An Efficient Method for QoS-aware Service Discovery and Composition

Ying ZHANG
Institute of Command Automation, PLA University of Science & Technology, Nanjing, China
Email: zhywl66@163.com

Xiaoming LIU, Zhixue WANG, Li CHEN
Institute of Command Automation, PLA University of Science & Technology, Nanjing, China

***Abstract*—Recently, more and more enterprises are embracing SOA paradigm to integrate and implement interoperable, robust and platform-independent distributed applications. Therefore, service discovery and composition become two main tasks which have gained great momentum. In order to improve the efficiency of service discovery and composition, a method is proposed in this paper. Firstly, some concepts and operations are defined. Then a composition algorithm is introduced in detail after definition. In addition, a QoS-aware evaluation method based on maximizing deviation calculation is proposed to resolve the problem of service selection. An instance and some simulation experiments are illustrated at last. The result shows it is an efficient and effective method for service discovery and composition.**

***Index Terms*—service discovery, service composition, Quality of Service**

I. INTRODUCTION

Nowadays, Service-Oriented Computing (SOC)[1-2], which is based on Service-Oriented Architecture (SOA)[3-4], becomes one of the hottest paradigm for integrating and developing applications both in academia and industry. There are more and more enterprises embracing SOC to integrate and implement interoperable, robust and platform-independent distributed applications. Services (or Web Services) are considered as self-contained, self-describing, and modular applications that can be published, located, and invoked across the Web[5]. Service discovery and composition are important aspects in SOC because the increasing quantity of services over the web and the complex of service-oriented application make it is unrealistic to fulfill the user needs with a single service.

Service discovery and composition are complex processes and have gained great momentum by lots of researchers, such as Reference [6-28]. Bellwood[7] discovered services based on matchmaking of key words, but the accuracy of services which are found would be low. Lee[8] implemented service composition by using data mining techniques for ubiquitous computing environments. Liang[9] proposed a design with object approach for Web services composition. Some researches completed service composition by workflow technology[10-16]. Benatallah et al.[10] proposed a framework to describe Web services composition by UML state diagram. Maamar et al.[11] presented a web services composition approach based on software agents and context. Chun et al.[12] proposed a policy-based web services composition by knowledge etc. Shi et al.[17] and Ma et al.[18] proposed matchmaking and discovery services based on description logic. AI techniques such as HTN[19-20], Petri Net[21-22], Genetic algorithm[23] are also widely used for service matchmaking, discovery and composition. Still some researches implemented service discovery and composition by graph-based or tree-based methods, such as Reference [24-28].

Based on their researches, we propose an efficient and effective service composition strategies based on *FAS* (Feature Association Set) in this paper. First of all, we define the *FAS* and some operations on it. After definition, we introduce our service composition algorithm based on *FAS* in detail, which has three main differences with other methods, as follows:

(1) When *FAS* is defined, the semantic similarity is considered in order to increase the recall rate and accuracy.

(2) An index table, which is building based on *FAS*, is created to reduce the searching spaces of candidate services. Therefore the searching efficiency will be increased obviously.

(3) Because services may provide similar function with different quality, the optimal service composition is chosen by Quality of Service (QoS), which is usually ignored by some service composition methods.

The remainder content is organized as follows: the *FAS* and its operations are defined in section II. The service composition algorithm is presented in detail in section III, while an instance for illustrating the algorithm is proposed in section IV. Some experiments and result analysis is given in section V. Section VI draw some conclusions and future works at last.

II. PRELIMINARIES

A. Service definition

When we mentioned service in repository, we refer to the atomic service rather than composite service[29]. This means we focus on the composition of atomic services, and the composite service is the result of service

doi:10.4304/jsw.7.1.102-109

composition which could be integrated to more complex service.

A service can be formally defined as follows:

Definition 1: service definition.

$$Service = \langle SN, SD, SF, SA \rangle$$

where:

SN is the name of a service.

SD is the descriptions of a service that are usually using natural languages.

SF is service features including inputs, outputs, preconditions and results of a service.

SA is the attributes of a service, especially the QoS attributes.

In theory, all these four aspects should be considered during service composition. However, with the limited length of one paper, we predigest the service definition to a 4-tuple:

$$Service = \langle SN, I, O, A \rangle$$

where:

SN is the name of a service.

I represents the input-set of a service.

O represents the output-set of the service.

A is the QoS attributes of a service, we consider 4 QoS attributes[30-31] in this paper, as follows:

• service cost $Q_c(S)$: the service cost is used to describe the amount of money that the service consumer must pay for using the service.

• response time $Q_t(S)$: the response time refers to the time duration from a request being sent to the results are received. It includes the total time for service performance and round-trip communication delay.

