Journal of Computer Science 4 (10): 821-825, 2008
ISSN 1549-3636
© 2008 Science Publications

# An Integrated Classification Scheme for Efficient Retrieval of Components

C.V. Guru Rao and P. Niranjan
Department of Computer Science and Engineering,
Kakatiya Institute of Technology and Science, Warangal, Andhra Pradesh, India-506 015

**Abstract:** Reuse is the key paradigm for increasing productivity and quality in software development. To be able to reuse software components, whether it is code or designs, it is necessary to locate the component that can be reused. Locating components, or even realizing 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 had been previously employed, free text, enumerated, attribute value and faceted classification. Experiences revealed that individual classification schemes were unable to solve the problems associated with component classification. We required a combination of classification techniques to meet the problems with individual schemes and to improve retrieval efficiency. This research looked at each of the classification techniques above, and proposes a new method for classifying component details within a repository.

**Key words:** Software reuse, component, classification techniques, reuse libraries

## INTRODUCTION

One of major impediments to realizing software reusability in many organizations is the inability to locate and retrieve existing software components. There often is a large body of software available for use on a new application, but the difficulty in locating the software or even being aware that it exists results in the same or similar components being re-invented over and over again. In order to overcome this impediment, a necessary first step is the ability to organize and catalog collections software components and provide the means for developers to quickly search a collection to identify candidates for potential reuse[2,16].

Software reuse is an important area of software engineering research that promises significant improvements in software productivity and quality[4]. Software reuse is the use of existing software or software knowledge to construct new software[11]. Effective software reuse requires that the users of the system have access to appropriate components. The user must access these components accurately and quickly, and be able to modify them if necessary.

Component is a well-defined unit of software that has a published interface and can be used in conjunction with components to form larger units[3]. Reuse deals with the ability to combine separate independent software components to form a larger unit of software. To incorporate reusable components into systems, programmers must be able to find and understand them. Classifying software allows reusers to organize collections of components into structures that they can search easily. Most retrieval methods require some kind of classification of the components. How to classify and which classifications to use must be decided, and all components put into relevant classes. The classification system will become outdated with time and new technology. Thus the classification system must be updated from time to time and some or all of the components will be affected by the change and need to reclassified.

## MATERIALS AND METHODS

**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[13] is to be able to located either the exact component required, or the closest match, in the shortest amount of tie, using a suitable

**Corresponding Author:** P. Niranjan, Department of Computer Science and Engineering,
Kakatiya Institute of Technology and Science, Warangal, Andhra Pradesh, India-506 015

query. The retrieved component (s) should then be available for examination and possible selection.

Classification is the process of assigning a class to a part of interest. The classification of components is more complicated than, say, classifying books in a library. A book library cataloguing system will typically use structured data for its classification system (e.g., the Dewey Decimal number). Current attempts to classify software components fall into the following categories: Free text, enumerated, attribute-value, and faceted. The suitability of each of the methods is assessed as to how well they perform against the previously described criteria for a good retrieval system, including how well they manage 'best effort retrieval'.

**Existing techniques:**

**Free text classification:** Free text retrieval performs searches using the text contained within documents. The retrieval system is typically based upon a keyword search[16]. All of the document indexes are searched to try to find an appropriate entry for the required keyword. An obvious flaw with this method is the ambiguous nature of the keywords used. Another disadvantage is that a search 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. Once approach to reducing the size of indexed data is to use a signature matching technique, however space reduced is 10-15% only.

**Enumerated classification:** Enumerated classification uses a set of mutually exclusive classes, which are all within a hierarchy of a single dimension[6]. A prime illustration of this is the Dewey Decimal system used to classify books in a library. Each subject area, e.g., Biology, Chemistry etc, has its own classifying code. As a sub code of this is a specialist subject area within the main subject. These codes can again be sub coded by author. This classification method has advantages and disadvantages pivoted around the concepts of a unique classification for each item. The classification scheme will allow a user to find more than one item that is classified within the same section/subsection assuming that if more than one exists. For example, there may be more than one book concerning a given subject, each written by a different author.

This type of classification schemes is one dimensional, and will not allow flexible classification of components into more than one place. As such, enumerated classification by itself does not provide a good classification scheme for reusable software components.

