A Digraph-Based Approach to Component Retrieving

Chunxia Yang
School of Computer Science & Engineering, Xi'an University of Technology, Xi'an, China
Email: ycxxaut@163.com

Yinghui Wang
School of Computer Science & Engineering, Xi'an University of Technology, Xi'an, China
Email: wyh_925@163.com

***Abstract*—Component retrieval is important to improve software productivity in the field of component based software development (CBSD). In this paper, static and dynamic behavior information of component interface is considered as retrieval items for component retrieval system at the same time. And interface automaton is adopted as the model to describe retriever's query and component in repository. Three kinds of matching models are developed to satisfy exact or approximate matching according to the information retriever can give. The implementation of the matching is illustrated based on incidence matrix of digraph corresponding to interface automaton. A retrieving algorithm is developed in which offline computation of matching relationship in repository is used to reduce the searching space and amend the retriever's request.**

***Index Terms*—component retrieval, interface automaton, incidence matrix**

I. Introduction

Component-based software development (CBSD) is believed as a way of resolving some issues identified by the software crisis, which is to assemble software systems from pre-existing software components to reduce development costs and increase the quality of the final system [1]. Then, the storage and retrieval reusable component in a large scale repository is an important issue in the field of CBSD. It is believed that a good component retrieval method can effectively help users find the appropriate components from a large scale of component repository and improve the efficiency of software development.

Nowadays various component retrieval methods have been put forward. One of the most popular is facets based searching [2-5], which involving semantic searching based on ontology. Others associate with information retrieval technique [6,7] and AI algorithm [8], etc.. However, it requires a well understanding of characteristics of software components before reusing them [9]. Component is an encapsulated unit, and the only channel for component to interact with environment is its interface [10]. Component interface exposes the abstract specifications of component and describes the behavior of the component for users to a great extent. When using the reusable components to assemble software, the first thing is to check interface to decide whether the component matches the requirements or not [11]. Therefore, the information provided by interface is used as retrieval content for component retrieval naturally.

Additionally, we also concerned on Web Service, a special branch of components, which used for business level in architecture and developed more advanced than usual components. The specification of Web Service, BPEL, exposes internal process in behavioral aspects to meet the need of reusing service. And the process view changed the situation of Web Service retrieval [12,13]. Inspired by this, the study focuses on the behavior expressed by component interface to design retrieving scheme.

Earlier, the behavior information declared in component interface, such as operations and their types, pre-condition and post-condition, etc., is employed for component retrieval. The representative methods are signature matching [14,15] and specification matching [16,17]. Signature matching is a component retrieval method in which component is retrieved by its signature. The signature of a component is the union of all interfaces signatures that it defines, and the signature of an interface is the union of the operations' signatures it declares. If the retriever knows in advance the component signature, the approach will act well in retrieval system. Specification matching is a component retrieval method in which component is retrieved by its specification. Compared with component signature, a more tight constraint is appended in its specification, that is, the pre-condition and post-condition of an operation is included in operation's specification. Since component specification is always expressed in a formal language based on the predicates, specification matching usually has good results due to its mathematical rigor. But, the formal expression and the following equivalence proof also costs high overhead.

Later, the information implied in interface that changed before and after component running, such as the type of operations and the range of variables, is captured for component retrieval. Components are described by a set of tuples and each tuple represents a characteristic

doi:10.4304/jsw.9.9.2491-2498

input-output transformation of a component [18,19]. Retriever can enlarge or shrink the range of results by continually decreasing or increasing the number of tuples. The tuple has a great influence on the retrieval effect in this method, thus it is necessary for domain experts to provide a set of candidate tuples.

Mili *et al.*[20] use a pair (S , R) to describe the specification of a component, where S is the space of the variables that the component defines on, and it is structured as the cartesian product of named elementary spaces, R is a relation on S and describes the change of space before input and after output. The components in repository are constructed as lattices by the refinement relationship of R for certain S . When retriever inputs S , the corresponding lattice whose space is S is found. Then retriever inputs R , a vertex of the lattice will be fixed and the corresponding component is retrieved. All the components under this vertex in the lattice are also retrieved according to the refinement relationship.