• network delay $Q_d(S)$: the network delay is the network transmission time required to receive the service. It is especially important for multimedia services.

• service availability $Q_a(S)$: the probability of the service is available.

Definition 2: service request definition.

$$SR = \langle I, O, T \rangle$$

where:

I represents the input-set of a service request.

O represents the output-set of the service request.

T is the time limit of request time.

B. FAS definition

Definition 3: the feature association set.

$$FAS(C) = \{SN_1, SN_2, ..., SN_n\}$$

where $FAS(C)$ is a set represented all registered services which can provide the output concept C in service repository. SN_i, $i = 1,...,n$ represents the service name.

It is worth to notice that both input concepts I and output concepts O have semantic support by a domain ontology in this paper. Semantic similarity computing is a hot topic in service matchmaking. However, we do not discuss the elaborative method here. We use the method mentioned in [32], but it is not the only resolution. The reason we introduce the semantic similarity is consideration of the fact that if one concept could output by a service, its semantic similarity concepts maybe also output by the same service. For example, if SN_1 could output the concept C_1, SN_2 could output the concept C_2, SN_3 could output the concept C_3. The semantic similarity between C_1 and C_2 is $Sim(C_1, C_2) = 0.75$, and $Sim(C_1, C_3) = 0.9$, $Sim(C_2, C_3) = 0.85$. If the threshold of semantic similarity is $\theta = 0.8$. Then we can obtain $FAS(C_1) = \{SN_1, SN_3\}$, $FAS(C_2) = \{SN_2, SN_3\}$ and $FAS(C_3) = \{SN_3, SN_1, SN_2\}$.

We build a *FAS* index table for service repository and update the list periodically. If there is a new service registered successfully, its related information will be added to the *FAS* index table accordingly. Therefore, once we get the *FAS* index table of all the registered services, we could improve the efficiency by searching the table instead of the whole service repository.

Definition 4: the overlap of *FAS*.

$$\begin{aligned} &FAS(C_1) * FAS(C_2) \\ &= \{SN \mid SN \in FAS(C_1) \wedge SN \in FAS(C_2)\} \end{aligned} \quad (1)$$

where the symbol $*$ represent the overlap operation of *FAS*, aiming at finding the services which could provide concept C_1 as well as C_2. For instance, if $FAS(C_1) = \{S1, S2\}$, $FAS(C_2) = \{S1, S3, S4\}$, then $FAS(C_1) * FAS(C_2) = \{S1\}$, that means $S1$ could provide concepts both C_1 and C_2.

Definition 5: The conjunction of *FAS*.

$$\begin{aligned} &FAS(C_1) \circ ... \circ FAS(C_n) \\ &= \left\{ \begin{array}{l} SN \mid SN : SN_1 \circ ... \circ SN_n \wedge SN_i \in FAS(C_i), \\ i = 1,...,n \end{array} \right\} \end{aligned} \quad (2)$$

where the symbol $\circ$ represent the conjunction operation of *FAS*, which means executing services from SN_1 to SN_n in sequence one by one to provide concepts $C_1...C_n$. SN_{n-1} is the predecessor service of SN_n, while SN_n is the subsequence service of SN_{n-1}. That is to say, the order is important in conjunction operation. For instance, if there is $FAS(C1) = \{S1\}$ and $FAS(C_2) = \{S2\}$, then $FAS(C_1) \circ FAS(C_2) = \{S1 \circ S2\}$. Correspondently, we also could obtain $FAS(C_2) \circ FAS(C_1) = \{S2 \circ S1\}$. In this paper, when we mention conjunction operation of two services, such as $S1 \circ S2$, we mean $S2$ takes all (or part of) outputs of $S2$ as inputs, i.e. the all (or part of) outputs of $S1$ are all (or part of) inputs of $S2$, thus the predecessor service $S1$ must be performed before the subsequence service $S2$ will be invoked. Therefore, in general, $FAS(C_1) \circ FAS(C_2) \neq FAS(C_2) \circ FAS(C_1)$.

