International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:3, 2008

A Specification-Based Approach for Retrieval of Reusable Business Component for Software Reuse

Meng Fanchao, Zhan Dechen, and Xu Xiaofei

Abstract**—**Software reuse can be considered as the most realistic and promising way to improve software engineering productivity and quality. Automated assistance for software reuse involves the representation, classification, retrieval and adaptation of components. The representation and retrieval of components are important to software reuse in Component-Based on Software Development (CBSD). However, current industrial component models mainly focus on the implement techniques and ignore the semantic information about component, so it is difficult to retrieve the components that satisfy user's requirements. This paper presents a method of business component retrieval based on specification matching to solve the software reuse of enterprise information system. First, a business component model oriented reuse is proposed. In our model, the business data type is represented as sign data type based on XML, which can express the variable business data type that can describe the variety of business operations. Based on this model, we propose specification match relationships in two levels: business operation level and business component level. In business operation level, we use input business data types, output business data types and the taxonomy of business operations evaluate the similarity between business operations. In the business component level, we propose five specification matches between business components. To retrieval reusable business components, we propose the measure of similarity degrees to calculate the similarities between business components. Finally, a business component retrieval command like SQL is proposed to help user to retrieve approximate business components from component repository.

Keywords**—**Business component, business operation, business data type, specification matching.

I. Introduction

COMPONENT-BASED Software Development (CBSD) is a key technology to tackling rapid development and software reuse of Enterprise Information System. CBSD is different from traditional methodology of software development, it emphasis much on retrieving reusable components from components repository, and these components retrieved are assembled to realize the functions of application system. However, current industrial component models such as CORBA, EJB and COM/DCOM mainly focus on the implement techniques and ignore component semantic information, so it is difficult to retrieve reusable components according to the representation provided by these component models. The main reason is that the information useful in reuse process is either implicitly represented, which requires extensive program analysis, or not formally represented, which hinders the possibility of formal analysis [1].

Component retrieval involves component representation, component classification and component searches. Currently, many component retrieval methods have been proposed, including text retrieval, facet-based retrieval and specification-based matching retrieval.

Text retrieval method use one or more key words represent components[2][3]. This approach is easy to understand and is well defined. However, it has no ability to describe complex semantic information[4]. Aim to the problem, some researchers use fuzzy mathematics and rough-fuzzy sets to retrieve components [5][6][7].

Facet-based retrieval method is a reuse approach that is widely accepted [8]. In this approach, a component is classified and searched using facets, and each facet includes some terms that describe component semantic information. In the faced scheme, a thesaurus provides vocabulary control, and a conceptual distance graph is used to evaluate the similarities between terms[9]. The main problems of facet-based retrieval are that system with large number of facets can't be used efficiently, and constructing a thesaurus and conceptual distance graph is labor-intensive. In addition, facet scheme is still not formalized so that can't effectively guide components retrieval and assembly[10].

Specification matching retrievals focus on the type information about the interface of a software component, they take advantage of formal techniques to describe components. Currently, a lot of research on Specification matching for software reuse has been proposed [11][12][13][14][15][16]. Formal techniques have a good mathematical basic for component specification. They emphasize on the completeness, preciseness and consistency, which are suitable to component retrieval and assembly. However, specification matching

Manuscript received July 2, 2006. This work was supported in part by the National Natural Science Foundation of China under Grant No.60573086 and Research Foundation for the Doctoral Program of Higher Education of China under Grant No.20030213027

Meng Fanchao is with the school of computer science and technology, Harbin institute of technology, Harbin, CO150001 China (corresponding author to provide phone: 86-41-86412664; fax: 86-0451-86412664; e-mail: mengfanchao74@ 163.com).

Zhan Dechen is with the school of computer science and technology, Harbin institute of technology, Harbin, CO150001 China (corresponding author to provide phone: 86-41-86412664; fax: 86-0451-86412664; e-mail: Denchen@hit.edu.cn).

Xu Xiaofei is with the school of computer science and technology, Harbin institute of technology, Harbin, CO150001 China.

retrievals require users know deeply about the problem domains[17].

In this paper, we propose a component retrieval approach based on approximate specification matching. The application background is a large-scale business component repository of enterprise information system. A business component is used in the context of an information system as a part of the system's architecture. Business components can satisfy the functional requirements of information systems by providing services [18].

The remainder of this paper is organized as follows. Section 2 discusses related works. Section 3 describes a business component model that mainly focuses on the business operations provided by business components. In section 4, we propose a business data type based on XML, give three matches, they are business data types match based on paternity, business operations match based on function specification, and business components match based on five specifications. Section 5 proposes a method of business component retrieval based on specification. In this section, we propose signature and action similarity degrees to calculate the similarities between business components, and present a business component query command that has the similar semantic as the conventional SQL. Finally, section 6 describes conclusion and future work.

II. Related Work

Specification matching methods for component retrieval have been frequently addressed in the software engineering literature.

B.H.C.Cheng etc[18] propose a two-tiered hierarchy of the repository based on formal specification using OSPL. The lower level is based on generality relationships, and the higher one is based on similarity relationships, which are assessed by a clustering algorithm.

In [19][20], approximate retrieval of incomplete and formal specifications is proposed. The classification and retrieval are based on functional similarities according to structural and semantic closeness. This paper defines four partial orderings among reusable components and different measures to quantify functional differences among them.