The other type of dynamic behavior information implied in component interface is the invocation sequence of the operations declared in interface. Meng *et al.* give a specification matching method for business component [21]. A specification of business component is described in two levels. One is the business operation signature, including input business data types, output business data types and the taxonomy of business operations. The other is the invocation sequence of the operations, where the symbol "$\prec$" represents sequence relationship between business operations, and the symbol "&" represents concurrent relationship between business operations. Given two business components, the proportion of matching operations to the sum of all the operations determines the degree of signature similarity, and the proportion of matching operation sequences to the sum of all the action sequence determines the degree of action similarity. The weighted sum of these two similarity degrees concludes the similarity of the two components.

Through the above analysis of literature, interface behavior information can be divided into two types, static behavior information and dynamic behavior information. The former mainly included operations, the type of operations, parameters, the type of parameters, the operation signature and the pre/post conditions declared in interface. The latter mainly included the operation invoking sequence and the changes happened before and after the component running. The paper will develop component retrieving method based on static and dynamic behavior information at the same time.

Our study started from the model of component interface. In the recent years, many component models are proposed [22], and interface automaton [23] is chose here for three reasons. Firstly, interface automaton describes the operations declared in interface and their invoked sequence, which including both static and dynamic behavior information. Meanwhile, the operations can be extended to almost all of existed static behavior information. Secondly, comparing with usual formal methods, interface automaton is more intuitive and easier to use since it can also be represented as digraph. Thirdly, interface automaton theory discusses component composition completely, which facilitates the further judgment and usage of retrieved component.

Our previous work [24] has a preliminary exploration of component retrieving based on interface model. In that work, retrievers' requirement is expressed by a flow chart that is transformed into an automaton later. Component is indexed by the set of routes in its digraph. Component matching in fact is the matching of the route sets of the two digraphs, i.e., for every route of query automaton, there must be a matching route in the matching automaton.

In general, literature [21] and [24] design matching method by automaton language matching, and no implementation is discussed in both of them. In this paper, a component retrieving method implemented based on incidence matrix of digraph is proposed. The paper is outlined as follows. In section 2, query and component in repository are modeled by a slightly modified interface automaton. In section 3, three levels of matching are defined to satisfy the need of approximate matching. In section 4, we propose a definition of digraph inclusion relationship to implement the matching definition given in section 3. We also give corresponding algorithm and an example illustrates how the algorithm works in detail. Component retrieving algorithm is developed in section 5 and related component organization in repository is discussed. The advantage of the retrieving method is discussed in section 6. Finally, a brief conclusion and future work is described in the last section.

II. Models for Query and Component

In search and retrieval system, same descriptions model of query and elements of repository will be great help of the retrieval effect. In order to formally model query and component, we use interface automaton (IA) [23], which is a state-based model, similar to finite state diagrams, for representing behavior required by retriever's and component. For the sake of discussion, the IA model is slightly modified to apply to our retrieval purpose, as it is defined as follows.

Definition 1 An interface specification of a component is a deterministic finite automaton $M := \langle V, v_0, \Sigma, T \rangle$, where

V is a set of states,

$v_0 \in V$ is an initial state,

$\Sigma = \{\delta \mid \delta = <?/!, \text{OpName}, \text{OutType}, \text{InType}>\}$ is a set of operation signatures declared in interface, and each is composed of four tuples, in which the symbol "?/!" denotes the call direction of the operations, in the other words, it indicates that the operation is a request function or a provide function, OpName denotes the name of the operation, OutType denotes the type of output, and InType denotes a set of types of input parameters.

$T = \{v_i \times \delta \times v_j \mid v_i, v_j \in V\}$ is a set of steps.

We specify $|\Sigma|$ as the size of M.

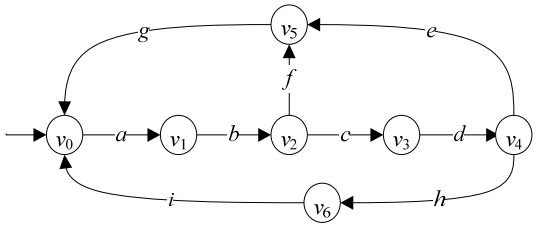


Figure 1. Interface automaton *M*.

Fig.1 is an example of IA. Obviously, the diagram of IA can be viewed as a digraph, in which the states, steps and operations of the automation correspond to the vertexes, directed edges and labels of edges of the digraph respectively.

Definition 2 (Ancestor-junior relationship) In an IA, if there exist some steps which constitute a continuous walk, e.g., $v_i \times \delta_i \times v_j$, $v_j \times \delta_j \times v_l$, …, $v_m \times \delta_m \times v_n$, then the operations of the previous step is called an ancestor of the operation of the following step, or the latter is called a junior of the former. For example, we call operation δ_i an ancestor of operation δ_j or operation δ_j a junior of operation δ_i, denoted by $\delta_i = ancestor(\delta_j)$ and $\delta_j = junior(\delta_i)$ respectively.

