DESIDOC Bulletin of Information Technology, Vol. 27, No. 5, September 2007, pp. 27-34

A Hybrid Technique for Searching a Reusable Component from Software Libraries

Rajesh K. Bhatia, Mayank Dave* and R.C. Joshi**

Department of Computer Science and Engineering
Thapar Institute of Engineering and Technology, Patiala-147 004

** Department of Computer Engineering*
National Institute of Technology, Kurukshetra-136 119

*** Department of Electronics and Computer Engineering*
Indian Institute of Technology, Roorkee-246 667

ABSTRACT

Reusing a previously developed and tested software component is the key to improve the quality and productivity of the software. Searching a software from a software library with an intent to reuse it is similar to searching a book from a library. Locating a book or document in library may not be that difficult as searching a reusable component in a software library. The main challenge in reusing software component lies in the retrieval and selection of the appropriate component that would need no or least adaptation from a software component library. Book can be searched on title, authors or some keywords, but these features are not sufficient for searching the required functionality in a component library. Using formal specifications to represent software components facilitates the determination of reusability of the software component. The functionality of the software, and the well-defined syntax of formal language makes processing amenable to automation. In this paper, a hybrid model based on natural language and formal specifications using K-nn technique has been discussed. Benefits of both formal methods and natural language have been exploited for the retrieval of reusable software components. Existing components have been weighted according to their degree of similarity on the basis of certain attributes to the required component.

Keywords: Software library, formal specifications, software reuse, latex markup, *Z* notations, K-nn technique

1. INTRODUCTION

Earlier libraries used to store only books, magazines, newspapers and other published material. With advances in information technology libraries now store data in electronic form also; rather we have e-libraries now. So the architecture and organisation of libraries are also changing. Now libraries are supposed to store knowledge, in an easy to access form. But, as the amount of knowledge is growing, it is difficult to store it in easy to access form. Software reuse is a previously developed concept in this new situation and

Received 21 January 2007

a mean for overcoming the software crisis[1,2,3]. Current techniques to represent and manage software components libraries are not sufficient. Software systems based on analysis of natural language documentation have been proposed for constructing software libraries[4,5]. However, searching of software components based on natural language may lead to ambiguity and inappropriate results.

All other problems due to natural language like incompleteness and inconsistency can also be minimised using formal specifications to represent software components[6-9]. In the present work a library of reusable software components has been constructed. Software components are considered similar to documents stored in libraries. The way a user asseses any document, book or any other knowledge component in a library, in the same way developers need to asses reusable components from reusable software libraries. The proposed technique is for software component search. A software component can be considered analogous to a book or any other source. So, the same technique can be applied to software components libraries.

1.1 Software Reuse

The major tasks of reuse system are to classify the reusable components, add them to a software components library, and retrieve them whenever needed. It involves encoding information at various levels of abstraction, storing this information for future reference, matching of new and old situations, duplication of already developed objects and actions, and their adaptation to suit new requirements. Effective reuse of software increases productivity, saves time, and reduces cost of software development. Various components that can be reused include code fragments, logical program structures, functional structures, domain knowledge, requirements, design documents, etc. The process of software reuse comprises three stages of component processing, i.e., analysis, organisation, and synthesis[10]. The first step in analysis is the identification of components in the present domain. Then comes the understanding and representation in a suitable formalism to reflect their function and semantics with possible generalisation to widen the scope of their future applications. Second step is classification of software components followed by its storage in appropriate software repositories. It also includes searching and retrieval whenever needed. Third step includes selection of a component from retrieved candidates, its adaptation and integration in the new software. Here, the main focus is on the component organisation of the software reuse in which storing, searching, and retrieving is done. It also covers the representation of the code fragment.

