Enhanced Component Retrieval Scheme Using Suffix Tree in Object Oriented Paradigm

Karambir,
Assistant Professor
Department of Computer Science and Engineering
University Institute of Engineering and Technology(UIET)
Kurukshetra University, Kurukshetra
bidhankarambir@rediffmail.com

Nisha,
M. Tech.
Department of Computer Science and Engineering
University Institute of Engineering and Technology(UIET)
Kurukshetra University, Kurukshetra
ernisha.05@gmail.com

Abstract –In today's world software are used everywhere i.e. every electronic devices use software. So there is a large demand of software but in the same proportion the development is not growing. It is similar to the supply demand problem of management. Hence there is need to resolve this issue. There is a need to find some alternate methods that can help in improvement of this development. Some methods may be like software reuse, develop common software that can be used by various users simultaneously. In software reuse we use the design, code, architecture etc. Software reuse has become of much interest in the software community due to its potential benefits, cost benefit, time saving, etc. which include increased product quality and decreased product development cost and estimated schedule. To select a component for reuse is becoming difficult, because before reusing there is need to retrieve the component from the repository. Repository having a large in size and there are thousands no. of component. Before retrieving a component there is need to search the relevant component as there are many components with approximate same name, same functionality etc. which make the searching of component very time consuming. Hence there is need of a new technique which makes the selection of component efficient and fast. For this purpose, we proposed a scheme for the searching of component using suffix tree, suffix tree is a way to maintain the component repository in well maintained way and result an efficient and fast searching, which will be more efficient than earlier schemes.

Keywords: Software Component, Searching, Component Retrieval, Suffix Tree

I. INTRODUCTION

Software reuse is technique in which we use the already designed software's code in our software system. There are many benefits of software reuse. It saves the potential, cost, time etc. This result in increase in the product quality, delivery on time, under budgeted development and decrease in the maintenance cost also [1]. From the above discussed benefits, software reuse plays an important role in software development. When we reuse a component there is need to select the proper component that will be used. For reusing there is a need to retrieve the components form the repository which is a special database containing a large number of components that may be reused in development process. Before retrieving a component there is need to search that component. There are a lot of component having similar name, similar functionality etc. that make the searching process difficult and ambiguous. Each component is stored with a name generally a string. So to search a component keyword based searching is applied. So the existing string matching algorithms can be applied on the component searching. Some techniques are earlier developed that can be used to search a component that are based on the string matching algorithms which are described below. They have their own benefits and drawbacks but do not fulfil all the requirements.

A. *Rabin Karp Algorithm [2]:* It is a string matching algorithm that uses hashing to find any one of a set of pattern strings in a text. The text of length n and p patterns of combined length m, its average and best case running time is O (n+m) in space O (p), but its worst-case time is O (nm). Rabin Karp can rapidly search through a paper for instances of sentences from the given source material, ignoring details such as case and punctuation.

B. *Brute Force Algorithm[3]:* It is also known as proof by exhaustion, also known as proof by cases, perfect induction, or the brute force method, is a method of mathematical proof in which the statement to be proved is split into a finite number of cases and each case is checked to see if the proposition in question holds. A proof by exhaustion contains two stages: proof that the cases are exhaustive; i.e., that each instance of the statement to be proved matches the conditions of (at least) one of the cases and a proof of each of the cases.

C. *Boyer Moore Algorithm [4]:* It is a particularly efficient string searching algorithm, and it has been the standard benchmark for the practical string search literature. The algorithm pre-processes the target string that is being searched for, but not the string being searched in. The execution time of the Boyer Moore algorithm, while still linear in the size of the string being searched, can have a significantly lower constant factor than many other search algorithms: it doesn't need to check every character of the string to be searched, but rather skips over some of them. Generally the algorithm gets faster as the key being searched for becomes longer. Its efficiency derives from the fact that with each unsuccessful attempt to find a match between the search string and the text it is searching, it uses the information gained from that attempt to rule out as many positions of the text as possible where the string cannot match. BMH approach uses only the Bad

D. *Knuth Morris Pratt Algorithm [5]:* KMP string searching algorithm searches for occurrences of a "word" W within a main "text string" T by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters.

E. *Longest Common Subsequence (Lcs) Algorithm [6]:* It is to find the longest subsequence common to all sequences in a set of sequences. The subsequence is different from a substring, it is a classic computer science problem, the basis of different (a file comparison program that outputs the differences between two files), and has applications in bioinformatics.