Hai Zhuge[21] presents a model retrieval approach based on a quantified similar signature matching. The application background is a mathematical model repository which is divided into two specification levels: a fundamental function level and a higher model level. By defining a multi-valued model specification relationship and a function specification relationship at two levels, the similarity between repository models is characterized.

In [22], propose an XML-based software component specification, the component retrieval method and a system can deal with exact, partial and reference matching.

Mili et al[23] presented a software library approach based on relational specification of components and queries as well as on an ordering of library components, done by a refinement ordering relation. Zaremski and Wing[24][25] applied specification matching to retrieval system. Queries and components are represented with pre-condition and post-condition pairs.

III. Business Component Model Oriented Reuse

A. Business Component

Component based software development should be based on Software Architecture (SA). Currently, Hierarchical Software Architectures are used broad in Enterprise Information System. Oliver Sims [26] proposed four-layer architectures; they are user layer, workshop layer, enterprise layer and resource layer. A larger-grained component called business component can span this four layers. Business components can provide services that satisfy the business requests of enterprise information systems. In this paper, we are interested in these business components that lie in enterprise layer.

Definition 1: Business components can be defined as $bc=(n, PS, RS, AS)$, where

(1) n is the name of business component.

(2) $PS=\{PI_1,PI_2,\ldots,PI_m\}$ is the set of provide interfaces, each PI_i (i=1,2,…,m) is composed of a set of provide business operations.

(3) $RS=\{RI_1,RI_2,\ldots,RI_n\}$ is the set of require interfaces, each RI_j (j=1,2,…,n) is composed of a set of require business operations.

(4) AS is action semantic of the business component [27]. It can be represented as a event partial-order multi-sets, denoted as $AS=[(V, BOP, \prec, \mu)]$, where, V is the set of events that represent activation of each business operations implemented by the business component or an external call from the business component; $BOP=PI_1 \cup PI_2 \cup \ldots \cup PI_m \cup RI_1 \cup RI_2 \cup \ldots \cup RI_n$ is the set of business operations in the business component's interfaces; $\prec$ is an irreflexive transitive binary relation on E; $\mu : E \rightarrow BOP$ is a mapping function, it assign business operations to events, each element of the event set represents an occurrence of business operation labeling it, with the events possibly having multiple occurrences, that is, μ need not be injective. AS can be expressed as a concurrent regular expression on BOP.

Definition 2: A business operation can be defined as $bop=(n,t,In,Out)$, where, n is the name of business operation, t is the type of business operation, In is the set of input business data type, Out is the set of output business data type.

In this paper, we adopt an approximate method to describe a business component. It mainly focuses on the input and output business data types of business operations in the interfaces of business components. Business data types are abstracted from business objects in domain business model. It can be represented as an extended DTD that can express the variable business data that can describe variety of business operations.

B. Business Data Type based on XML

Current most component models adopt still traditional data structure used in program language to represent business data type. However, this method will suffer the influence of weak interfaces with the increase of complexity of information systems. An approach to solve this problem is utilizing XML represent business data. Today XML has been used generally to

represent all kinds of data type. This paper uses a set of XML to represent business data.

When business components deal with business data, they need to check the syntax of business data, and then abstract data item according to their names. In general, the order of data items should be ignored. In addition, the business operations in business component's interfaces should have the ability to deal with the variability of input and out parameters of business operations, however current most component models don't consider this aspect.

In this paper, we use DTD to represent the business data type. Here, we only consider elements (that can have a nested structure) ignoring attributes. The variability of business data Type can be described by the operators in DTD such as "?", "+" and "*" etc. To standardize the representation of DTD describing business data type, compound operators need be predigested, for example, ((a*) +) = a*. Because DTD describing business data type doesn't distinguish the sequence of elements, this is different from stander DTD criterion. To express this requirement, we use symbol DTD^+ represent business data type, and use symbol XML^+ represent business data.

Definition 2: Let D be a DTD^+, X be a XML^+, if X satisfy the format of D, then X is called as an instance of D, denoted as $X \in Instance(D)$, where $Instance(D)$ is the set of instance of D.

An example of DTD^+ and XML^+ is shown as Fig. 1. Fig. 1(a) is a DTD^+ that describes business data type of a check order. Fig. 1(b) is a XML^+ that satisfies the format of DTD^+ in fig1(a), and Fig. 1(c) is also a XML^+ that satisfies the format of DTD^+ in Fig. 1(a). According to the definition 2, the $XML1^+$ and XML2+ in Fig. 2(b) and Fig. 1(c) are the instances of DTD^+ in Fig. 1(a), but that have different structure.