The ancestor-junior relationship is transitive. And we specify that, if there is a circle in IA, the operation arrived from v_0 before others is an ancestor of the rest in the circle.

III. Three Matching Levels

For simplicity, *M* is denoted as IA in component repository and *Q* is denoted as IA of query. Due to approximate matching is necessarily for retrieval systems to increase recall rate, three matching levels are elaborately designed.

Definition 3 (*Strong Constraint Matching*, SCM) If there exists a mapping $f: Q \to M$ satisfies the following three conditions, then f is called a *strong constraint matching*, and *M* is called a SCM component of *Q*:

(1) $f(\delta_i) = f(\delta_j) \Rightarrow \delta_i = \delta_j, \delta_i, \delta_j \in \Sigma_Q$.

(2) $\delta_i \equiv f(\delta_i)$, where the meanings of "≡" is shown as follows:

(a) The call direction of δ_i is the same to that of $f(\delta_i)$.

(b) $distance(\text{OpName}, f(\text{OpName})) \le w$, i.e., the semantic of OpName is the same to that of $f(\text{OpName})$, and w is the semantic similarity threshold set by retriever.

(c) The number and the output type of δ_i, i.e., OutType, is the same to that of $f(\delta_i)$.

(d) The number and the input types of parameters of δ_i, i.e., InType, are the same to that of $f(\delta_i)$ respectively.

(3) $\delta_i = ancestor(\delta_j) \Rightarrow f(\delta_i) = ancestor(f(\delta_i))$.

Definition 4 (*Strong Constraint Approximate Matching*, SCAM) If there exists a mapping $f: Q \to M$ satisfies the following three conditions, then f is called a *strong constraint approximate matching*, and *M* is called a SCAM component of *Q*:

(1) $f(\delta_i) = f(\delta_j) \Rightarrow \delta_i = \delta_j, \delta_i, \delta_j \in \Sigma_Q$.

(2) $\delta_i \approx f(\delta_i)$, some tuples of δ_i can be neglected, especially, the elements of *InType* can be neglected completely or partially. And for the given tuples of δ_i, "≈" means that:

(a) The call direction of δ_i is the same to that of $f(\delta_i)$.

(b) $distance(\text{OpName}, f(\text{OpName})) \le w$, i.e., the semantic of OpName is the same to that of $f(\text{OpName})$, and w is the semantic similarity threshold set by user.

(c) The output type of δ_i, i.e., OutType, is the same to that of $f(\delta_i)$.

(d) Each of the input types listed in InType of δ_i has a consistent item in InType of $f(\delta_i)$.

(3) $\delta_i = ancestor(\delta_j) \Rightarrow f(\delta_i) = ancestor(f(\delta_i))$.

Definition 5 (*Weak Constraint Approximate Matching*, WCAM) If there exists a mapping $f: Q \to M$, satisfies the following three conditions, then f is called a *weak constraint approximate matching*, and *M* is called a WCAM component of *Q*:

(1) $f(\delta_i) = f(\delta_j) \Rightarrow \delta_i = \delta_j$, $\delta_i, \delta_j \in sub(\Sigma_Q)$, where $sub(\Sigma_Q)$ is a subset of Σ_Q, and the percentage of $|sub(\Sigma_Q)| / |\Sigma_Q|$ is set by retriever.

(2) $\delta_i \approx f(\delta_i)$, the meanings of "≈" is the same to condition (2) of Definition 4.

(3) $\delta_i = ancestor(\delta_j) \Rightarrow f(\delta_i) = ancestor(f(\delta_j))$.

In these definitions, the first condition is to ensure the mapping is an injective mapping. The rest two conditions are operation signature matching and operation invoked sequence matching respectively, which are the two sides of behavior matching the paper mainly focuses on.

Obviously, the constraints of the three kinds of matching are weakening in order. And the difference of the three levels matching is manifested in the second condition, operation signature matching. Here, we do not intend to discuss operation signature matching in detail, but we conclude that, the matching of call direction is a symbol or character matching, the matching of OpName is a semantic matching, and the matching of OutType or InType is a kind of type matching. The loosing matching strategy results from the retrievers' imperfect knowledge of the requirements. And it is benefit for retriever to give the query flexibly based his/her assurance of the need.