The above discussed algorithms are string matching algorithms that are used to match the strings. In component searching, the component is stored using a keyword that is also a string, so these mechanisms can be used. These searching mechanisms are efficient and each has its own advantages and disadvantages. To improve the above algorithms some more work has done in this field which are described in the section 2. Proposed work is described in section 3, section 4 contains their results and conclusion is written in section 5.

2. RELATED WORK

Software component is stored with a name which is combination of characters known as keyword. To search a component keyword matching is generally used. Simple string matching algorithms were discussed above. But today these algorithms are not much in use, because they have many pitfalls. Some of the work has been done on their pitfalls and some new relevant and efficient techniques are developed for component selection. Various authors scheme are described below that make improvement in searching mechanism.

In this paper, Dixit et. Al. [7], gave an idea to retrieve a component using the genetic algorithms, they tried to solve the issue of Component selection. This paper described how a Genetic Algorithms based approach can be used for component selection to minimize the gap between components needed and components available. Now a relevant objective has at hand in this direction that is to make use of these methodologies acceptable from the software engineering community. Therefore, in this paper he developed Genetic Algorithms based approach for selection component.

In this paper, Viana et al. [8] gave a new scheme named as ‘A Search Service for Software Components based on Semi-Structured Data Representation Model’. This paper presented the architecture, functionalities and implementation of a search service that adopts techniques for indexing semi structured data, making possible the discovery of software assets through regular path expression queries. The search service proposed in this paper performed the indexing of assets described using a semi structured data representation model, as opposed to automatic extraction of information from the source code or textual documentation approaches.

In this paper, Li et. Al. [9] performed a work on, “Component Retrieval Based on Domain Ontology and User Interest”. In this paper, a method of retrieving software components based on domain ontology and user interest was studied and implemented**.** This paper emphasized the definition of ontology feature domain model, the presentation of component description model based on ontology feature and the retrieval method of user interest. Based on these, he presented the component retrieval framework and an algorithm for retrieving related components. Finally, a component retrieval system was given, and an instance with components in E-Commerce field proved validity comparing with the retrieval methods based on keywords and facet.

In this paper, Aboud *et al.* [10] premised a work on, “Automated architectural component classification using concept lattices”. This paper discussed that, as the use of components grown in software development, building effective component directories became a critical issue as architects required help to search components in repositories. During the life-cycle of component-based software, several tasks, such as construction from scratch

or component substitution, would benefit from an efficient component classification and retrieval. In this paper, he analyzed how we can build a classification of components using their technical description (i.e. functions and interfaces) in order to help automatic as well as manual composition and substitution. The approach was implemented in the CoCoLa prototype, which was dedicated to Fractal component directory management and validated through a case study.

In this paper, Peng *et al.* [11] performed a work on, "An Ontology-Driven Paradigm for Component Representation and Retrieval". Here the key factors of component reuse were discussed and it was pointed out that component reuse is actually the reuse of knowledge about component. Component ontology was employed to represent the knowledge about component. Domain-specific terms were used to represent component by importing domain ontology into component ontology. In this paper component retrieving algorithm was implemented by ontology query and reasoning. This model was used in a large scale distributed simulation system and the fact revealed that component ontology was flexible enough for component reuse and efficiency of retrieving algorithm.

Khode et. al. [1] presented a paper entitled, "Improving Retrieval Effectiveness using Ant Colony Optimization". They proposed a technique that helps re-user to identify and retrieve software component. In their first step it matched keywords, their synonyms and their interrelationships. And then made use of ant colony optimization, a probabilistic approach to generate rule for matching the component against the re-user query. The method also shows very good values of precision and recall.

Shao[12] presented a work on **,** "Research on Decision Tree in Component Retrieval". In accordance with the limitation of research on traditional software component library management, he proposed the idea to apply data mining technology to the management of software component, provided auxiliary decision support to the relevant personnel of the component library. Secondly, in accordance with actual application, he built an applied model of software component retrieval management by data mining technology, and analyzed the execution step of the applied model. Lastly, the model had been verified through experiment, thus the feasibility and validity of this strategy had been verified.

From the above discussion, we can conclude that software searching is a difficult task and a lot of work is doing in this field. There is still many pitfalls and required more work.