Definition 6: The union of *FAS*.

$$FAS(C_1)+...+FAS(C_n) = \left\{ \begin{array}{l} SN \mid SN: SN_1+...+SN_n \wedge \\ SN_i \in FAS(C_i), i=1,...,n \end{array} \right\} \quad (3)$$

where the symbol + represent the union operation of *FAS*, which means service $SN_1...SN_n$ are indispensable to provide concepts $C_1...C_n$. However, unlike the conjunction operation, the union operation of *FAS* has no order limits. For instance, if $FAS(C_1)=\{S1\}$, $FAS(C_2)=\{S2,S3\}$, then

$$\begin{aligned} FAS(C_1)+FAS(C_2) &= \{S1+S2, S1+S3\} \\ &= \{S2+S1, S3+S1\} \\ &= FAS(C_2)+FAS(C_1) \end{aligned}$$

Definition 7: Some axioms.

$$\prod_{j=1}^{n} FAS(C_j) = FAS(C_1)*FAS(C_2)*...*FAS(C_n) \quad (4)$$

$$\sum_{j=1}^{n} FAS(C_j) = FAS(C_1)+FAS(C_2)+...+FAS(C_n) \quad (5)$$

$$FAS(C)*FAS(C) = FAS(C) \quad (6)$$

$$\varnothing \circ FAS(C) = \varnothing \quad (7)$$

$$FAS(C)\circ FAS(C) = FAS(C) \quad (8)$$

$$\varnothing + FAS(C) = \varnothing \quad (9)$$

$$FAS(C)+FAS(C) = FAS(C) \quad (10)$$

III. SERVICE COMPOSITION AND SELECTION

A. Service Composition Algorithm

The SCFAS (Service Composition based on *FAS*) algorithm is a recursive process with two parts: service discovery and service composition. Service matchmaking is the main task in service discovery, including inputs matchmaking and outputs matchmaking. And service composition mainly in two ways: conjunction and union, as defined in section II. We define two functions **InputsProcess** and **OutputsProces**s for SCFAS algorithm.

The main principle of SCFAS is: in according with the inputs and outputs provided by service request, searching the *FAS* index table that build in advance to obtain the candidate services set. If there is no single service in service repository could meet the service request, service composition will go to work. The algorithm is shown as follows.

SCFAS ALGORITHM
INPUTS: service request $SRq = \langle I_q, O_q, T_{out} \rangle$
OUTPUTS: the matched service set *MatchedServiceSet*
STEPS:
obtain the output-set $O = O_q = \{O_1, O_2, \cdots, O_n\}$ and input-set $I = I_q = \{I_1, I_2, \cdots, I_m\}$ from service request;
while time < T_{out}
{
MatchedServiceSet = *NULL* ;
ServiceSet = $FAS(O_1)$;
OutputsProcess (*ServiceSet* , *O*);
return *MatchedServiceSet* ;
}

Figure 1. The SCFAS algorithm

OutputsProcess (*ServiceSet* **,** *O* **)**
{
i=1;
$n = |O|$;
if n is an odd number
then $m = (n+1)/2$;
else $m = n/2$;
while $i <= n$ and *ServiceSet* $\neq \varnothing$
{
get $FAS(O_i)$ from *FAS* index table;
ServiceSet = *ServiceSet* $* FAS(O_i)$;
i= i+1;
};
if *ServiceSet* $\neq \varnothing$
then
{
for each element in *ServiceSet* do
{
tempSet = *ServiceSet.element* ;
tempSet =**InputsProcess** (*tempSet*);
if *tempSet* $\neq \varnothing$
then add *tempSet* to *MatchedServiceSet* ;
}
}
else
{
k=1;
while $k <= m$ and *MatchedServiceSet* = *NULL*
{
for i =1 to n
{
if $((\sum_{i=1}^{k} FAS(O_i) \neq \varnothing)$
&& $(\prod_{j=1}^{n-k, j\neq i} FAS(O_j) \neq \varnothing))$
then
{
$ServiceSet = \sum_{i=1}^{k} FAS(O_i) + \prod_{j=1}^{n-k, j\neq i} FAS(O_j)$;
for each element in *ServiceSet* do
{
tempSet = *ServiceSet.element* ;
tempSet =**InputsProcess** (*tempSet*);

```
            if tempSet ≠ ∅
            then add tempSet to MatchedServiceSet ;
          }
        }
      } // end for i
      k=k+1;
    } // end while
  } // end else
  return MatchedServiceSet
}
```

Figure 2. The OutputsProcess function for SCFAS

```
InputsProcess ( tempSet )
{
   obtain input-set of tempSet by searching the
  service repository as Iser
   Inew = Iser − I ;
   if Inew = ∅
   then return tempSet ;
   else
   {
    Onew = Inew ;
   tempSet =OutputsProcess (tempSet, Onew) ∘ tempSet ;
    return tempSet
   }
}
```

Figure 3. The InputsProcess function for SCFAS

Let us understand the algorithm in detail. In the initialization, we get the output set O and input set I from service request. And then invoke **OutputsProcess**. In **OutputsProcess**, searching $FAS(O_i)$ in the *FAS* index paper to find all services which can output concept O_i firstly. If there is any $FAS(O_i)=\varnothing$, it means no service in service repository can produce the concept O_i currently, the match failed. Otherwise, the next step is computing the overlap set S of all O_i, i.e $S=\prod_{j=1}^{n} FAS(O_j)$ as the candidate services set, the object of this step is searching for the single services which matched all outputs with service request. Thus there are mutually exclusive two results:

Case Opt.1: $S\neq\varnothing$: this means at least one of the services in service repository can produce all outputs which described by O, then turn to inputs matchmaking, i.e. invoking the **InputsProcess**. For each service in S, searching its input-set I_{ser} in service repository, then compute the difference set I_{new} of I_{ser} and I, i.e. $I_{new}=I_{ser}-I$, now there will be three cases of I_{new}:

Case Ipt.1: $I_{new}=\varnothing$, it means current inputs provided by I are sufficient to produce all needed inputs, matchmaking successfully.

Case Ipt.2: $I_{new}\neq\varnothing$, this means there is some inputs needed by S have not been provide by I, i.e. it is needed to find the predecessor services of current service. Let $O_{new}=I_{new}$, invoking **OutputsProcess** with the new output-set compute O_{new}. Now it is beginning the recursive matchmaking process until the matchmaking process is finished successfully or time is out.

Case Opt.2: $S=\varnothing$: this means there is no single service can be matched out-put set O directly, thus the next stage is performing service composition. From O_1 to O_n, selecting each element in O, computing $S=FAS(O_i)+\prod_{j=1}^{n-1,j\neq i} FAS(O_j)$, which is case of $ServiceSet=\sum_{i=1}^{k} FAS(O_i)+\prod_{j=1}^{n-k,j\neq i} FAS(O_j)$ with $k=1$. If $S\neq\varnothing$ and $S\neq FAS(O_i)$, it means the candidate service set is formed by composition of two groups of service, the one produce concept O_i, the other provide the remain concepts. Then invoking **InputsProcess** function and beginning recursive matchmaking and composition process until the composition is finished successfully or time is out. However, if composition of two groups of services still cannot satisfy all the outputs of O, as $ServiceSet=\sum_{i=1}^{k} FAS(O_i)+\prod_{j=1}^{n-k,j\neq i} FAS(O_j)$ with $k=k+1$, composition of three groups of services will go to work in the same way, then four groups, five groups, and so on.

B. Service Selection

If the SCFAS algorithm returns a null set, it means that the matching is fail. Otherwise, if the algorithm executes successfully with only one service set, it is just the matched service(s) we need. Or, if there are more than one service sets, it means there are several optional solutions for performing the service request. Although these services may provide same or similar function, the QoS attributes of them are different generally. How to choose the optimal composite service is important but usually ignored by many service composition methods.

Different QoS attributes have different measurements. For example, the the unit of service cost $Q_c(S)$ uses the unit of money while unit of response time $Q_t(S)$ uses the unit of time. Therefore, we classified service selection as Multiple Attributes Decision Making (MADM) problem, and resolved the problem by maximizing deviation method[33]. The steps are as follows.

Step 1. Compute the QoS integrated values for each composite service according to Table 1.

TABLE 1 QOS INTEGRATED VALUES FOR COMPOSITE SERVICE

QoS attribute	∘ **(conjunction)**	+ **(union)**
service cost $Q_c(S)$	$\sum_{i=1}^{n} Q_c(S_i)$	$\sum_{i=1}^{n} Q_c(S_i)$
response time $Q_t(S)$	$\sum_{i=1}^{n} Q_t(S_i)$	$\max\{Q_t(S_1),...,Q_t(S_n)\}$

QoS attribute	○ (conjunction)	+ (union)
network delay $Q_d(S)$	$\sum_{i=1}^{n} Q_d(S_i)$	$\max\{Q_d(S_1),...,Q_d(S_n)\}$
availability $Q_a(S)$	$\prod_{i=1}^{n} Q_a(S_i)$	$\prod_{i=1}^{n} Q_a(S_i)$

Step 2. Normalize the QoS attributes. There are two types of attributes in this paper: the efficiency attribute and the cost-based attribute. The former features its value is the bigger the better, such as the the service availability $Q_a(S)$. The latter features its value is the smaller the better, such as the $Q_c(S)$, $Q_t(S)$ and $Q_d(S)$.

The efficiency attribute is normalized as:

$$r_{ij} = \frac{x_{ij}}{x_j^+} \tag{11}$$

where $x_j^+ = \max_{1\le i\le n}(x_{ij}), i \in N$.

The cost-base attribute is normalized as:

$$r_{ij} = \frac{x_j^-}{x_{ij}} \tag{12}$$

where $x_j^- = \min_{1\le i\le n}(x_{ij}), i \in N$.