IV. MATCHING IMPLEMENTATION

Since users cannot describe the query comprehensively due to the complexity of requirements, they are always demanded to describe no more than the behavior features he/she assures. For a query IA Q and a component IA M which satisfies query Q, there always has $|Q| \le |M|$. Moreover, as for the equivalence of IA and its diagram, the matching relationship of M and Q is translated into the inclusion relationship between their corresponding digraphs. Here we don't differentiate the symbolic representation of IA and its digraph, but the matching is actually implemented based on digraph. Next, the matching algorithm based on incidence matrix of digraph is described and an example is given.

A. Matching Algorithm

Since the matching is conceived based on the inclusion relationship between the corresponding digraphs of two IA, the digraph inclusion relationship and related concept are defined firstly.

Definition 6 (*Digraph inclusion relationship*) Given two digraphs $Q = \langle V_Q, E_Q, L_Q \rangle$ and $M = \langle V_M, E_M, L_M \rangle$, where V_Q, V_M are the vertex sets, and E_Q, E_M are the directed edge sets, and L_Q, L_M are the label sets of edges respectively. If there is an injective mapping from L_Q (or $Sub(L_Q)$) to L_M, such that for every mapping pair (l, l'), l' matches with l in meanings of operation signature matching. Moreover, for any two mapping pairs (l_1, l_1') and (l_2, l_2'), if $l_1 = ancestor(l_2)$, there always existing $l_1' = ancestor(l_2')$, then, we say digraph M including digraph Q.

Here, the label matching is corresponding to the operation signature matching of SCM, SCAM, and WCAM respectively. " $sub(L_Q)$ " is derived from " $sub(\Sigma_Q)$ " in condition (1) of WCAM. And the ancestor-junior relationship is kept. Therefore, the digraph inclusion relationship realizes all the three levels matching defined before. As the labels (operations in IA) of the edges are different from each other, we use the label to denote the directed edge.

Definition 7 (*Matching edges, Irrelevant edges*) Given two digraphs M and Q, For an edge $l' \in L_M$, if there exists an edge $l \in L_Q$, such that l' matches with l, then, we say l' is a *matching edge*; if there does not exist an matching edge in L_Q, then, l' is called an *irrelevant edge*.

We denote the set of *matching edge* as abbreviation *MatchingSet*, and the set of *irrelevant edge* as *IrrelevantSet*.

Here we give an implementation of component matching algorithm based on incidence matrix. Usually depth-first or width-first algorithm will be considered for dealing of digraph. However, since the emphasis is the edges in digraph rather than the vertexes, the work makes use of incidence matrix of digraph to determine whether the inclusion relationship exists between two digraphs.

The incidence matrix of IA M in Fig.1 is denoted as M_{Matrix}. For the convenience of description, edges labels and vertexes are marked on the heads of columns and rows of M_{Matrix} respectively. As to retriever, diagram is used to give his/her query and corresponding incidence matrix is got automatically from the diagram.

$$M_{Marix} = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{ccccccccc} a & b & c & d & e & f & g & h & i \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

When query Q and component M are both represented as incidence matrixes, the matching algorithm is given as follows.

Matching Algorithm

Input: Q_{Matrix}, M_{Matrix} and matching criteria SCM, SCAM or WCAM

Output: Successful or Failure

Firstly, matching the labels in matrix Q_{Matrix} and M_{Matrix};

If $L_Q \subseteq L_M$

//here " $\subseteq$ " is obtained by semantic matching of matching definitions in section 3; and for WCAM, the condition will be $sub(L_Q) \subseteq L_M$

For each $u \in IrrelevantSet$

and $M[v_1, u] = 1 \wedge M[v_2, u] = -1$

{

If $\exists w \in MatchingSet$

$M[v_1, w] = -1 \wedge M[v_2, w] = 0$

$M[v_2, w] \leftarrow -1$

Delete u and the column it located

}

Arrange the order of the elements in *MathcingSet* as that of the matching edges in L_Q

Else

Return fault

For each $w \in L_Q$ and its matching edge $w' \in MatchingSet$ in L_M

//the condition will be $w \in sub(L_Q)$ if WCAM is chose

{

If $Q[v_n, w] = 1 \wedge M[v_m, w'] = 1$ then

$v_n' \leftarrow v_m$

If v_n' already exists in the column

{

If $\exists\, j \in MatchingSet \wedge M[v_n', j] = M[v_m, j] = -1$

For $\forall\, i \in MatchingSet$

Compute $M[v_n', i] \leftarrow M[v_n', i] \oplus M[v_m, i]$

Else

Return fault

}

Else

Skip

Arrange the order of $v_n' \in M_{Matrix}$ in the column according to that of $v_n \in Q_{Matrix}$

If for $\forall\, Q[v_n, j] \neq 0$

$M[v_n', j'] = Q[v_n, j]$

Return successful

Else

Return fault

The algorithm is actually divided into four steps, and in the next subsection an example will be illustrated how the algorithm is implemented.