.

```
<! ELEMENT Check order (Number, Standard, Object,
(Requisition?|Instockbill?), QualityRate?, Checker*,
CheckItems*)>
<! ELEMENT Number (#PCDATA)>
<! ELEMENT Standard (#PCDATA)>
<! ELEMENT Object (#PCDATA)>
<! ELEMENT Requisition (#PCDATA)>
<! ATTLIST Requisition DataType CDATA "String">
<! ELEMENT Instockbill (#PCDATA)>
<! ELEMENT QualityRate (#PCDATA)>
<! ELEMENT Checker (#PCDATA)>
<! ELEMENT CheckItems (Item, Unit, Value)>
<! ELEMENT Item (#PCDATA)>
<! ELEMENT Unit (#PCDATA)>
<! ELEMENT Value (#PCDATA)>
```

(a) DCheckorder

```
<Check order>
  <Number> ZJ001 </Number>
  <Standard> GB-2000 </Standard>
  <Object> coal </Object>
  < Requisition > 05001 </ Requisition >
  <Checker> Tom </Checker>
  <CheckItems>
    <Item> water </Item>
    <Unit> % </Unit>
    <Value> 10 </Value>
  </CheckItems>
</Check o
```

(b) XCheckorder1

```
<Check order>
  <Number> ZJ002 </Number>
  <Standard> GB-2000 </Standard>
  <Object> coal </Object>
  < Requisition > 05001 </ Requisition >
  <QualityRate>2</QualityRate>
  <Checker> Tom </Checker>
  <CheckItems>
    <Item> water </Item>
    <Unit> % </Unit>
    <Value> 10 </Value>
  </CheckItems>
</Check order>
```

(c) XCheckorder2

Fig. 1 An example of DTD^+ and XML^+

C. Business Component Model

Business components can be identified and created from domain business model that can be represented as UML class diagrams. A business component can implement the functions of one or more business objects which are represented by classes. For example, Fig. 2 shows a domain business model that includes five business objects.

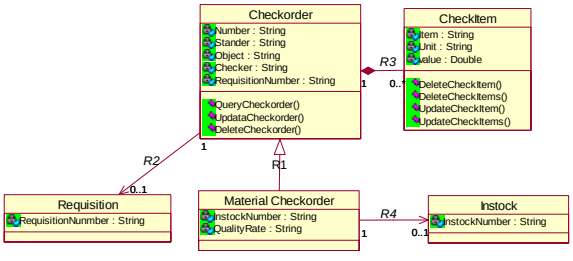


Fig. 2 Domain Business Model

Base on domain business model, we can identify reusable business components for across systems. Here, we give a business component (CCheckorder) which provides four

business operations, they are QueryCheckorder, UpdateCheckorder, RequsitiontoCheckorder and UpdateCheckorderItems. The business component specification is shown as follows:

```
Business Component CCheckorder
  Provide Interfaces
    Interface ICheckorder
      Business Operation GetCheckorder
        Type="Query";
        Input
          DQuery;
        End Input
        Output
          DResult;
        End Output
      End Business Operation;
      Business Operation UpdateCheckorder
        Type="Update";
        Input
          DCheckOrder;
        End Input
        Output
        End Output
      End Business Operation;
      Business Operation RequsitiontoCheckorder
        Type="Transformation";
        Input
          DRequsition;
        End Input
        Output
          DCheckOrder;
        End Output
      End Business Operation;
      Business Operation UpdateCheckorderItems
        Type="Update";
        Input
          DCheckOrderItem;
        End Input
        Output
        End Output
      End Business Operation;
    End Interface
  End Interfaces
  Action
    [(GetCheckorder)&(UpdateCheckord ≺ UpdateCheckorderItems)&(RequsitiontoCheckorder)]
  End Action
End
```

In the above action semantic specification, the symbol "≺" represents sequence relationship between business operations, and the symbol "&" represents concurrent relationship between business operations. The input and output business data type of every business operation can be abstracted from domain business model. Here, we give an example of business data type to explain the difference between business data type based on XML and traditional data type.

The business operation *QueryCheckorder* can be represented as *GetCheckorder*(*String Number*, *String Stander*): *Checkorder* in current most component models. In our approach, the input parameters of business operation can be transformed into DTD^+ shown as in Fig. 3(a), and the output parameters of business operation can be transformed into DTD+ shown as in Fig. 3(b). Comparing with current component models, our approach has some advantages as follows:

- It can describe the variability of input and out parameters of business operations, which can not be represented in many component models.
- It can not only describe static structure of a business component, but also describe dynamic action feature..
- It is independent to implement platform and development language.

```
<! ELEMENT Check order (Number, Standard >
<! ELEMENT Number (#PCDATA)>
<! ELEMENT Object (#PCDATA)>
```

(a) DQuery

```
<! ELEMENT Check order (Number, Standard, Object,
(Requisition | Instockbill), QualityRate, Checker*>
<! ELEMENT Number (#PCDATA)>
<! ELEMENT Standard (#PCDATA)>
<! ELEMENT Object (#PCDATA)>
<! ELEMENT Requisition (#PCDATA)>
<! ELEMENT Instockbill (#PCDATA)>
<! ELEMENT QualityRate (#PCDATA)>
<! ELEMENT Checker (#PCDATA)>
```

(b) DResult

```
<! ELEMENT Check order (Number, Standard, Object,
Requisition?, Checker*, CheckItems*)>
<! ELEMENT Number (#PCDATA)>
<! ELEMENT Standard (#PCDATA)>
<! ELEMENT Object (#PCDATA)>
<! ELEMENT Requisition (#PCDATA)>
<! ELEMENT Checker (#PCDATA)>
<! ELEMENT CheckItems (Item, Unit, Value>
<! ELEMENT Item (#PCDATA)>
<! ELEMENT Unit (#PCDATA)>
<! ELEMENT Value (#PCDATA)>
```

(c) DCheckorder'

Fig. 3 Example of Business Data Type

IV. Business Component Retrieval based on Specification Matching

Evaluation of business component similarity is based on the specification match relationships in two levels: business operation level and business component level. In business operation level, we use input business data types, output business data types and the taxonomy of business operations

evaluate the similarity between business operations. In the business component level, we propose five specification matches between business components.