1.2 Formal Methods

Formal methods refer to the use of techniques from formal logic and discrete mathematics in the specification, design, and construction of computer systems and softwares. This is analogous to the role of mathematics in all other engineering disciplines. Mathematics provides ways of modelling and predicting the behaviour of systems through calculation. Examples of formal languages are *Z*, VDM, Larch, Promela, ACSR, Trio and x-machines, etc. Reuse systems use formal methods that remove ambiguities, incompleteness and inconsistencies. Formal modelling of a system usually entails translating description of the system from a non-mathematical model (data-flow diagrams, object diagrams, scenarios, English text, etc.) into a formal specification using one of several formal languages. This results in a system description that possesses a high degree of logical precision. The use of formal methods to specify components can more precisely describe the functionality of the components that helps to retrieve the set of more appropriate and required components.

2. SPECIFICATION OF SOFTWARE COMPONENTS

Formal methods can be used to describe software components in the library. In this paper, *Z* notation has been used to specify software components. The query specification is in *Z* notation, which is compared with the existing specification in the database.

The *Z* notation[11] is a model-oriented formal specification language developed by the Programming Research Group at Oxford University Computing Laboratory in the early 1980s. Since then, it has been used to specify a wide spectrum of software systems including database systems, transaction systems, distributed computing systems, and operating systems[12]. *Z* is a non-executable but strongly-typed specification language. ZTC is a type-checker for *Z*, which determines if there are syntactical and typing errors in *Z* specifications[13]. There is no compiler for Z. However, there are tools to animate, or execute, subsets of *Z*.

LATEX markup is used to specify components in *Z* notation. A markup is a mapping to or from unicode representation. LATEX markup based on a 7-bit ASCII is suitable for processing by tools to render *Z* characters in their mathematical form[14]. The markups described show how to translate between a markup token (strings of ASCII markup characters) into the corresponding string of *Z* characters. Remaining individual markup characters that do not form a special markup token such as digits, latin letters, and much punctuation convert directly to the corresponding *Z* character, from ASCII-xy to Unicode U+00xy. A LATEX command is a backslash '\' followed by a string of alphabetic characters (up to the first non-alphabetic character), or by a single non-alphabetic character. The LATEX markups for the *Z* specifications are given below:

```
\begin{spec}
\begin{zed}
        [NAME, INFO]
\end{zed}

\begin{schema}{DataDictionary}
        dict: NAME\pfun INFO\\
         defined: \power NAME
\where
         defined = \dom dict
\end{schema}
\end{spec}
```

3. K-NEAREST NEIGHBOURS TECHNIQUE

The term nearest neighbours technique refers to a technique that identifies the closest points to a given point in some multi-dimensional space. How nearness is best measured and how data is best organised are challenging, non-trivial problems. Any k-nearest neighbours technique is effective only to the extent that the assumption of clustering behaviour is correct. The k-nearest neighbours method is most frequently used to tentatively classify points when firm class bounds are not established. The vital part of any nearest neighbours technique is an appropriate distance metric, or similarity-measuring function[15]. It is reasonable to assume that objects or observations, which are close together according to some appropriate metric, will have the same attributes, behaviour and thus classification.

4. PROPOSED SYSTEM USING K-NN TECHNIQUE

The aim of using nearest neighbours techniques is that items with similar attributes tend to cluster. On the basis of certain attributes, similarity between the two specifications is determined. As stated in the Introduction of the paper, using natural language alone may hinder the retrieval process due to the problem of ambiguity, incompleteness, and inconsistency. To remove these problems, formal methods were also used as they have numerous benefits for describing the system and in software reuse. Therefore, use of both the natural language and formal methods were tried for the reuse system. This paper presents a hybrid approach, which uses both formal method and natural language for the library construction and for retrieval of reusable software components. This approach is flexible as it allows user to select weights. If natural keywords in a system appear to give better results, then more weight can be assigned to natural keywords, otherwise, more weights can be assigned to formal specifications.