Step 3. Compute the optimal weight w as follows.

$$w_j = \frac{\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{ij} - r_{kj}\right|}{\sum_{j=}^{m}\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{ij} - r_{kj}\right|}, j \in M \tag{13}$$

Step 4. Calculate the comprehensive value for each composite service:

$$z_i = \sum_{j=1}^{m} r_{ij} w_j \tag{14}$$

Step 5. Sort the composite services based on the comprehensive value z_i. The one has the biggest value is the optimal one.

IV. CASE STUDY

In this section, we give an example for comprehending the SCFAS algorithm better. There are six registered services in the service repository. Concept G is similar to concept H and $sim(G,H) = 0.95$, $\theta = 0.8$. The service repository and its *FAS* index table as shown in table 1. We suppose that the service request provides input concepts $I = \{A, B, C\}$ and wants to get output concepts $O = \{D, E, F\}$.

TABLE 2 THE SERVICE REPOSITORY AND ITS FAS INDEX TABLE

Service Repository							*FAS* index table
SN	*I*	*O*	*A*				
			$Q_c(S)$ ($)	$Q_t(S)$ (sec)	$Q_d(S)$ (sec)	$Q_a(S)$ [0,1]	
S1	{B}	{E,F}	1.2	2.8	0.5	0.88	$FAS(B) = \{S6\}$
S2	{C,G}	{D}	2.0	1.0	0.6	0.90	$FAS(D) = \{S2\}$
S3	{K}	{E}	0.8	3.5	0.8	0.85	$FAS(E) = \{S1, S3\}$
S4	{A,B}	{G}	1.5	2.6	0.5	0.75	$FAS(F) = \{S1\}$
S5	{C}	{H}	1.8	2.2	0.3	0.72	$FAS(G) = \{S4, S5\}$
S6	{A,K}	{B}	2.5	3.0	0.7	0.76	$FAS(H) = \{S5, S4\}$

The SCFAS algorithm executes as follows:

(1) According to the out-put set of service request O , invoke **OutputsProcees**.

(2) For the sake of the initial candidate service set is $S = FAS(D) * FAS(E) * FAS(F) = \varnothing$, which belonged to case **Opt.2** in section 3. The next step is computing $S = FAS(O_i) + \prod_{j=1}^{n-1, j\neq i} FAS(O_j)$.

Because $FAS(F) + (FAS(D) * FAS(E)) = \varnothing$, and $FAS(E) + (FAS(D) * FAS(F)) = \varnothing$, thus the current candidate services set is

$$S = FAS(D) + (FAS(E) * FAS(F)) = \{S2\} + \{S1\} = \{S2 + S1\} \neq \varnothing$$,

thus turn to matchmaking inputs, i.e. invoke **InputsProcess**.

(3) Obtain the input-set of $S = \{S2 + S1\}$ by searching service repository, that is $I_{ser} = \{C, G, B\}$. Thus we get the different set $I_{new} = I_{ser} - I = \{G\}$. Because G is not included by I , this is belonged to **case Ipt.2**, so invoke **OutputsProcess** again with the new output-set G .

(4) Now, the candidate service set $S = \prod FAS(G) = \{S4, S5\}$ is belonged to case **Opt. 1**, invoke **InputsProcess** again.

(5) For each element in S, searching the service repository and getting the inputs of $S4$ are $\{A, B\}$ which has been provided in I, this is the **case Ipt.1**, thus service composition finished successfully. Moreover, the input of $S5$ is $\{C\}$ which also has been included in I. Therefore, there are two matched service composition: the one is $S5 \circ (S2 + S1)$, and the other is $S4 \circ (S2 + S1)$.

Now, it is need to make a decision between composite services CS_1=$S5 \circ (S2 + S1)$ and $CS_2 = S4 \circ (S2 + S1)$. According to the method proposed in Section III, the steps are as follows.

(1) According to Table 1, the QoS integrated values for CS_1 and CS_2, and the results are shown in Table 3.

TABLE 3 QOS INTEGRATED VALUES FOR CS_1 AND CS_2

Composite service	$Q_c(S)$	$Q_t(S)$	$Q_d(S)$	$Q_a(S)$
CS_1	4.7	5.4	1.1	0.5940
CS_2	5.0	5.0	0.9	0.5702

(2) Based on formula (11) and (12), the normalized QoS attributes of CS_1 and CS_2 are shown as Table 4.