The computational complexity of graph traversal is $o\left(\max\left(|V_M| \cdot |V_M|, |V_M| \cdot |E_M|\right)\right)$. In the aspect of complexity, it looks like there is no great difference from DFS algorithm based on adjacency matrix, whose complexity is $o\left(|V| \cdot |V|\right)$. Noticed that, there involved semantic matching in the matching process, whose complexity cannot be ignored although no specific semantic matching algorithm is assigned in this paper. While the labels matching is repeated to adapted to a new choice of path in DFS, it only needs to compute one time to finish the whole matching process in our algorithm. That is the efficiency of our method.

B. Examples

Suppose the IA M in Fig. 1 is a component in repository, and Q_1 and Q_2 shown in Fig. 2 are query automatons. Here, we show how to determine M is a matching of Q_1 but Q_2.

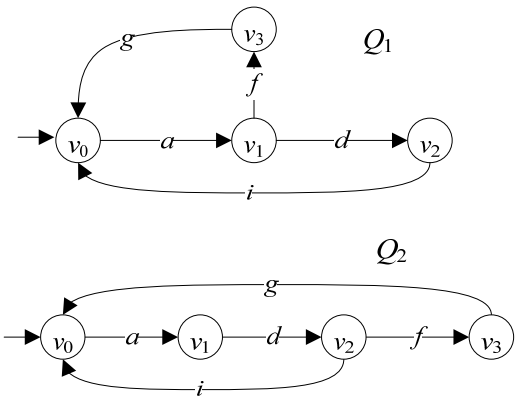


Figure 2. Query automaton Q_1 and Q_2.

Step 1 Suppose the kind of matching retriever chose is SCM. $Q_{Matrix1}$ and $Q_{Matrix2}$ are the incidence matrixes of Q_1 and Q_2 respectively. For simplicity, the same lowercase letters marked on the head of the columns stand for the matching edges.

$$Q_{Matrix1} = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccccc} a & d & g & f & i \end{array} \\ \begin{bmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix} \end{array}$$

$$Q_{Matrix2} = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccccc} a & d & g & f & i \end{array} \\ \begin{bmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix} \end{array}$$

Step 2 The connected feature of irrelevant edges, b, c, e, h, is extended to the matching edges and then these edges are deleted in M_{Matrix}. When all of the irrelevant edges are deleted and the matching edges are arranged in the same order as that appears in Q_{Matrix}, the result is got as M'_{Matrix}.

$$M'_{Matrix} = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{ccccc} a & d & g & f & i \end{array} \\ \begin{bmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Step 3 In $Q_{Matrix1}$, the start point of a is v_0, and the start point of the matching edge a in M'_{Matrix} is v_0, it means that vertex v_0 in M'_{Matrix} is corresponding to vertex v_0 in $Q_{Matrix1}$, so v_0 in M'_{Matrix} is modified with v_0'. In $Q_{Matrix1}$, the start point of d is v_1, and the start point of the matching edge d in M'_{Matrix} is v_3, it means that vertex v_3 in M'_{Matrix} is corresponding to vertex v_1 in $Q_{Matrix1}$, so v_3 in M'_{Matrix} is modified with v_1'. Analogously, v_5 and v_6 in M'_{Matrix} are modified with v_3' and v_2' respectively. As for vertex v_2 in M'_{Matrix}, it should be modified with v_1', but v_1' already exists, then the row fixed by v_2 is added to the row fixed by v_1', and we denote v_2 as (v_1'). The correspondence of vertexes of $Q_{Matrix1}$ to M'_{Matrix} is shown as $M''_{Matrix1}$. Finally, the order of the corresponding vertexes is arrange as same as the vertexes appear in $Q_{Matrix1}$, and the result is shown

as $M'''_{Matrix1}$. The similar result is obtained for $Q_{Matrix2}$ is $M''_{Matrix2}$, but the combination of v_2 and v_6 can not compute since the prerequisite is not hold. So the conclusion is obtained that M is not a match of Q_2.