**Attribute value:** The attribute value classification schemes uses a set of attributes to classify a component[6]. For example, a book has many attributes such as the author, the publisher, it's ISBN number and it's classification code in the Dewey Decimal system. These are only example of the possible attributes. Depending upon who wants information about a book, the attributes could be concerned with the number of pages, the size of the paper used, the type of print face, the publishing date, etc. Clearly, the attributes relating to a book can be:

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

**Faceted:** 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[2,8,12,17] has proposed a faceted scheme that uses six facets.

- The functional facets are: Function, Objects and Medium
- The environmental facets are: System type, Functional area, Setting

Each of the facets has to have values assigned at the time the component is classified. The individual components can then be uniquely identified by a tuple, for example.

<add, arrays, buffer, database manager, billing, book store>

Clearly, it can be sent that each facet is ordered within the system. The facets furthest to the left of the tuple have the highest significance, whilst those to the right have a lower significance to the intended component. When a query is made for a suitable component, the query will consist of a tuple similar to the classification one, although certain fields may be omitted if desired. For example:

<add, arrays, buffer, database manager, *,*>

The most appropriate component can be selected from those returned since the more of the facets from the left that match the original query, the better the match will be.

Frakes and Pole conducted an investigation as to the most favorable of the above classification methods[9]. 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, number of attributes.

## RESULTS AND DISCUSSION

**Proposed classification:** Whilst it is obvious that some kind of classification and retrieval is required, one problem is how to actually implement this, most other systems follow the same principle: Once a new component has been identified, a librarian is responsible for the classification must be highly proficient with the classification system employed for two reasons. Firstly, the librarian must know how to classify the components according to the schema.

Secondly, a lexicographical consistency is required across the whole of the system. The classification system is separate to the retrieval system, which is for all of the users.

Most established systems tend to rigidly stick with one classification and retrieval scheme, such as free text or faceted. Others tend to use one type of retrieval system with a separate classification system such as the Reusable Software Library, which uses an Enumerated classification scheme with Free Text search.

In this research, we propose a new classification scheme that incorporates the features of existing classification schemes. In this system (Fig. 1) the administrator or librarian sets up the classification scheme. The developers develop and put their components into library. The users who are also developers can retrieve components from the library. Query tracking system can be maintained to improve the classification scheme. The proposed system will provide the following functionality to the users.

- Storing components
- Searching components
- Browsing components

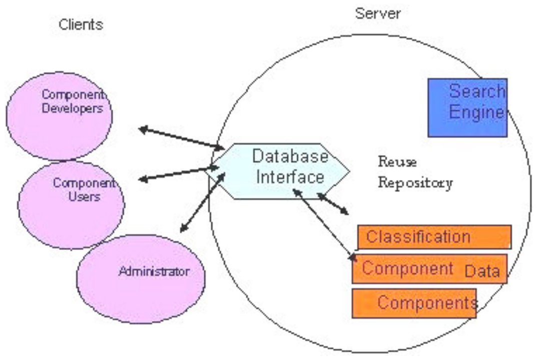


Fig. 1: Proposed system

The librarian task is to establish classification scheme. Each of the four main classification schemes has both advantages and disadvantages. The free text classification scheme does not provide the flexibility required for a classification system, and has too many problems with synonyms and search spaces. The faceted system of classification provides the most flexible method of classifying components. However, it can cause problems when trying to classify very similar components for use within different domains. The enumerated system provides a quick way to drill down into a library, but does not provide the flexibility within the library to classify components for use in more than one way. The attribute value system allows multidimensional classification of the same component, but will not allow any ordering of the different attributes.

Our solution to these problems would be to use an attribute value scheme combined with faceted classification scheme to classify the components details. The attribute value scheme is initially used to narrow down the search space. Among the available components in the repository only components matching with the given attributes are considered for faceted retrieval. Restrictions can be placed upon which hardware, vendor, O.S., type and languages attributes. This will be the restrictive layer of the classification architecture, by reducing the size of the search space. All, some or none of the attributes can be used, depending upon the required specificity or generality required. Then a faceted classification scheme is employed to access the component details within the repository. The proposed facets are similar to those used by Prieto-Diaz, but are not the same.

