A Firm Retrieval of Software Reusable Component Based On Component Classification

Gowtham Gajala[1,] MV Phanindra[2]
[1, 2] Assistant Professor, Dept. of IT, Kakatiya Institute of Technology & Science, Warangal

Abstract

The reuse system presented here is based on the principles of Attribute value classification and Threshold value. It allows a software designer to define the component, and retrieve the components which are similar to the required one. Algorithms to compute these reuse candidates are described. Once the reuse candidates for the required operations have been selected, the reuse system computes a list of packages for the set of operations. Linear-Search Algorithms for determining the package reuse list are also presented. If the suggested packages don't satisfy the requirements, the user may try slightly different operation descriptions to find other candidates. This approach facilitates the user to browse among similar components in order to identify the best candidates for reuse. The proposed classification system takes advantage of the positive sides of each classification scheme, whilst hopefully rendering the negative sides redundant. This classification scheme uses the attribute value 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.

KEYWORDS: *Reuse, Software components, classification, search, insert, attributes.*

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.

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.

Component: 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. Example: 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 users to organize collections of components into structures that they can search easily.There have been many attempts to classify reusable components using various techniques. Normally, each of these methods has been implemented discretely. Each of the four main methods described (free text, attribute value, enumerated and faceted classification) has advantages and disadvantages associated with them. The proposed classification system takes advantage of the positive sides of each classification scheme, whilst hopefully rendering the negative sides redundant. This classification scheme uses the attribute value 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. The system also stores the descriptions of each component uploaded in the repository. So the system can also support keyword based search. If system stores most of the component's properties the system can serve better and can be used in different ways. Systematic software reuse is seen as a solution to address the need for short development time without compromising efficiency. Research is ongoing to develop more user-friendly and effective reuse systems. A considerable number of tools and mechanisms for supporting reuse activities in software development have been proposed.
Software Reuse:

Definition1: "Reusability is a measure of the ease with which one can use those previous concepts or objects in the new situations".
Definition2: "Reuse is the use of previously acquired concepts or objects in a new situation, it involves encoding development information at different levels of abstraction, storing this representation for future reference, matching of new and old situations, duplication of already developed objects and actions, and their adaptation to suit new requirements".
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.

II. EXISTING SYSTEM

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.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'.Component Classification Schemes: There are four classification techniques.

2.1 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.

2.2 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.

2.3 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.

2.4 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.

3. Proposed System

There have been many attempts to classify reusable components using various techniques. Normally, each of these methods has been implemented discretely. Each of the four main methods described (free text, attribute value, enumerated and faceted classification) has advantages and disadvantages associated with them. The proposed classification system takes advantage of the positive sides of each classification scheme, whilst

hopefully rendering the negative sides redundant. This classification scheme uses the attribute value 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 next step is retrieval of component based on the component name and the threshold value. The technique used here is linear search algorithm. First it retrieve based on component name from repository and then find the distance and compare it with the threshold value. If the distance value is less than the threshold value add the component to the final out put list and display them as the output. Here we have download option by click on it you can download that component.

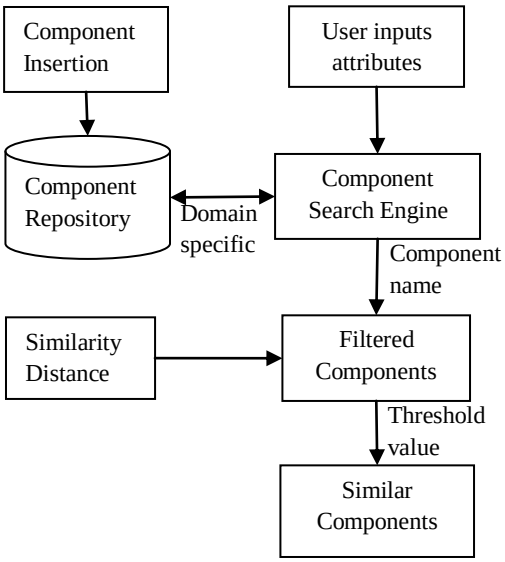

Fig. 1 Proposed System

3.1 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.

1. Operating system
2. Language
3. Keywords
4. Inputs
5. Outputs
6. Domain
7. Version
8. 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.

3.2 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.

3.3 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 componentsPlace table titles above the tables.
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.

4. Conclusion and Future Scope

The performance of this reuse system can be evaluated from the standpoint of user effort and maintenance effort. The user effort consists of all the effort which must be expended by the user in order to use the reuse system. It is very difficult to formally measure user effort. However, queries can be easily formulated, and therefore the user is not required to learn any formalism. The maintenance effort consists of all the effort which is necessary to keep the system working and up to date. This effort includes adding components to the knowledge base. The maintenance stage is highly facilitated in this system, as insertion of new components into the knowledge base can be done incrementally.

All the algorithms can be implemented in common lisp. The proposed reuse system can be used within an application domain like Unit, and utilize the reusable concepts of Ada. More recent object-oriented reusable designs like frameworks can also work with our system. One of the prime economic justification that for proposing this reuse system is to allow high-speed and low-cost replacement of aging systems, whose functions and data requirements have become well known.

User gets logged-in and searches for the components from the database. Then the user stores the searched components in the repository. Later on next user gets logged in and searches the component from the repository .Then the matched components are displayed on the grid view.

In addition to the retrieval of relevant component 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.

References

[1] Gowtham Gajala, "Implementation of Attribute values and Faceted value classification scheme for constructing Reuse Repository", International Journal of Computer Trends and Technology- volume4Issue1- 2013.

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

[3] R.Prieto-Diaz and P. Freeman, "Classifying Software for Reusability", *IEEE Soflware,* Ja.niiary 1987. pp. 6-16.

[4] Boelim, B., Penedo, M.H., Stuckle, E.D., Williams, R.D. and Pyster, A.B. "A Software Development Environment for Improving Software Productivity", *IEEE Computer,* 17(6), 1984, pp. 30-42.

[5] E. J. Ostertag and J. A. Hendler, An AI-based reuse system, Tech. Rep. *(2.9-TR-2197, UMIACSTR- 89-16,* Univ. of Maryland, Dept. of Computer Science, Feb 1989, pp. 1-2G.

[6] M. Wood and I. Sommerville, "An Information Retrieval System for Software Components," SIGIR Forum, Vol. 22, No. 314, Spring/Summer 1988, pp. 11-25.

[7] N. J. Nilsson, Principles of Artificial Intelligence, Morgan Kaufmann, CA, MIT Press, 1980, pp. G8-88. [GI Thomas H. Cormen, Charles E. Leisersoii and Ronald L. Rivest, Introduction to Algorithms, MIT Press, 1990, pp. 525-530.

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

[9] 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.