$$M''_{Matrix1}=\begin{array}{c} \\ v_0\to v_0' \\ v_1 \\ v_2\to(v_1') \\ v_3\to v_1' \\ v_4 \\ v_5\to v_3' \\ v_6\to v_2' \end{array}\begin{array}{c} \begin{array}{ccccc} a & d & g & f & i \end{array} \\ \begin{bmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$M'''_{Matrix1}=\begin{array}{c} \\ v_0' \\ v_1' \\ v_2' \\ v_3' \\ v_1 \\ v_4 \end{array}\begin{array}{c} \begin{array}{ccccc} a & d & g & f & i \end{array} \\ \begin{bmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$M''_{Matrix2}=\begin{array}{c} \\ v_0\to v_0' \\ v_1 \\ v_2\to v_2' \\ v_3\to v_1' \\ v_4 \\ v_5\to v_3' \\ v_6\to(v_2') \end{array}\begin{array}{c} \begin{array}{ccccc} a & d & g & f & i \end{array} \\ \begin{bmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Step 4 The first four lines of $M'''_{Matrix1}$ is compared with $Q_{Matrix1}$ to check the ancestor-junior relationship, and the rest lines are redundant for the check. Clearly, for each nonzero value in $Q_{Matrix1}$, there is a same value in the corresponding position of $M'''_{Matrix1}$, then, we conclude the digraph M includes Q_1, then component interface M matches with query automaton Q_1.

More examples are verified and the conclusion can be confirmed by digraphs in Fig. 1 and Fig.2 directly, which states the correctness of the method.

V. Retrieving Algorithm

A. Repository Organization

There are two problems to be solved in a component library. One is how to build the component description, as well as index components on the basis of the description. Here, we suppose that diagrams of component and corresponding incidence matrixes are two kinds of indexes in repository. The former is showed for retriever to further verify the retrieval result, and the latter is used to check the inclusion relationship.

The other problem is how to classify those components in the library. Here, components in repository are classified into groups by the number of edges of corresponding digraph, and the matching relationship between them is established by applying the matching algorithm to each of component. For a component M, the number of its edges is denoted as |M|.

Additionally, the matching relationship between components in repository can be computed offline. The advantage of pre-computed matching relationship is that the related components can be obtained through the matching relationship in repository without compute once a matching component is found. Then the cost of retrieving all the matching components is largely less than the method of comparing component in repository one by one.

B. Retrieving Algorithm

As previously mentioned in section IV, the users gives the query with the content he/she assures to get more exact retrieving result. For a component and the related component in repository determined by matching algorithm, there is a transitivity relationship among them only if SCM and SCAM matching are chose. Therefore, considering the precision and recall ratio of searching, here SCAM matching result in repository is used in retrieving algorithm. And SCM can be used of course to increase precision ratio.

Retrieving algorithm

Input: user's query Q, where |Q|=k, and parameter *i*

Output: retrieving result *MResult*

Call the matching algorithm in component groups $N=k$

Return $MResult=\{M_1,M_2,\cdots,M_s\}$

Compute

$MResult=MResult\cup Match(M_1)\cup Match(M_2)\cup\cdots\cup Match(M_s)$

// $Match(M_i)$ is the SCAM matching result of M_i obtained by pre-computed matching relationship

For ($N=k+i$; $i=1$; $i++$)

{

If $M_j\in MResult$

Skip

Else

{

Call matching algorithm

If M_j matching successful

$MResult=MResult\cup M_j\cup Match(M_j)$

Else

Skip

}

}

Return *MResult*

Here, the algorithm searches the candidate components only from groups whose number of edges is from *k* to *k+i*.

The reason is the retriever already has a good knowledge of his/her with the developed part of software, and under this assumption, the number and correctness of information of operations given by retriever is closed to the true component. Therefore, a proper expansion of retriever's query could satisfy the need of searching effect and efficiency. The value of parameter *i* is determined by retriever, and he/she can modify it if the retrieving result is dissatisfied. If the value of *i* is increased to including all the groups in repository, then the result is equal to traversing search of repository.

The repository organization is in favor of ranking the retrieval result. The retrieved components are ranked by the number of edges, and they can be given in random sequence if their number of edges is same. And the digraph of component will be great helpful for users to verify quickly if the component is that he/she wants.