- Behavior. What the component does.
- Domain. To which application domain the components belong.
- Version. The version of the candidate component

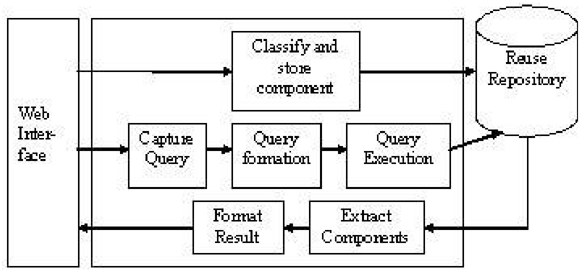


Fig. 2: Architecture of proposed System

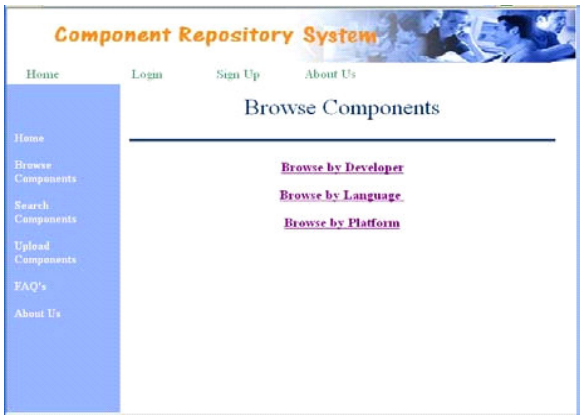
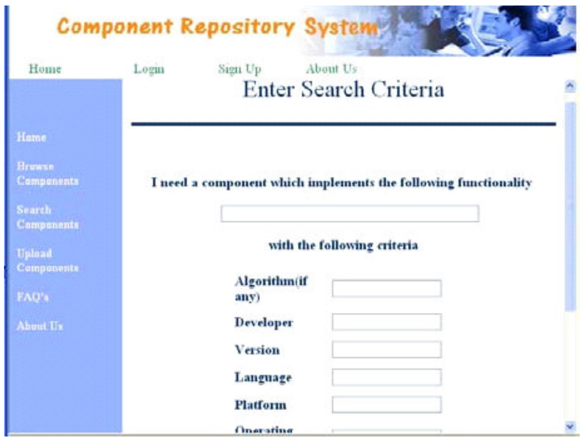


Fig. 3: Screen for browsing components



I need a component which implements the following functionality

with the following criteria

Algorithm(if any)
Developer
Version
Language
Platform
Operating

Fig. 4: Screen for searching components

Figure 2 shows the architecture of proposed system. The repository is library of component information, with the components stored within the underlying platforms. Users are provided with interface through which they can upload, retrieve and browse components. User will provide the details of the components to upload components. While retrieving components the user can give his query or he can give details so that the system will find the matching components and give the results. Then the user can select his choice from the list of components. The user can select from versions of components also. In addition the descriptions of the components are stored along with components facilitate text searches also.

Figure 3 and 4 shows the experimental prototype screens. The user can browse components by developer, language or platform. In the search screen the user can give his requirement along with fields that are optional.

## CONCLUSION

This study presented the methods to classify reusable components. Existing four main methods (free text, attribute value, enumerated and faceted classification) are investigated and presented advantages and disadvantages associated with them. The proposed classification system takes advantage of the positive sides of each classification scheme. This classification scheme for different parts of a component. The attribute value scheme is initially used within the classification for specifying the vendor, platform, operating system and development language relating to the component. This allows the search space to be restricted to specific libraries according to the selected attribute values. Additionally, this method will allow the searches to be either as generic or domain specific as required. The functionality of the component is then classified using a faceted scheme.

In addition to the functional facets is a facet for the version of the component. The version of a component is directly linked to its functionality as a whole, i.e. what it does, what it acts upon and what type of medium it operates within. Future work involved with this classification scheme will be to refine the scheme, and formalize it for implementation. A prototyped system for presenting and retrieving software reusable components based on this classification schema is now under implementation.