Formal specifications of the software components are stored in the library along with URL and relevant keywords in natural language for a particular component. Existing components were weighed according to their degree of similarity to the required component. The proposed reuse system compares query and existing components in terms of their

numerous features. The following attributes were used to compute the similarity:

- Ratio of number of keywords matched to number of keywords entered.
- Ratio of number of keywords matched to number of keywords in a database for a component specification.
- Declaration part of schema.
- Ratio of number of LATEX markup matched to the total number of input markup in query specification.
- Ratio of number of LATEX markup matched to the total number of markups in specifications present in the database.

To implement the system, different weights were assigned to different parts of the specifications (Table 1). To reduce the search space, keywords were first matched. Specifications corresponding to matched keywords were then matched to the query specification to get the percentage match of the component. The outcome was the set of retrieved components with percentage of similarity in descending order. The equation used to calculate the percentage match of query specification and the specifications in the database, using the assigned weights, is as follows:

$$\text{Percentage match} = (n_m/N_1 * W_1) + (n_m/N_2 * W_2) + m\ (W_3 + t_{i\max})$$

where, $t_{i\max} = (n_i/Nq_i * W_4) + (n_i/Nd_i * W_5)$

i = number of method schema in a component and i = 1, in case of state schema

m = 1, in the case of declaration match, or 0 otherwise

n_m = number of keywords matched

N_1 = number of keywords entered by user for matching

N_2 = number of keywords present in the database for a component specification

n_i = number of Z notation markups matched

Nq_i = number of Z notation markups entered by user in predicate part or data invariant part of query specification

Nd_i = number of Z notation markups in predicate part or data invariant part of the specification to be matched in the database

W_1 = Percentage weight given to query keywords match

W_2 = Percentage weight given to keywords match present in database

W_3 = Percentage weight given to declaration part of the schema

W_4 = Percentage weight given to number of markups matched entered by user in predicate part of the schema

W_5 = Percentage weight given to number of markups matched present in specification of database in predicate part of the schema.

Table 1. Weights assigned to various specifications

Attribute	Percentage weight assigned
Ratio of number of keywords matched to number of input keywords (W_1)	10
Ratio of number of keywords matched to number of keywords in a database for a component specification (excess keywords) (W_2)	10
Declaration part of schema (W_3)	35
Ratio of number of markups matched to number of input markups in query specification (predicate part) (W_4)	25
Ratio of number of markups matched to number of markups in specification present in database (predicate part) (W_5)	20

5. TOOL SUPPORT

ReuseWELL system was implemented based on k-nn technique. The system is implemented in Java, and takes the input of *Z* specification in LATEX markup as per ISO Standard 13568 for *Z* notation because all systems do not support 16-bit Unicode[15], the definitive representation of *Z* characters. Only a limited subset of LATEX markup was used for implementation of ReuseWELL system.

Though the ReuseWELL tool is specifically designed for retrieving components, similar types of systems should be applicable to other kinds of analog prediction problems where large databases exists. The proposed system will comprise two parts: (i) a large database of components that are specified with *Z* notation using LATEX markup along with natural specification of component for search, and (ii) K-nn algorithm that measures the similarity between the two specifications based on some attributes.

5.1 Database of *Z* Specifications of Components

The developed database comprises *Z* specifications of the components using LATEX markup along with the keywords and URL of a particular component. There are four tables in the database. Description of each table and its fields are:

KeywordsT: In this table, there are total two fields, keywords and ID. Keywords field contains the keywords and ID field contains identifications for the corresponding keywords entered by the user for adding different specifications.

URLT: In this table, there are two fields ID and URL. In the table, specification ID and location of the actual components are there.

StateschemaT: There are three fields in this table, ID, state, and data invariants. State of the system, if any, is added in the state field. Data invariants are stored in the data invariants field. ID specification is stored in the ID field, which is in *Z* notation using LATEX markup.

SchemaT: There are three fields in this table, ID, signature, and predicates. Method signature, if any, is stored in signature field and its predicates, i.e., pre- and post- conditions are stored in the predicates field. The ID of the specification, which is in *Z* notation using LATEX markup, is stored in the ID field.