VI. Discussions

In this paper, IA is used as the model of component to discuss component retrieving, and three levels of matching definition are given to meet the need of users. Meanwhile, component matching algorithm and retrieving algorithm are proposed respectively. The advantage of the design is analyzed as follows.

Firstly, IA can describe both static and dynamic behavior information. It is a model that can give description of interface behavior comprehensively and intuitively. Meanwhile, the discussion of composition theory of IA in [23] provides a strong support for software assemble, which reduces the troublesome arisen by the inconsistent of model in following model checking and composition verification. It is seldom considered by most of literatures, and they usually viewed component retrieving as a single problem rather than a part of software product line.

Secondly, as we mentioned many times, the matching definitions and algorithms in this paper are proposed under a certain assumption, that is, users already have a proper knowledge of the component they wanted with the knowledge part of software that has been developed. This assumption is naturally and it increases the believability of the query given by retriever, which is part of the reason of local searching in design of retrieving algorithm.

Thirdly, though retrievers have assurance of their query to a large extent under our assumption, uncertainty is usually happened, too. Then three different levels of definitions are developed to meet the different assurance of retrievers. Though no specific semantic matching algorithm is assigned to the matching definitions, different matching levels are also helpful to restrict the range of result.

Next, most of literatures about component matching are discussion of method. A specific implementation algorithm of component matching is given in this paper based on incidence matrix of digraph. It is a further step for the applying of the method.

Lastly, repository organization, component index method and retrieving result method are discussed in this paper, and the retrieving algorithm is proposed. Different from traversal search in literatures [14-17, 21], it adopts a partial search strategy. The idea of utilization of repository organization is similar to [20], but the component model is different. Since the retrieving algorithm makes full use of the features of component organization and index methods, no bad influence will happen to retrieving effect, but, retrieving efficiency is increased.

VII. Conclusions

In the paper, a component retrieving method is proposed based on the incidence matrix of diagram. The paper starts from the model of component interface and IA is chose to describe the static and dynamic behavior information of interface. Three levels of matching are defined to adapt retriever's knowledge of requirement. Component matching is turned into digraph inclusion relationship which is implemented by incidence matrix. Moreover, component classification in repository, component index and how to rank the retrieving result is discussed. With the discussion of repository organization, a retrieving algorithm is developed. The most remarkable feature of the algorithm is that it traverses part of repository to get the candidate components and a quite of them are obtained from the offline matching compute of repository.

Since the paper gives full consideration of applicable of component model in CBSD when choosing IA as the model, the retrieving method can be integrated into software product line perfectly. In the future, platform will be built and tests will be conducted to checking the result analyzed in this paper.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 61100009), Shaanxi Province Major Project of Innovation of Science and Technology (No. 2009ZKC02-08), Shaanxi Province Department of Education Industrialization Training Project (No.09JC08) and Shaanxi Technology Committee Industrial Public Relation Project (No.2011K06-35).

References

[1] M. D. McIlroy, "Mass produced software components," *In NATO Software Engineering Conference*, P. Naur and B. Randell, Eds. Brussels.1968, pp.138-155.

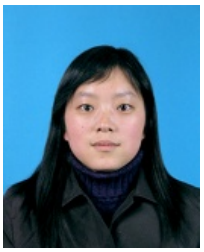
[2] L. Yanpei , G. Yuesheng and J. Chen, "Research on component retrieval methods," *Journal of Software*. vol. 7, pp.1633-1640, July 2012.

[3] W. Yuanfeng, Z. Yong, R. Hongmin, Z. Sanyuan and Q. Leqiu, "Retrieving components Based on Faceted Classification," *Journal of Software*. China, vol.13, pp. 1546-1551, 2002.

[4] W. Yuanfeng, X. Yunjiao, Z. Yong, Z. Sanyuan and Q. Leqiu, "A matching model for software component classified in faceted scheme," *Journal of Software*. China, vol. 14, pp. 401-408, 2003.

[5] M. Liang, X. Bing and Y. Fuqing, "The unified faced-based method to retrieve component in multi-library," *ACTA Electronica SinicaL*. China, vol. 30, pp. 2149-2152, 2002.