## REFERENCES

1. Henninger, S., 1997. An evolutionary approach to constructing effective software reuse repositories. ACM Trans. Software Eng. Methodol., 2: 111-150. DOI: 10.1145/248233.248242
2. Ruben Prieto-Diaz, 1991. Implementing faceted classification for software reuse. Commun. ACM, 34: 88-97. DOI: 10.1145/103167.103176

3. Gerald Kotonya, Ian Sommerville and Steve Hall, 2003. Towards a classification model for component based software engineering research. Proceeding of the of the 29th EUROMICRO Conference, Sep. 1-6, IEEE Computer Society Washington, DC, USA., pp: 43. http://portal.acm.org/citation.cfm?id=943278
4. William B. Frakes and Thomas. P. Pole, 1994. An empirical study of representation methods for reusable software components. IEEE Trans. Software Eng., 20: 617-630. DOI: 10.1109/32.310671
5. Sorumgard, L.S. G. Sindre and F. Stokke, 1993. Experiences from application of a faceted classification scheme. Proceedings of the2nd International Workshop on Advances in Software Reuse., Selected Papers from the Mar. 24-26, IEEE Xplore, USA., pp: 116-124. DOI: 10.1109/ASR.1993.291711
6. Jeffrey, S. Poulin and Kathryn P. Yglesias 1993. Experiences with a faceted classification scheme in a large Reusable Software Library (RSL). Proceedings of the 7th Annual International Conference on Computer Software and Applications Conference, Nov. 1-5, IEEE Xplore, Phoenix, AZ, USA., pp: 90-99. DOI: 10.1109/CMPSAC.1993.404220
7. De Jr Lucena, V.F., Facet-Based Classification Scheme for Industrial Automation Software Components. http://research.microsoft.com/en-us/um/people/cszypers/events/wcop2001/lucena.pdf
8. Ruben Prieto-Diaz, 1990. Implementing faceted classification for software reuse. Proceedings of the 12th International Conference on International Conference on Software Engineering, Mar. 26-30, IEEE Computer Society Press, Los Alamitos, CA., USA pp: 300-304. http://portal.acm.org/citation.cfm?id=100296.100338
9. Klement, J. Fellner and Klaus Turowski, 2000. Classification framework for business components. Proceedings of the 33rd Hawaii International Conference on System Sciences, Jan. 4-7, IEEE Xplore, USA. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=927009
10. Vitharana, Fatemeh, Jain, 2003. Knowledge based repository scheme for storing and retrieving business components: A theoretical design and an empirical analysis. IEEE Trans. Software Eng., 29: 649-664. DOI: 10.1109/TSE.2003.1214328
11. William B. Frakes and Kyo Knag, 2005. Software reuse research: status and future. IEEE Trans. Software Eng., 31: 529-536. DOI: 10.1109/TSE.2005.85
12. Prieto-Diaz, R. and P. Freeman, 1987. Classifying software for reuse. IEEE Software, 4: 6-16. DOI: 10.1109/MS.1987.229789
13. Rym Mili, Ali Mili, and Roland T. Mittermeir, 1997. Storing and retrieving software components a refinement based system. IEEE Trans. Software Eng., 23: 445-460. DOI: 10.1109/32.605762
14. Hafedh Mili, Estelle Ah-Ki, Robert Godin and Hamid Mcheick, 1997. Another nail to the coffin of faceted controlled vocabulary component classification and retrieval. Proceedings of the Symposium on Software Reusability May 1997, Boston USA., pp: 89-98. http://portal.acm.org/citation.cfm?doid=258368.258393
15. Hafedh Mili, Fatma Mili and Ali Mili, 1995. Reusing software: issues and research directions. IEEE Trans. Software Eng., 21: 528-562. DOI: 10.1109/32.391379
16. Gerald Jones and Ruben Prieto-Diaz, 1998. Building and managing software libraries. 1988. COMPSAC 88. Proceedings of the 12th International Conference on Computer Software and Applications, Oct. 5-7, IEEE Xplore, Chicago, IL, USA pp: 228-236. DOI: 10.1109/CMPSAC.1988.17177
17. Prieto-Diaz, Freeman, 1997. Classifying software for reuse. IEEE Software, 4: 6-16.