TABLE 4 THE NORMALIZED QOS VALUES OF CS_1 AND CS_2

Composite service	$Q_c(S)$	$Q_t(S)$	$Q_d(S)$	$Q_a(S)$
CS_1	1.0000	0.9259	0.8182	1.0000
CS_2	0.9400	1.0000	1.0000	0.9600

(3) Compute the optimal weight w according to formula (13).

$$w_1 = \frac{\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{i1} - r_{k1}\right|}{\sum_{j=}^{m}\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{ij} - r_{kj}\right|} = 0.2538,$$

$$w_2 = \frac{\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{i2} - r_{k2}\right|}{\sum_{j=}^{m}\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{ij} - r_{kj}\right|} = 0.2519,$$

$$w_3 = \frac{\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{i3} - r_{k3}\right|}{\sum_{j=}^{m}\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{ij} - r_{kj}\right|} = 0.2379,$$

$$w_4 = \frac{\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{i4} - r_{k4}\right|}{\sum_{j=}^{m}\sum_{i=1}^{n}\sum_{k=1}^{n}\left|r_{ij} - r_{kj}\right|} = 0.2564.$$

(4) Calculate the comprehensive value for each composite service:

$$z(CS_1) = \sum_{j=1}^{4} r_{1j} w_j = 0.9381$$

$$z(CS_2) = \sum_{j=1}^{4} r_{2j} w_j = 0.9745$$

(5) Because $z(CS_2) \succ z(CS_1)$, the optimal composite service is $CS_2 = S4 \circ (S2 + S1)$.

V EXPERIMENT RESULTS AND ANALYSIS

In order to evaluate the efficiency and effectiveness of SCFAS, we take emulation experiments on a Intel Core2 Duo 1.99GHz with 1GB RAM.

We compare two algorithms with SCFAS. The one is Front-to-Back algorithm, which is matchmaking inputs of service request and service at first. If one service needs inputs less than service request provided, then checking its outputs could provide all outputs needed by service request or not. If it could, it will be the matched service. The other is Back-to-Front algorithm, which matchmaking outputs firstly. If a service in the service repository provide all outputs described by service request, then matchmaking its inputs and service request. If the inputs are included by inputs of service request, the service is the matched service. Fig. 4 and Fig. 5 show some of the results.

In Fig. 4, we simulate 50 groups of service requests in different service repository, each group has 10 service requests, and the scale of service repository are 100, 200, 500, 1000, 2000 and 5000. We observe the average processing time of three algorithms. From this picture, we can see that with the increase of services, the average processing time of Front-to-Back algorithm and the Back-to-Front algorithm are increasing obviously while the SCFAS algorithm is increasing smoothly.

In Fig. 5, we try to observe the average processing time of different service requests in same quantity of services. The service repository has 500 services, and the numbers of service requests are 5, 10, 15, 20 and 50. From this picture, we can see that with the increase of service requests, the three algorithms are all increased, and the Front-to-Back method has the longest time cost.

From the experiments, we could conclude that both the scale of service repository and numbers of service request are the factors influencing the processing time of service composition. When the differences of these two factors are not huge, the differences of responding time among the three algorithms are inconspicuous. However, with the increasing of service requests or the extending of the service repository, the *SCFAS* tend to have obvious advantage in time consuming because it is using the *FAS* index table. The front-to-back algorithm for service discovery and composition is easy to understand but has

less efficiency. The back-to-front algorithm is differing from the front-to-back algorithm, it is goal-driven. This means the back-to-front algorithm can avoid some meaningless searching of finding matched service inputs. However, these two ways both need to match the service request and all registered service in service repository one by one. They are time costly and not suitable especially the size of service repository is going larger and larger.