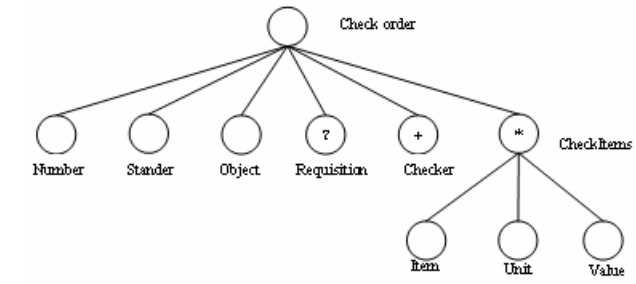
A. Matching between Business Operations

Definition 3: Let D_1 and D_2 be two DTD$^+$, if for every $X \in Instance(D_1)$ such that $X \in Instance(D_2)$, then D_1 is called as a subtype of D_2, denoted as $D_1 \subseteq D_2$. If $(D_1 \subseteq D_2) \wedge (D_2 \subseteq D_1)$, then D_1 is equivalent to D_2, denoted as $D_1 \equiv D_2$.

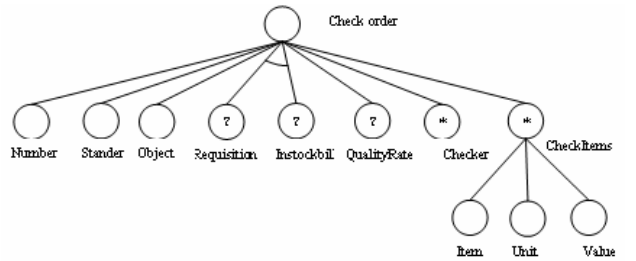
$D_1 \subseteq D_2$ denotes that the instance set of D_2 contains the instance set of D_1, that is to say, the expression ability of D_2 is stronger than that of D_1. Fig. 3(c) gives another DTD$^+$. According to the definition 2, the XML$^+$ in fig 1(a) is also an instance of the DTD$^+$ in Fig. 3(c), and for every instance that satisfies the format of DTD$^+$ in Fig. 3(c), it also satisfies the format of DTD$^+$ in Fig. 1(a), so the DTD$^+$ in Fig. 3(c) is a subtype of the DTD$^+$ in Fig. 1(a).

To judge the paternity between two business data types, we map each DTD$^+$ into an unorder labeled tree, and then take advantage of the matching relationship between two unorder labeled trees to judge the paternity between two business data types.

Definition 4: A DTD$^+$ can be represented as an unorder labeled tree $T=(V,E,root(T))$, where, V is the set of nodes, and each node represents an element, $root(V)$ is the root node of labeled tree. E is the set of edges, and $(u,v) \in E$ represents that u is the father node of v, denoted as $u=Parent(v)$. $u=Parent?(v)$ represents that v is the optional element of u. $u=Parent\%(v_1,v_2,\ldots,v_n)$ represents that there will be an element selected from $v_1,v_2,\ldots,v_n$. $u=Parent+(v)$ represents that v can repeat 1 or more times. $u=Parent*(v)$ represents that v can repeat 0 or more times. $u=Parent1(v)$ represents that v can't be repeated and absent.



(a) Label Tree 1



(b) Label Tree 2

Fig. 4 An Example of label trees

For example, Fig. 4(a) gives a unorder labeled tree that represents the DTD$^+$ in Fig. 3, and Fig. 4(b) gives another unorder labeled tree that represents the DTD$^+$ in Fig. 1(a).

Firstly, we discuss the method of judging the paternity between two unorder labeled trees whose depths are one. Let $T_1=(V_1,E_1,root(T_1))$ and $T_2=(V_2,E_2,root(T_2))$ be two unorder labeled trees, $Depth(T_1)=1$, $Depth(T_2)=1$, if T_1 and T_2 satisfy the following conditions, then $T_1 \subseteq T_2$.

Condition 1: $label(root(T_1)) \sim label(root(T_1))$, where $\sim$ represents that the tag name of $root(T_1)$ and the tag name of $root(T_2)$ are synonymous.