[6] L. Ge, Z. Lu, L. Yan, X. Bing and S. Weizhong, "Shortening retrieval sequences in browsing-based component retrieval using information entropy," *Journal of Systems and Software*. vol.79, pp.216-230, 2006.
[7] N. Sathit, S. Peraphon and E. R. William, "Fuzzy subtractive clustering based indexing approach for software components classification," *Proceedings of the 1st ACIS International Conference on Software Engineering Research & Applications (SERA'03)*, R. Y. Lee and K. W. Lee Eds. San Francisco, 2003, pp. 100-105.
[8] R. K. Bhatia, M. Dave and R. C. Joshi, "Ant colony based rule generation for reusable software component retrieval," *Proceedings of the 1st Conference on India Software Engineering Conference (ISEC'08)*, Hyderabad, 2008, pp.129-130..
[9] S. Mahmood, R. Lai and Y. Kim, "Survey component based software development," *IET Software*. vol.1, pp.57-66, 2007. Doi: 10.1049/iet-sen: 20060045.
[10] K. Wojtek. "Composite nature of component," *Proceedings of International Workshop on Component-Based Software Engineering*, I. Crnkovic, S. Larsson and J. Stafford, Eds. Los Angeles, 1999, pp.73-77.
[11] A. Y. Basem, "A precise characterization of software component interfaces," *Journal of Software*. vol. 6, pp.349-365, March 2011.
[12] E. Rik and G. Paul, "Structural matching of BPEL Processes," *Proceedings of 5th European Conference on Web Service (ECOWS'07)*, Halle, 2007, pp.171-180, Doi: 10.1109/ECOWS.2007.22.
[13] M Bouzeghoub, D. Grigori and A. Gater, "A Graph-based approach for semantic process model discovery," *Graph Data Management: Techniques and Applications*. S. Sakr, & E. Pardede, Eds. Hershey, 2012, pp. 438-462, Doi:10.4018/978-1-61350-053-8.ch019.
[14] A. M. Zaremski and J. M. Wing, "Signature matching: a tool for using software libraries," *ACM Trans. Softw. Eng. Methodol*. vol.4, pp.146–170, 1995.
[15] A. M. Zaremski and J. M. Wing, "Specification matching of software components," *ACM Trans. Softw. Eng. Methodol*. Vol.6, pp.335-369, 1997.
[16] A. M. Zaremski and J. M. Wing, "Signature matching: a k ey to reuse," *Proceedings of 1st ACM SIGSOFT Symposium on the Foundations of Software Engineering*, N. David, Eds. Los Angeles, 1993, pp.182-190.
[17] D. Hemer and P. Lindsay. "Specification-based retrieval strategies for module reuse," *Proceedings of Australian Software Engineering Conference*, G. D. Douglas and S. Leon, Eds. Canberra, 2001, pp. 235-243, Doi:10.1109/ASWEC.2001.948517.
[18] J. Sametinger, *Software Engineering with Reusable Components*, Springer-Verlag, 1997.
[19] R. T. Mittermeir and H. Pozewaunig, "Classifying components by behavioral abstraction," *Proceedings of 4th Joint Conference on Information Sciences*, W. P. Paul, Eds. North Carolina, 1998, pp. 547-550.
[20] R. Mili, A. Mili, and R. T. Mittermeir, "Storing and retrieving software components: a refinement based system," *IEEE Trans. Softw. Eng.*. vol.23, pp.445-460, Jul 1997.
[21] M. Fanchao, Z. Dechen and X. Xiaofei, "A specification-based approach for retrieval of reusable business component for software reuse." *World academy of science, engineering and technology*. vol.15, pp.240-247, 2006.
[22] K. K. Lau and Z. Wang, "Software component models." IEEE Trans. *Softw. Eng.*. vol.33, pp.709-724, Oct 2007.
[23] L. de. Alfaro and T.A. Henzinger. "Interface automata," *Proceedings of the joint 8th European Software Engineering Conference and the 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Vienna, 2001, pp.109-120. Doi: 10.1145/503209.503226.
[24] W. Yinghui and Y. Chunxia, "A perfect design of component retrieval system," *Information*. vol.15, pp.1687-1704, 2012.



Chunxia Yang, is a phD student at Xi'an University of Technology in School of Computer Science & Engineering, Xi'an University of Technology. She received her M.S. degree in 2007 from School of Science, Xi'an University of Technology. Her main research interests include software component retrieval, component composition and deployment in CBSE.

Yinghui Wang, is a professor of School of Computer Science & Engineering, Xi'an University of Technology, China. He received his B.S., M.S., and PhD degrees in 1989, 1999, and 2002, respectively. He is a senior member of China Computer Society (CCF). He has a long software development and maintenance experience in oil field systems. His research interests include software development, software evolution and pattern recognition.