pp. 445-460

[14] Hafedh Mili, Estelle Ah-Ki, Robert Godin, and Hamid Mcheick, "Another nail to the coffin of faceted controlled vocabulary component classification and retrieval", Proceedings of the 1997 symposium on software reusability (SSR'97), May 1997, Boston USA, pp.89-98.

[15] Hafedh Mili, Fatma Mili, and Ali Mili, "Reusing Software: Issues and Research Directions", IEEE Transactions on Software Engineering, Vol.21 No.6, June 1995.

[16] Gerald Jones and Ruben Prieto-Diaz, "Building and Managing Software Libraries",© 1998 IEEE, pp.228-236.

[17] Prieto-Diaz, Freeman, "Classifying Software for Reuse", IEEE Software, vol.4, mo.1, pp.6-16, 1997

[18] Nancy G. Leveson, Kathryn Anne Weis, "Making Embedded Software Reuse Practical and Safe "12 th ACM SIGSOFT, October, 2004.

[19] William B. Frakes and Kyo Kang, "Software Reuse Research Status and Future" IEEE transactions on Software Engineering, Vol. 31, No.7, July 2005.

[20] William. B. Frakes and Thomas. P. Pole, "An Empirical Study of Representation Methods for Reusable Software Components", IEEE Transactions on Software Engineering vol. 20, no. 8, Aug. 1994, pp. 617–630.

[21] Lars Sivert Sorumgard, Guttorm Sindre and Frode Stokke, "Experiences from Application of a Faceted Classification Scheme" ©1993 IEEE, pp 116-124.

[22] William B. Frakes and Kyo Kang, "Software Reuse Research: Status and Future", IEEE Transactions on Software Engineering, VOL. 31, NO. 7, JULY 2005

[23] R. Prieto-Diaz and P. Freeman, "Classifying Software for Reuse", IEEE Software, 1987, Vol. 4, No. 1, pp. 6-16.

[24] Rym Mili, Ali Mili, and Roland T. Mittermeir, "Storing and Retrieving Software Components a Refinement Based System", IEEE Transactions of Software Engineering, 1997, Vol. 23, No. 7, pp. 445-460.

[25] Roger S. Pressman, "Software Engineering A Practitioner's Approach", 5th Edition, Mc-Graw Hill.

[26] .Henninger, "An Evolutionary Approach to Constructing Effective Software Reuse Repositories", ACM Transactions on Software Engineering Methodology, no 2, 1997, pp. 111-150

[27] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", Communication of the ACM, Vol. 34, No.5, May 1991

[28] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", © 1990 IEEE, pp.300-304