3. PROBLEM IDENTIFIED

In Component Retrieval there is need to search a component. To search component, the user enter a query into the search interface of the searching module. The searching module returns a list of all the components according to search query as a result. A lot of research goes into how to search the best component in return. To provide results quickly and efficiently is a big challenge. As the repository size grows the time required to search a component increases. This is also a major issue that needs to resolve. Hence we take these two issues as a problem and gave a solution.

4. PROPOSED WORK

In the above section, the problem definition was discussed. Suffix Tree is a data structure that is also used for the purpose of the string matching. The suffix tree is often used for storing words or sequences of words so they can be looked up easily [13]. It was also called position tree but it was extremely difficult to understand. The improved algorithm that was simpler and better space efficiency for suffix tree construction was given by McCreight in 1979 is known as "A Space-Economical Suffix Tree Construction Algorithm" [14]. The tree was working backward from the longest suffix to the shortest. Building the tree in this manner is simple and allows the tree to be built in a single scan of the string.

A new algorithm named as "Constructing Suffix Trees On-Line in Linear Time" was given by Ukkonen[15] in 1992. This algorithm is easier to understand and offers several properties that make it useful for the language modelling. Ukkonen's algorithm for suffix tree formation is easiest and different from the previous algorithms. It starts with an empty tree and extending it for each character in in the string in a single forward scan. The tree formed in this manner called on-line algorithm: so that after inserting the Nth character the suffix tree is complete for the first N characters of the input string.

Suffix tree [16] can be defined as: Let S=S [1...n] be a string of length *n* over a fixed alphabet Σ. A suffix tree for S is a tree with n leaves (representing n suffixes) and the following properties:

- Every internal node other than the root has at least 2 children.
- Every edge is labeled with a nonempty substring of *S*.
- The edges leaving a given node have labels starting with different letters.
- The concatenation of the labels of the path from the root to leaf *i* spell out the i-th suffix S [i...n] of S. We denote S [i...n] by S_i.
- In the above paragraph some features and method to construct suffix tree has been discussed. Suffix tree is a data structure that can be used for string matching purpose. If suffix tree can be applied on the

component searching it may return the efficient and qualitative results and the another benefit of the suffix tree is that at each level of suffix tree, there are at least two nodes hence when a new component is added for the searching it is simply add with the existing nodes so there is least change in the hierarchy of the suffix tree. Hence there is not much effect of the increase in the size of the repository. As the size of the component repository increases there will be little increment in the searching time. So I proposed a new architecture for the component searching from repository using suffix tree.

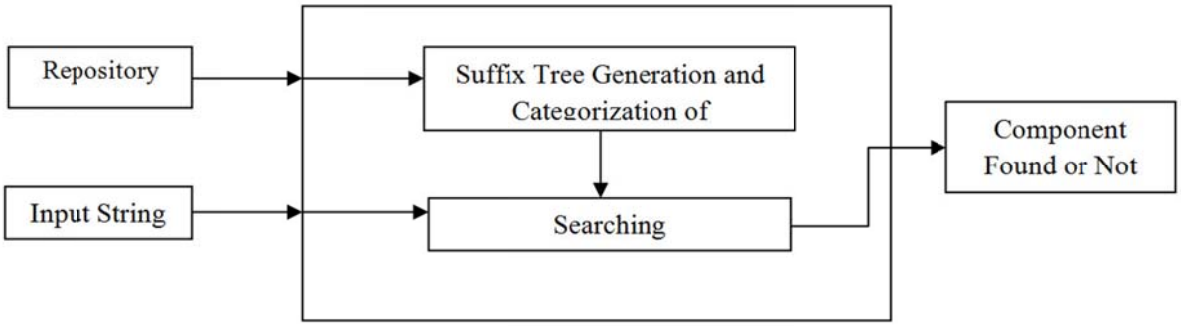

Figure 1: - Architecture of Proposed Searching Scheme Based On Suffix Tree

In this architecture, there is one input module from which we can enter component repository and a string to be searched. In the second module, which is searching module, in step 1, the suffix tree is generated according to the number of components presented in the component repository that is given as an input and in step 2 the input string is searched on the suffix tree. If the component found on the suffix tree then the output module return the result component found with the time used in searching that component and vice-versa.

5. IMPLEMENTATION AND RESULT

We implemented our proposed work in J2SE. We formed a prototype of the above architecture and gave Template.java file as a component repository to the prototype. It found and showed the all the components like classes available, methods available, and the fields available. This is also shown in figure 2. In the figure 3, we showed the suffix tree formed according to components found in repository and in table 1. We searched the various components in the suffix tree and got result and their corresponding searching time is shown in tabular form. The suffix tree is formed in a linear time and component searching is also performed in the linear time hence the searching time of this architecture is also linear that makes it more efficient than already existing algorithms.