Condition 2: for every element v in $V_1-\{root(T_1)\}$, there exist an element v' in $V_2-\{root(T_2)\}$, such that:

- $label(v) \sim label(v')$;
- $u=parent1(v) \Rightarrow u'=parent1(v') \vee u'=parent?(v') \vee u'=parent+(v') \vee u'=parent*(v')$, where u is the father node of v, and u' is the father node of v';
- $u=parent?(v) \Rightarrow u'=parent?(v') \vee u'=parent*(v')$;
- $u=parent+(v) \Rightarrow u'=parent+(v') \vee u'=parent*(v')$;
- $u=parent*(v) \Rightarrow u'=parent*(v')$;

Condition 3: if there exists an element v' in $V_2-\{root(T_2)\}$, but there dose not exist element v in $V_1-\{root(T_1)\}$ such that: $label(v) \sim label(v')$, then

- $u'=parent?(v') \vee u'=parent*(v') \vee u'=parent\%(v_1',v_2',\ldots,v',\ldots,v_n')$.
- if there exists relation $u'=parent\%(v_1',v_2',\ldots,v',\ldots,v_n')$, suppose that for each element in $v_1',v_2',\ldots,v_k'$, there dose not exist element v_i $(i=1,2,\ldots,k)$ in $V_1-\{root(T_1)\}$ that satisfies conditions: $label(v_i) \sim label(v_i')$, and for each element in $v_{k+1}',v_{k+2}',\ldots,v_n'$, there exist element v_j $(j=k+1,k+2,\ldots,n)$ in $V_1-\{root(T_1)\}$ that satisfies conditions: $label(v_i) \sim label(v_i')$. if $k \geq 2$, then there exist relation $u=parent\%(v_1,v_2,\ldots,v_k,\ldots,v_m)$, $label(v_i) \sim label(v_i')$ $(i=1,2,\ldots,k)$.

For two unorder labeled trees whose depths are bigger than one, we can use above approach and width search technique to judge the paternity between them. In the following, we illustrate the method by an example.

Fig. 5 shows the mapping relationships between two unorder labeled trees T_1 and T_2 that are the labeled trees shown in Fig 4. In $Layer_0$, $label(T_1)=label(T_2)$. In $Layer_1$, for every element v in T_1, there exists an element v' in T_2, they satisfy condition 2. Fig 5 gives the map relationships from T_1 to T_2.For the node *Instockbill* and *QualityRate* in T_2, there does not exist corresponding elements in T_1, but they satisfy condition 2. In T_1, the element *CheckItem* includes three son elements, and in T_2, there exist also corresponding elements that satisfy above conditions. So DTD$^+$ represented by T_1 is a subtype of DTD$^+$ represented by T_2.

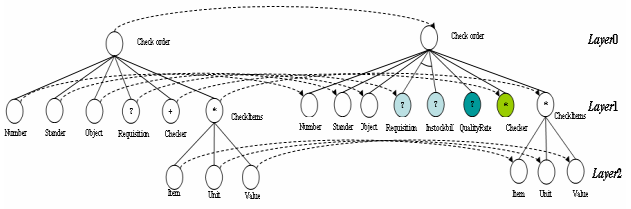


Fig. 5 Label Tree Matching

Definition 5: Let $bop_1=(n_1,t_1,In_1,Out_1)$ and $bop_2=(n_2,t_2,In_2,Out_2)$ be two business operations, if bop_1 and bop_2 satisfy condition: $(t_1=t_2) \wedge (In_1 \subseteq In_2) \wedge (Out_1 \subseteq Out_2)$, then bop_2 is called as the specification of bop_i, denoted as $bop_1 \rightarrow_S bop_2$.

Theorem 1: Specification matching between two business operations satisfies reflexive and transitive, that is, (1) $bop \rightarrow_S bop$; (2) $bop_1 \rightarrow_S bop_2 \wedge bop_2 \rightarrow_S bop_3 \Rightarrow bop_1 \rightarrow_S bop_3$.

Let bop_1 and bop_2 be two business operations, $bop_1 \rightarrow_S bop_2$, denotes that the service provided by bop_2 is stronger than the service provided by bop_1. Suppose bop_1 and bop_2 are two query business operations. The input business data type of bop_1 is the DTD+ in Fig. 6(a), and the output business data type of bop_1 is the DTD^+ in Fig. 2(a). The input business data type of bop_2 is the DTD^+ in Fig. 3, and the output business data type of bop2 is the DTD+ in Fig. 6(b). According to the method of judging the paternity between business data types, we have $In_1 \subseteq In_2$, $Out_1 \subseteq Out_2$, thus $bop_1 \rightarrow_S bop_2$.

```
<! DOCTYPE Check order [
<! ELEMENT Check order (Number, Standard, Object)>
<! ELEMENT Number (#PCDATA)>
<! ELEMENT Standard (#PCDATA)>
<! ELEMENT Object (#PCDATA)>]>
```

(a) $DTD1^+$

```
<! DOCTYPE Check order [
<! ELEMENT Check order (Number, Standard?, Object?)>
<! ELEMENT Number (#PCDATA)>
<! ELEMENT Standard (#PCDATA)>
<! ELEMENT Object (#PCDATA)>]>
```

(b) $DTD2^+$

Fig. 6 Two DTD^+s

B. Matching between Business Components

From the standpoint of reuse, a business component can be represented as set of business operations. Every business operation is a signature of the business components. Here we give five matching relationships between business components.

Definition 6: Let bc_1 and bc_2 be two business components, $BOP(bc_1)$ is the set of business operations provide by bc_1, and $BOP(bc_2)$ is the set of business operations provided by bc_2.