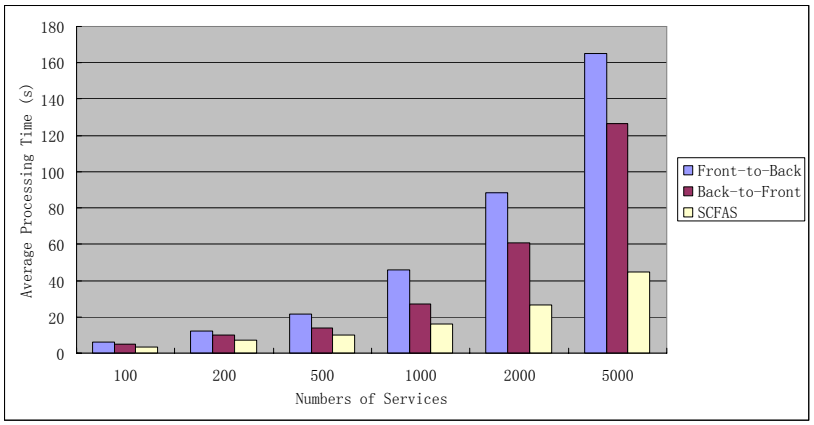

Figure 4. The average processing time in different service repository

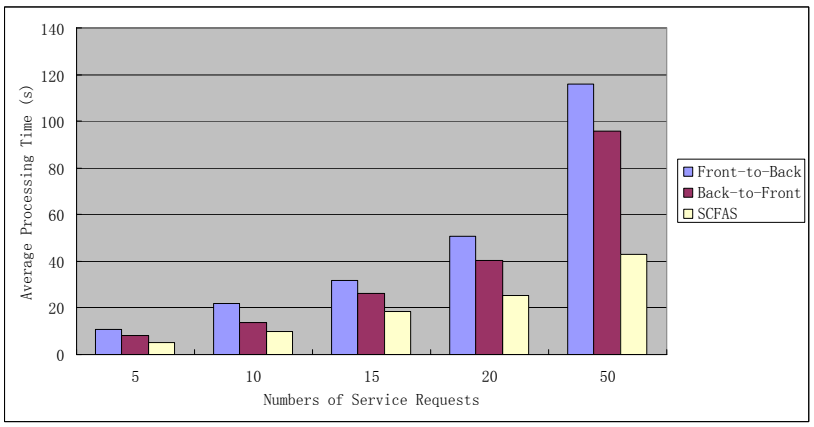

Figure 5. The average processing time with different service requests

VI Conclusions

In this paper, we propose an efficient and effective service composition method based on *FAS*. Firstly, we defined the *FAS* and some operations on it, such as the overlap, conjunction and union operations. Based on *FAS*, we introduce the SCFAS algorithm in detail. Considering the ignorance of service selection in some service composition algorithm, we used a QoS-aware method based on maximizing deviation calculation to resolve the problem. Additionally, we gave an instance and some simulation experiment to illustrate our method. The SCFAS algorithm reduces the searching spaces by retrieving in *FAS* index table instead of the whole service repository. Therefore, although building and maintenance *FAS* index table will cost some time, the SCFAS algorithm still is an efficient an effective method for automatic service composition.

The future work mainly includes two aspects. On the one hand, we will research how to implement the composition of services based on fuzzy information. On the other hand, we only discuss the QoS attributes with the data type of real number in this paper, however, the type of QoS attributes are various in reality. Therefore, we will also study the service selection method with fuzzy and uncertain QoS information, and so on.

References

[1] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, et al. “Service-Oriented Computing: State of the Art and Research Challenges”, *Computer*, 2007.

[2] Michael P. Papazoglou, Joachim W. Schmidt, John Mylopoulos, *Service-Oriented Computing*, The MIT Press Cambridge, Massachusetts London, England, 2009.

[3] Newcomer E, Lomow G, *Understanding SOA with Web Services*, Pearson Education, Inc, 2005.

[4] Erl T, *SOA design Patterns*, Prentice Hall, 2008.

[5] J Rao, X Su, “A Survey of Automated Web Service Composition Methods”, *Proc. of First International Workshop on Semantic Web Services and Web Process Composition*, 2004.

[6] Adam Barker, Christopher D. Walton, David Robertson, “Choreographing Web Services”, *IEEE Transactions on services computing*, 2(2), pp. 152-166, 2009.

[7] Bellwood T, Capell S, Clement L, Colgrave J, Dovey MJ, Feygin D, et al. *UDDI version 3.0*, 2002.