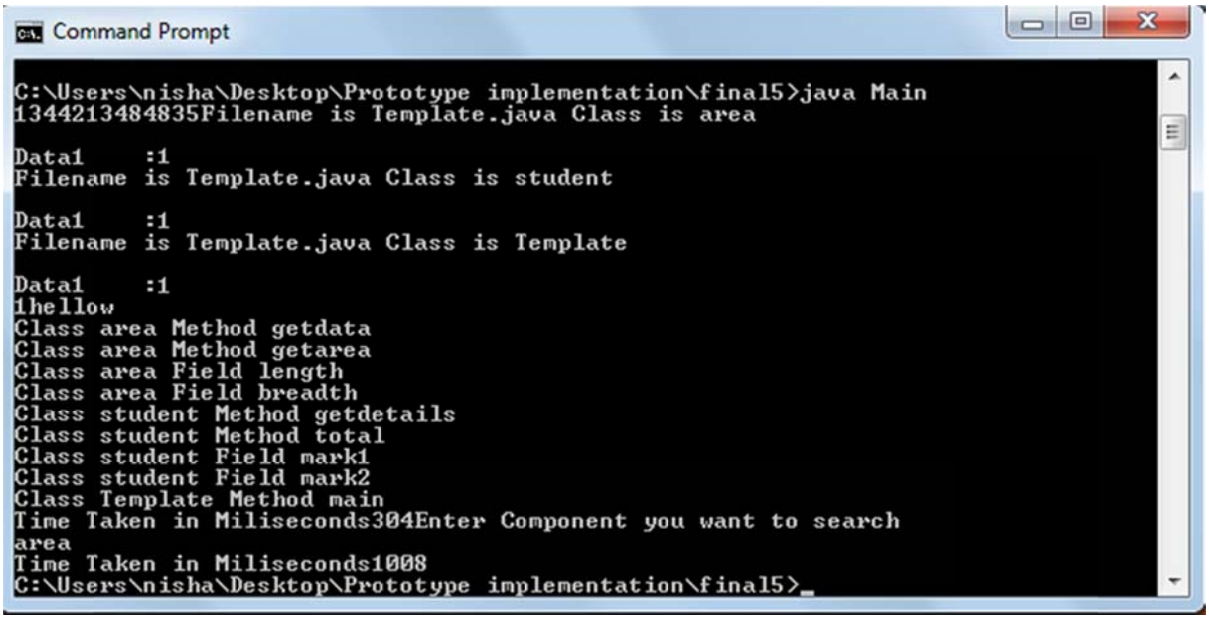

Fig 2:- Finding the components in component repository

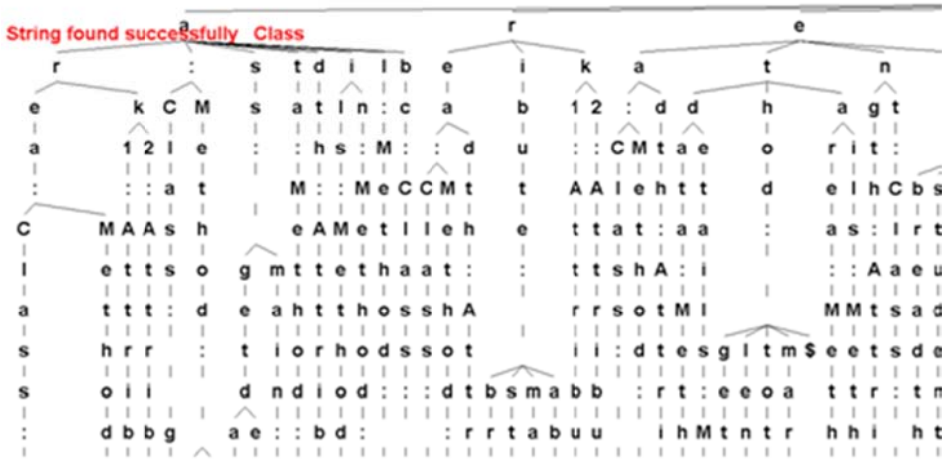

Fig 3:- Suffix Tree generated according to components found

Table1:- Searching time of different components using suffix tree

Sr.No.	Type Of Components	Name of component	Time Complexity in millisec (Searching time from suffix tree)
1	Method	getdata( )	1030
2	Class	area	1008
3	Method	total( )	1030
4	Method	mark1( )	1014
5	Class	student	1014
6	Method	getdetails ( )	1217
7	Method	getarea()	1009