1) If there exist a one-to-one and onto mapping from $BOP(bc_1)$ into $BOP(bc_2)$, and for every $bop_i \in BOP(bc_1)$, there exists a $bop_j \in BOP(bc_2)$ such that $bop_i \rightarrow_S bop_j$, then bc_2 is called an equivalent specification from bc_1, denoted as $bc_1 \rightarrow_{Equi} bc_2$;

2) If for every $bop_i \in BOP(bc_1)$, there exists a $bop_j \in BOP(bc_2)$ such that $bop_i \rightarrow_S bop_j$, then bc_2 is called an extension specification from bc_1, denoted as $bc_1 \rightarrow_{Extend} bc_2$;

3) If for every $bop_j \in BOP(bc_2)$, there exists a $bop_i \in BOP(bc_1)$ such that $bop_i \rightarrow_S bop_j$, then bc_2 is called a partial specification from bc_1, denoted as $bc_1 \rightarrow_{Part} bc_2$;

4) If there exists a $bop_i \in BOP(bc_1)$ and a $bop_j \in BOP(bc_2)$ such that $bop_i \rightarrow_S bop_j$, then bc_2 is called a modification specification from bc_1, denoted as $bc_1 \rightarrow_{Modi} bc_2$.

5) If there does not exist a $bop_i \in BOP(bc_1)$ and a $bop_j \in BOP(bc_2)$ such that $bop_i \rightarrow_S bop_j$, then bc_2 is called a non-specification from bc_1, denoted as $bc_1 \rightarrow_{Non} bc_2$.

In above five matching relationships, the equivalent specification is the strongest, both the extension specification and the partial service specification are weaker than equivalent service specification and stronger than the modification service specification, and the non- specification is the weakest.

V. Business Component Retrieval

A. Similarity Degree

Reusable business components are stored in the repository. To retrieve the suitable business component form the repository, I give the rule of measurement that can evaluate the similarity degree between two business components.

- **Signature Similarity Degree:**

Let bc_1 and bc_2 be two business components, the signature similarity degree between bc_1 and bc_2 can be defined as $SSD(bc_1,bc_2)=\frac{2\cdot|BOP(bc_1 \cap bc_2)|}{BOP(bc_1)\cap BOP(bc_2)}$,

where $BOP(bc_1 \cap bc_2)$ represents the set composed of the pairs of business operations of bc_1 and bc_2 that satisfy specification matching, that is, $BOP(bc_1 \cap bc_2)=\{(bop_{1i}, bop_{2j}) | bop_{1i} \in BOP(bc_1), bop_{2j} \in BOP(bc_2), bop_{1i} \rightarrow_S bop_{2j}\}$. $BOP(bc_1)$ and $BOP(bc_2)$ are the sets of business operations included in bc_1 and bc_2.

- **Action Similarity Degree:**

Action similarity degree between two business components can be calculated by the action semantic of business components. The action semantic can be expressed as a concurrent regular expression on business operation which is decomposed into the disjoint set of partial order business operations. Here we call every business operations set as a business operation sequence.

Definition 7: Let $p_1=bop_{11} \prec bop_{12} \prec \ldots \prec bop_{1m}$ and $p_2=bop_{21} \prec bop_{22} \prec \ldots \prec bop_{2n}$ be two business operation sequences, if p_1 and p_2 satisfy conditions: (1) $m=n$; (2) $bop_{1i} \rightarrow_S bop_{2i}$ (i=1,2,…,m), then p_2 is called as the specification of p_1, denoted as $p_1 \rightarrow_S p_2$.

Let bc_1 and bc_2 be two business components, the action similarity degree between bc_1 and bc_2 can be defined as $ASD(bc_1,bc_2)=\frac{2\cdot|P(bc_1 \cap bc_2)|}{P(bc_1)\cap P(bc_2)}$,

where $P(bc_1 \cap bc_2)$ represents the set composed of the pairs of business operation sequences of p_1 and p_2 that satisfy specification matching, that is, $P(bc_1 \cap bc_2)=\{(p_{1i}, p_{2j}) | p_{1i} \in P(p_1), bop_{2j} \in P(bc_2), p_{1i} \rightarrow_S p_{2j}\}$. $P(bc_1)$ and $P(bc_2)$ are the sets of business operation sequences included in bc_1 and bc_2.

- **Similarity Degree between business components:**

Let bc_1 and bc_2 be two business components, the similarity degree between bc_1 and bc_2 can be defined as $SD(bc_1,bc_2)=w_S \cdot SSD(bc_1,bc_2)+w_A \cdot ASD(bc_1,bc_2)$, where $w_S \in [0,1]$ is the weight of structural similarity, $w_A \in [0,1]$ is the weight of action similarity, and $w_S + w_A = 1$.

B. Retrieval Command

In order to reuse already developed business components which can satisfy the functionality specified by the query, we proposed the component query command which has the similar semantic as the conventional SQL. The syntax of component query command is represented as

Select <*x*> **from** <*C*>
[**Where** <*Q*>]
[**Order by** <(*bc'*, *SDT*,*OT*)>]

where x is the name of target business component to be retrieved, and C is the reusable business component repository Q is the query condition, $Q ::= Q \wedge Q \mid Q \vee Q \mid \sim Q \mid x\ \theta\ bc$, where bc is the name of business component that can be represented as the set of business operations, and $\theta \in \{=, >, <, like, \neq\}$ is match operator. The query returns all business components in the repository that satisfy the query condition. In the following, we give the signification of some basic query conditions.