5.2 Steps for Matching

Following steps are used for matching:

(i) Find the specifications in the library whose keywords matched with input keywords

(ii) Find similarity on the basis of number of keywords matched between query and available specifications

(iii) Find similarity of specifications of each input method to the specifications of the every other method of that particular component schema

(iv) Method with highest percentage of similarity with input method is considered and its similarity is added to the overall percentage of similarity between the specifications of components

(v) Overall similarity is computed on the basis of different attributes and it is placed at a proper position in descending order of similarity in a list of similar component for user

(vi) Steps 1 to 5 are repeated until no other match is found

(vii) List of components with their URL or addresses is displayed to the user.

6. CASE STUDY

Query specifications entered for matching and specifications of the software component present in the library are shown in Table 2 as:

$N_1 = 3$ $N_2 = 4$ $n_m = 2$

$n_i = 5$ $Nq_i = 5$ $Nd_i = 8$

$W_1 = 10$ $W_2 = 10$ $W_3 = 35$

$W_4 = 25$

$W_5 = 20$; $m = 1$, because number and type of markups in both the query and present

Table 2. An example for matching

Input Specifications	Specifications available
1) Natural language specification entered—add, insert, Birthday Book 2) Formal specification entered *Declaration part: -* \Delta Birthday Book name?: NAME date?: DATE *Predicate part:* name? \notin known birthday' = birthday \cup {name? \inj date?}	1) Natural language specification present—add, save, date, Birthday Book 2) Formal specification present *Declaration part: -* \Delta Birthday Book name?: NAME date?: DATE *Predicate part:* name? \notin known birthday' = birthday \cup {name? \inj date?} \land \# count < max

specification is the same

t = (5/5 * 25) + (5/8 * 20) = 37.5

Therefore, percentage match =

(2/3*10) + (2/4*10) + 1(35 + 37.5) = 84.16

Percentage match for the entered query specification with this particular specification present in database is 84.16. Keeping all other specifications same, if the number of input keywords or keywords present in the database are changed, the numbers of keywords match may vary. Results will change as shown in Table 3.

Case 1: Number of keywords available in natural language specifications in database for a component is in excess as compared to matched keyword, resulting in per centage match of 84.16.

Case 2: As compared to Case 1, here number of available keywords and keywords matched are the same (no excess keywords), which resulted in increase in percentage match from 84.16 to 89.16.

Case 3: In this case, number of input keywords and matched keywords are the same, which gives the percentage match of 90.

Case 4: In this case, the number of input keywords, available keywords, and matched keywords is the same, which further improves the percentage match to 92.5 from 84.6.

To further increase the percentage match, the number of markups entered by the user in predicate part of input specification should match with the number of markups present in predicate part of available specification.

7. CONCLUSION AND FUTURE SCOPE

The paper discussed the reuse and benefits of formal methods in reuse. It also demonstrated the benefits of using natural language along with formal methods, which is supported by this tool. Approach followed in the present work, is not fully dependent on formal specification; instead it tried to exploit the

Table 3. Percentage result match for different cases

	No. of input keywords (N_1)	No. of matched keywords (n_m)	No. of keywords present (N_2)	% match
Case 1	3	2	4	84.16
Case 2	3	2	2	89.16
Case 3	3	3	4	90.00
Case 4	3	3	3	92.50

benefits of both formal methods and natural language in the retrieval of software components.

Its future scope includes extending this work for more accuracy and optimisation in the component retrieval process. Further, this can be explored in the following directions:

- At present, the ReuseWELL system has just been implemented for subset of the *Z* notation. It can be extended and implemented for every *Z* markup.
- ReuseWELL tool has been implemented for software library, the same can be extended for other libraries also.