[8] S. Y. Lee, J. Y. Lee, B. I. Lee, "Service composition techniques using data mining for ubiquitous computing environments", *International Journal of Computer Science and Network Security*, 6(9), pp. 110-117, 2006.
[9] Wen-Yau Liang, Chun-Che Huang, Horng-Fu Chuang, "The design with object (DwO) approach to Web services composition", *Computer Standards & Interfaces*, 29, pp. 54 – 68, 2007.
[10] B. Benatallah, M. Dumas, Q. Sheng, A. Ngu, "Declarative composition and peer-to-peer provisioning of dynamic web services", *Proc. of the 18th International Conference on Data Engineering (ICDE'02)'*, California, USA, 2002, pp. 297– 308.
[11] Z. Maamar, K.M. Soraya, Y. Hamdi, "A web services composition approach based on software agents and context", *Proc. of SAC'04*, Nicosia, Cyprus, 2002, pp. 1619–1623.
[12] S.A. Chun, V. Atluri, N.R. Adam, "Policy-based web services composition", *Proc. of the 14th International Workshop on Research issues on Data Engineering: Web Services for E-Commerce and E-Government Applications*, 2004, pp. 85–92.
[13] A. Mingkhwan, P. Fergus, O. Abuelma'Atti, et al. "Dynamic service composition in home appliance networks", *Multimedia Tools and Applications*, 29(3):257-284, 2006.
[14] H. Pourreza, P. Graham, "On the fly service composition for local interaction environments", *Proc. of IEEE International Conference on Pervasive Computing and Communications Workshops*, 2006.
[15] Mirko Viroli, Enrico Denti, Alessandro Ricci, "Engineering a BPEL orchestration engine as a multi-agent system", *Science of computer programming*, vol. 66, pp. 226–245, 2007.
[16] A. Bottaro, J. Bourcier, C. Escoer, et al. "Autonomic context-aware service composition", *Proc. of 2nd IEEE International Conference on Pervasive Services*, 2007.
[17] Shi Zhongzhi, Jiang Yunchen, et al. "Agent service matchmaking based on description logic", Chinese Journal of Computers, 27(5), pp. 626−635, 2004 (in Chinese with English abstract).
[18] Ma Yinglong, Jin Beihong, Feng Yulin, "Dynamic discovery for semantic Web services based on evolving distributed ontologies", Chinese Journal of Computers, 28(4), pp. 603−615, 2005 (in Chinese with English abstract).
[19] L. Qiu, Z. Shi, F. Lin. "Context optimization of AI planning for services composition", *Proc. of the IEEE International Conference on e-Business Engineering*, pp. 610-617, 2006.
[20] Therani Madhusudan, N. Uttamsingh, "A declarative approach to composing web services in dynamic environments", *Decision Support Systems*, 41(2), pp. 325-357, 2006.
[21] Yu-Liang Chi, Hsun-Ming Lee, "A formal modeling platform for composing web services", *Expert Systems with Applications*, vol. 34, pp. 1500–1507, 2008.
[22] Valentín Valero, M. Emilia Cambronero, Gregorio Díaz, et al. "A Petri net approach for the design and analysis of Web Services Choreographies", *The Journal of Logic and Algebraic Programming*, vol. 78, pp. 359–380, 2009.
[23] LIU Xiangwei, XU Zhicai, YANG Li, "Independent Global Constraints-aware Web Service Composition Optimization Based on Genetic algorithm", *Proc. of International conference on industrial and information systems*, 2009.
[24] Bin Xu, Tao Li, Zhifeng Gu, et al. "SWSDS: Quick Web Service Discovery and Composition in SEWSIP", *Proc. of the 8th IEEE International Conference on Ecommerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, 2006.
[25] Li Kuang, Ying Li, Jian Wu, et al. "Inverted Indexing for Composition-Oriented Service Discovery", *Proc. of IEEE International Conference on Web Services*, 2007.
[26] A. Zhou, S. Huang, X. Wang, "BITS: A Binary Tree Based Web Service Composition System," *Journal of Web Services Research*, vol. 4, 2007.
[27] H. Tang, F. Zhong, C. Yang, "A Tree-Based Method of Web Service Composition", *Proc. of the Int. Conf. on Web Services*, Beijing, China, 2008.
[28] M.M. Shiaa, J.O. Fladmark, B. Thiell, "An Incremental Graphbased Approach to Automatic Service Composition", *Proc. of the Int. Conf. on Services Computing*, Honolulu, HI, USA, 2008.
[29] The OWL Services Coalition. *OWL-S: Semantic markup for Web Services*, 2003.
[30] Tao Yu, Kwei J Lin, "Service Selection Algorithms for Web Services with End-to-End QoS constraints", *Proc. of the IEEE International Conference on E-commerce Technology*, 2004.
[31] Minhyuk Oh, Jongmoon Baik, Sungwon Kang, et al. "An Efficient Approach for QoS-Aware Service Selection Based on A Tree-Based Algorithm", *Proc. of Seventh IEEE/ACIS International Conference on Computer and Information Science*, 2008.
[32] P.Resnik, "Semantic similarity in a taxonomy: An Information-Based Measure and its Application to problems of Ambiguity in Natural Language", *Journal of Artificial Intelligence Research*, vol.11, 1999.
[33] Xu Zeshui. *Uncertain Multiple Attribute Decision Making: Methods and Applications*, Tsinghua Univercity Press, 2004 (in Chinese).

Ying ZHANG was born in 1982. She is a candidate for doctor in Institute of Command Automation, PLA University of Science & Technology. Her main research is software engineering, requirement engineering, SOA; Email: zhywl66@163.com

Xiaoming LIU was born in in 1956. He is a professor for doctor in Institute of Command Automation, PLA University of Science & Technology. His main research is system engineering, computer simulation.

Zhixue WANG was born in 1961. He is a professor for doctor in Institute of Command Automation, PLA University of Science & Technology. His main research is requirement engineering, system engineering, SOA.

Li CHEN was born in 1982. He received the Bachelor of Engineering and Master of Engineering degrees from Institute of Meteorology, PLA University of Science and Technology in 2005 and 2008, respectively. He began his PhD study in Institute of Command Automation, PLA University of Science and Technology since 2008. His research interests include semantic Web, data mining, and Web services.