- $x=bc$ means that bc is the equivalent specification of x. If the query condition Q = "$x=bc$", then $RC=\{x \mid (x \in C) \wedge (x \rightarrow_{Equi} bc)\}$;
- $x>bc$ means that bc is the partial specification of x. If the query condition Q = "$x>bc$", then $RC=\{x \mid (x \in C) \wedge (x \rightarrow_{Part} bc)\}$;
- $x<bc$ means that bc is the extension specification of x. If the query condition Q ="$x<bc$", then $RC=\{x \mid (x \in C) \wedge (x \rightarrow_{Extend} bc)\}$;
- x *like* bc means that bc is the modification specification of x. If the query condition Q = "x *like* bc", then $RC=\{x \mid (x \in C) \wedge (x \rightarrow_{Modi} bc)\}$;
- $x \neq bc$ means that bc is the non-specification of x. If the query condition Q = "$x \neq bc$", then $RC=\{x \mid (x \in C) \wedge (x \rightarrow_{Non} bc)\}$;

According to above basic query conditions, we can construct complex query conditions. For example, if the query condition is Q = "$(x>bc_1) \wedge (x<bc_2)$ ", then $RC=\{x \mid (x \in C) \wedge (x \rightarrow_{Extend} bc_1) \wedge (x \rightarrow_{Part} bc_2\}$.

The [**Order by** <(*bc'*, *SDT*,*OT*)>] represents that the business components retrieved from component repository C according to query condition Q need to be sorted by the similarity degrees, and bc is the target business component. $SDT ::= SSD|ASD|SD$ represents the type of similarity degree, where *SSD* represents the Signature Similarity Degree, *SSD* represents the Action Similarity Degree, and *SD* represents the Similarity Degree which is the weighted sum of Signature Similarity Degree and Action Similarity Degree. $OT ::= ASCEND|DESCEND$ represent sort type.

For example, the following query command retrieves all business components that are the partial specification of *Checkorder*. The query result is sorted by *DESCEND* according to the Signature Similarity Degree, and the target business component is *Checkorder*.

Select *x* **from** *C*
Where *x*>*Checkorder*
Order by (*Checkorder*, *SSD*, *DESCEND*)

C. An Example of Application

Here, we give an example to express the business component retrieval method. Let bc be a query business component, and assuming that the repository consists of the six reusable business components: bc_1, bc_2,…,bc_6. Here we ignore the business data type, and use name to represent business components and business operations. Table I shows the seven business components.

TABLE I
BUSINESS COMPONENTS

Business component	Business operations	Concurrent regular expression
bc	a, b, c, d	$(a \prec b)$ &($c \prec d$)
bc_1	a, d	a & b
bc_2	a, c, d	b &($c \prec d$)
bc_3	a, b, c, d, e, f, i, g, k, l	$(a \prec b)$ & ($c \prec d$) & $(e \prec f)$ & i & g & $(k \prec l)$
bc_4	a, c, d, e	a&$(c \prec d)$& e
bc_5	c, d, f, g	$(c \prec d)$& $(e \prec f)$
bc_6	e, f, g	$(e \prec f)$&g

According to definition 6, we have $bc \rightarrow_{Part} bc_1$, $bc \rightarrow_{Part} bc_2$, $bc \rightarrow_{Extend} bc_3$, $bc \rightarrow_{Modi} bc_4$, $bc \rightarrow_{Modi} bc_5$, $bc \rightarrow_{Non} bc_6$. We assign weight of signature similarity degree 0.8, and weight of action similarity degree 0.2, According to the formula of similarity degree, we have

$SSD(bc,bc_1)=2/3$, $ASD(bc,bc_1)=0$, $SD(bc, bc_1)=0.5$;
$SSD(bc,bc_2)=4/7$, $ASD(bc,bc_2)=1/2$, $SD(bc, bc_2)=0.6$;
$SSD(bc,bc_3)=4/7$, $ASD(bc,bc_3)=1/2$, $SD(bc, bc_3)=0.6$;
$SSD(bc,bc_4)=3/4$, $ASD(bc,bc_4)=2/5$, $SD(bc, bc_4)=0.7$;
$SSD(bc,bc_5)=1/2$, $ASD(bc,bc_5)=1/2$, $SD(bc, bc_5)=0.2$;
$SSD(bc,bc_6)=0$, $ASD(bc,bc_6)=0$, $SD(bc, bc_6)=0.2$.

Once the retrieval process has finished, the user has to select the most closet business component that satisfy the function requirement for the query business component. From a semantic viewpoint, we select these business components that are equivalent and extension specifications from the query business component. In this example, bc_3 is an extension specification from bc. From a similar viewpoint, we select the business component that has the biggest similarity degree with query business component. In this example, bc_4 has the biggest signature similarity degree with bc, bc_2, bc_3 and bc_5 have biggest action similarity degrees with bc, and bc_4 have biggest similarity degree with bc.

VI. Conclusion

Comparing with the previous approaches, the proposed approach has the following characteristics. First, the proposed component model can describe both the static structure information about the interface and the dynamic behavior feature of a business component. The business type based on XML proposed can express the variable business data that can describe the variety of business operations. Second, we propose a multi-layer matching mode that can enrich the semantic information of business component repository. Finally, to retrieve closet business component with the query business component, we propose a method of calculating the similarity degree between business components, and give the query command to help user to retrieve approximate business components about business requirement. In order to continue this approach proposed, currently, we have developed business modeling and business component identification and retrieval prototype systems.