REFERENCES

1. Biggerstaff, Ted J. Software reusability: Concepts and models, Vol. 1. ACM Press, New York, 1989.
2. Biggerstaff, Ted J. Software reusability: Application and experience, Vol. 2. ACM Press, New York, 1989.
3. Krueger, Charles W. Software reuse. *ACM Computing Surveys,* 1992, **24**(2), 131-83.
4. Maarek, Y.S.; Berry, D.M. & Kaiser, G.E. An information retrieval approach for automatic constructing software libraries. *IEEE Trans. Software Engineering*, 1991, **17**(8), 800–913.
5. Helm, R. & Maarek, Y.S. Integrating information retrieval and domain-specific approaches for browsing and retrieval in object-oriented class libraries. *In* Proceedings of OOPSLA '91, 1991, pp. 47-61.
6. Betty, H.C.; Cheng; & Jeng, Jun-Jang. Formal methods applied to reuse. *In* Proceedings of the Fifth Workshop in Software Reuse, 1992.
7. London, R.L. Specifying reusable components using Z: Realistics sets and dictionaries. *ACM SIGSOFT Software Engineering Notes,* 1989, **14**(3), 120-27.
8. Weide, B.W.; Ogden, W.F. & Zweben, S.H. Reusable software components. *Advances in Computers*, 1991, **33,** 1- 65.
9. Wing, Jeannette M. A specifiers introduction to formal methods. *IEEE Computer*, 1990, **23**(9), 8–24.
10. Cybulski, Jacob L. Introduction to software reuse. Department of Information Systems, The University of Melbourne, Parkville, Australia.
11. The *Z* notation: A reference manual, Ed. 2, edited by J.M. Spivey. Prentice Hall International, London, 1992.
12. Specification case studies, Ed. 2, edited by I. Hayes. Prentice Hall International, London, 1993.
13. Xiaoping, Jia. ZTC: A type checker for Z. Institute of Software Engineering, DePaul University, Chicago, USA,1994.
14. Lamport, L. LATEX: A document preparation system, Ed. 2. Addison-Wesley, 1994.
15. Dudani, S.A. The distance-weighted k-nearest neighbour rule. *IEEE Trans. Systems, Man, and Cybernetics,* 1976, **SMC-6**(4), 325-27.
16. Information technology—Universal Multiple-Octet Coded Character Set (UCS), Pt 1. Architecture and basic multilingual plane. ISO/IEC 10646-1:1993.

Contributors

Shri Rajesh Bhatia obtained his BE in Computer Science and Engineering in 1994, and ME in Computer Science in 2001, respectively. His research interests are software reuse, software testing and component-based software engineering. Presently, he is working as Assistant Professor in the Department of Computer Science and Engineering at Thapar Institute of Engineering & Technology (Deemed University), Patiala. He is also Principle Investigator of a research project on Center of Excellence in Software Repositories.

Dr Mayank Dave obtained his BSc (Engg) from the Aligarh Muslim University in 1989, MTech from the University of Roorkee in 1991, and PhD from IIT, Roorkee in 2002. He joined Department of Computer Engineering at the National Institute of Technology, Kurukshetra (previously Regional Engineering College) in 1991 where he is continuing as Senior Lecturer. He has guided several BTech and MTech projects and dissertations and currently guiding two PhDs. He has published 30 papers in referred journals and conferences. He is member of IEEE, IEEE Computer Society, IETE, ISTE and Institute of Engineers (India). His research interests include computer networks, software engineering, and database systems.

Dr R.C. Joshi received his BE (Electrical Engineering) from the Allahabad University in 1967, and ME and PhD (Electronics and Computer Engineering) from the University of Roorkee in 1970 and 1980, respectively. He joined J.K. Institute, Allahabad University as Lecturer in 1967, and is presently Head of Institute's Computer Centre. He has published more than 50 research papers in national/international journals/conferences. Dr Joshi has also done 18 months training at ENSER Grenoble, France under Indo-French Collaboration Programme. He is recepient of *Gold Medal by Institute of Engineers* for best research paper in 1978. His research interests include parallel and distributed processing, artificial intelligence, DBMS, and wireless networks.