6. CONCLUSION

In this paper, we presented an approach to search a component from the component repository using suffix tree. In our method, we gave the path of component repository. The prototype distinguish the components presented in the repository like classes, methods etc. That may be reused. Then according to component found in repository the suffix tree is generated. The user's query is matched from the suffix tree, if the query is matched it returns their searching time otherwise failure. The running time of this algorithm is linear that made it fast and efficient; this algorithm matched all the characters of input string in a series that make the qualitative results. These two issues were major in software component searching that was solved using our proposed system. This algorithm takes very small time that is also shown in above table. As a conclusion, we can say that proposed approach will be very efficient and useful for the software industry to search a component for reuse.

REFERENCES

[1] Sandeep G. Khode et al., "Improving Retrieval Effectiveness using Ant Colony Optimization", IEEE 2009.

[2] Karp et al., "Efficient randomized pattern-matching algorithms", march 1987.

[3] Nimisha singla et al., "String Matching Algorithms and their Applicability in various Applications", ijsce ISSN: 2231-2307, Volume-I, Issue-6, January 2012.

[4] Boyer et. al., "A fast string searching algorithm.", Comm. ACM page no. 762-772 DOI: 10.1145/359842.359859. ISSN 0001-0782.

[5] Knuth et al., "Fast pattern matching in strings", SIAM journal on Computing 6(2): 323-350. DOI: 10. 1137/020624.

[6] Thomas H.Corman, Charles E. Lierersin, Ronald L. Rivest and Clifford Stein (2001). "15.4". Introduction to Algorithms (2nd ed.) MIT PRESS and McGraw-Hill. Pp. 350-355. ISBN 0-262-53196-8

[7] Dixit et. al., "Software Component Retrieval Using Genetic Algorithms", International Conference on Computer and Automation Engineering, 2009 IEEE, ISBN: 978-0-7695-3569-2, pp. 151-155.

[8] Viana et. al., "A Search service for software components based on a semi-structured data representation model", 6th international Conference on information technology, IEEE, ISBN: 978-1-4244-3770-2, pp. 1478-1484.

[9] Li et. al., "Component Retrieval Based on Domain Ontology and User Interest", EBISS International Conference, IEEE, ISBN: 978-1-4244-2909-7, pp. 1-4.

[10] Aboud et. al., "Automated architectural component classification using concept lattices", Software Architecture & European Conference on Software Architecture WICSA/ECSA@2009 IEEE, ISBN: 978-1-4244-4984-2, pp. 21 -30.

[11] Peng et al.," An Ontology-Driven Paradigm for component Representation and Retrieval", Ninth International Conference on Computer and Information Technology © IEEE, ISBN: 978-0-7695-3836-5, Vol. 2., pp. 187-192.
[12] Shao et. al., "Research on Decision Tree in Component Retrieval", Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD) IEEE, ISBN: 978-1-4244-5931-5, Vol. 5, pp. 2290- 2293.
[13] Weiner et. al., "Linear pattern matching algorithms" Proceedings of the 14th Annual Symposium of Switching and Automata Theory, IEEE Computer Society,pp.1-11.
[14] McCreight,E., (1976) "A space-economical suffix tree construction algorithm" ACM,Vol.23, pp.262-272.
[15] Ukkonen,E., "Constructing suffix trees on-line in linear time" In Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture Information Processing '92,Vol.1,pp.484-492.
[16] Gusfield et. al., "Algorithms on Strings, Trees, and Sequences" Computer Science and Computational Biology. Cambridge University Press.

Authors Profile

Mr. Karambir is currently working as a assistant professor in Department of Computer Science and Engineering,University Institute of Engineering and Technology (UIET) Kurukshetra University, Kurukshetra. He did his B.Tech from Nagpur University and received his M.Tech degree from GJU, Hisar, Haryana.

Nisha was born in Haryana, India in 1989. She received her B.Tech degree in Computer Engineering from DIET, Karnal. Presently she is a M.Tech scholar and pursuing M.Tech from UIET, Kurukshetra University, Haryana, India.