References

[1] William C. Chu, Chih-Wei Lu, Hongji Yang and Xudong He. A formal approach for component retrieval and integration analysis. Journal of Software Maintenance: Research And Practice. 2000; 12:325-342.
[2] William B. Frakes. A case study of a reusable component collection in the information retrieval domain. The Journal of Systems and Software 72 (2004) 265–270.
[3] W.B. Frakes, T.P. Pole, An empirical study of representation methods for reusable software components, IEEE Transactions on Software Engineering 20 (8) (1994).
[4] Hafedh Mili, Estelle Ah-Ki, Robert Godin, Hamid Mcheick, An experiment in software component retrieval. Information and Software Technology 45 (2003) 633–649.
[5] D. Merkl, A.M. Tjoa, G. Kappel. Learning the semantic similarity of reusable software components. Proceedings of 3rd International Conference on Software Reuse(ICSR'94), IEEE Computer Society Press, 1994. 33-41.
[6] S. Henninger. Supporting the process of satisfying information needs with reusable software libraries: an empirical study. Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability, ACM Press, 1995. 267-270.
[7] D. Vijay Rao, V.V.S.Sarma. A rough-fuzzy approach for retrieval of candidate components for software reuse. Pattern Recognition Letter 24(2003): 875-886.
[8] Chung-Horng Lung and Joseph E. Urban. An Approach to the Classification of Domain Models in Support of Analogical Reuse. SSR '95, Seattle, WA, USA G 1995 ACM 0-89791 -739- 1/95/0004.
[9] E. Damiani, M.G. Fugini, C. Bellettini. A hierarchy-aware approach to faceted classification of object-oriented components[J]. ACM Transactions on Software Engineering and Methodology, 1998, 8(3): 215-262.
[10] Hsien-chou liao, ming-feng chen and feng-jian wang. A Domain-Independent Software Reuse Framework Based on a Hierarchical Thesaurus. software—practice and experience, Vol. 28(8), 799–818 (10 July 1998).
[11] David Hemer. Specication-based retrieval strategies for component architectures. In： Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05).
[12] Lamia Labed Jilani, Jules Desharnais, Retrieving Software Components That Minimize Adaptation Effort.
[13] Hai-Feng Guo, Miao Liu, Jiaxiong Pi. Precise Specification Matching for Automated Component Retrieval and Adaptation.
[14] David Hemer, Peter Lindsay. Specification-based Retrieval Strategies for Module Reuse.2001.
[15] John Penix, Perry Alexander. Using Formal Specification for Component Retrieval and Reuse.
[16] Amy Moormann Zaremski, Jeannette M. Wing. Specification Matching of Software Components. ACM Transactions on Software Engineering and Methodology, 1997, 6(4): 333–369.
[17] Kakeshita.T, Murata.M. Specification-based component retrieval by means of examples. Proceedings of International Symposium on Database Applications in Non-Traditional Environments (DANTE '99), 1999:411~420.
[18] B.H.C. Cheng and J.J.Jeng. Reusing analogous components. IEEE Transaction on Knowledge and Data Engineering, 9(2), March, 1997.
[19] Redondo, R.P.D.; Arias, J.J.P.; Vilas, A.F.; Martinez, B.B. Approximate Retrieval of incomplete and formal specifications applied to vertical reuse[D]. Proceedings of International Conference on Software Maintenance (ICSM'02), 3-6 Oct. 2002:618- 627.
[20] Redondo, R.P.D.; Arias, J.J.P.; Vilas, A.F.; Martinez, B.B. Approximate Retrieval of Incomplete and Formal Specifications applied to horizontal reuse[D]. Proceedings of 28th Euromicro Conference, 4-6 Sept. 2002:90 – 97.
[21] Hai Zhuge. An inexact model matching approach and its applications[J]. The Journal of Systems and Software 67 (2003) 201–212.
[22] Praphamontripong, U.; Hu, G. XML-based software component retrieval with partial and reference matching. **P**roceedings of the 2004 IEEE International Conference on Information Reuse and Integration，8-10 Nov. 2004:127 – 132.
[23] Mili, R. Mili, and R. Mittermeir, Storing and Retrieving Software Component: A Refinement Based Approach. IEEE Transactions on software Engineeing, Vol. 23, No.7, page 139-170, 1999.
[24] Amy Moormann Zaremski, Signature and Specification Matching, Ph. Disseration, Carnegie Mellon University, 1996.
[25] Amy Moormann Zaremski Xerox Corporation, Jeannette M. Wing. Specification Matching of Software Components. ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 4, October 1997, Pages 333–369.
[26] Peter Herzum, Oliver. Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise. John Wiley& sonc, Inc, 2000.
[27] Marlon E.R. Vieira. A Compositional Approach for analyzing Dependencies in Component-Based System. Ph. Disseration, University of California, 2003.

Meng Fanchao received the B.S.and M.S. degrees from Heilongjiang University in 1997 and Harbin science and technology university in 2000, respectively. He is presently a Ph.D candidate at Center of Intelligent computing of Enterprises, School of Computer Science and Technology in Harbin Industrial of Technology of China. His current research areas include Enterprise Modeling, ERP, and software engineering..