Web Service Aggregation Using Semantic Interoperability Oriented Method

JIANXIAO LIU, KEQING HE AND DA NING
State Key Lab of Software Engineering
Wuhan University
Wuhan, 430072 P.R. China

In the modern world of service-oriented software engineering (SOSE), the web services can be aggregated from the semantic interoperability level to meet the user’s personal and diversified needs. Firstly, the paper proposes a service clustering method based on service ontology and it clusters services from the aspect service function to form the service clusters. This can significantly reduce the overhead and enhance the service discovery efficiency. In addition, this paper makes use of the service capability and the interaction information to organize the service clusters form the semantic interoperability level. And it discusses the sufficient and necessary capability, and the type of interoperability. The users can discover the related services that can meet their needs efficiently. The corresponding algorithms are also designed. Finally, the effectiveness and feasibility of the proposed method are validated through experiments and a practical case study.

1. INTRODUCTION

Service-oriented computing is emerging as a new promising computing paradigm for developing software applications. As one of the key technologies of services computing, web service is designed to support interoperable machine-to-machine interaction over the network, it can provide a set of relatively independent functions. All kinds of resources (including data, component, etc.) can be packaged to form a unified service interface for users. In order to achieve the user-centered service provision and meet the public service requests in high quality of experience (QoE), not only one or more discrete services need to be provided, but also a set of services that can realize semantic interoperability are needed. That is to say, we need to combine the atomic service to form composite services [1] to meet user’s requirements better.

However, with the increasing number of web services, the user’s requirements have the personal and diversified characteristics. And these personal and diversified requirements can be understood in the following two aspects. On one hand, different users will choose different service combination processes for the same goal. So the corresponding discovered services that can realize the processes will be different. On the other hand, different services have different Qos (such as execution time, cost, reliability) values and sometimes these Qos values are largely different. Then users who have different preferences will select different services in face of a large number of services which realize...
same function. In order to solve the above problems, some service clustering methods have been proposed and they cluster the services which realize same function but have different Qos values to form service clusters. Then these service clusters will be organized using some organization methods [2-4]. This can form the service assets [5] and therefore the proper services can be selected directly according to user’s personal requests. Due to the heterogeneity and heterogeneous features of the abundant web service resources, we need to realize service aggregation and organization from the semantic interoperability level. Then there will be certain semantic interoperability capability between the organized services, and it helps to enhance the service discovery efficiency and accuracy. At present, there exists some research work about this aspect, such as GODSS [2], VINCA [3], workflow [4] and so on. These methods only organize the services from the aspect of service execution process. They didn’t aggregate services from the aspect of semantic interoperability, and the service’s heterogeneity characteristic is also not considered. In addition, the time of some existing service clustering approaches are too long and it leads to low efficiency. The accuracy of some other clustering methods is not high.

The main contributions of our work are summarized as follows,

(1) We propose a service clustering method through service ontology. The services which realize same function but have different Qos values are clustered to form service clusters.

(2) We introduce a service ontology semantic interoperability clustering method and the different service clusters are organized from aspects of service capability and semantic interaction information.

(3) We use experiments to evaluate the service clustering efficiency, accuracy and service discovery efficiency of the proposed approach. A case study is also given to illustrate the process of semantic interoperability clustering algorithms.

The rest of the paper is organized as follows. In section 2, we introduce the related concepts and service aggregation framework. The service clustering calculation process is described in section 3. The organization of service clusters is elaborated in section 4. The related work will be described in section 5. The experiments and a case study are introduced in sections 6 and 7. Finally, the conclusion and the next step work are discussed.

2. BASIC CONCEPT AND SERVICE AGGREGATION FRAMEWORK

2.1 Service Aggregation

Service clustering: it refers to group the web services which realize same function and have same interface into service clusters. The services in the same service cluster have different Qos values.

Service composition: it refers to orchestrate one or more services to form the composite service that can offer a value-added service [6].
Service aggregation: Based on the formed service clusters, it refers to integrate these service clusters according to business logic relationship to form services execution process. The service aggregation emphasis the integration of the service clusters and service composition concentrates on the integration of services.

2.2 Semantic Interoperability

The semantic interoperability [7] refers to the capability that two software modules or systems exchange the data with precise meaning, and the receiving party can accurately translate or convert the information and ultimately produce effective collaborative results. The levels of semantic interoperability are generally classified as: Meaning Interoperability, Partial Semantic Interoperability and No Semantic Interoperability.

2.3 Service Aggregation Framework

This paper proposes a kind of framework that realizes service aggregation from two clustering levels, as shown in Fig. 1.

(1) Service clustering

According to the service ontology (see section 3.2) that is generated by the modeling of specific type of available services, the services are clustered to form service clusters. This can significantly reduce the overhead and enhance the service discovery efficiency.

(2) Service ontology semantic interoperability clustering

Based on service ontology, the interoperable collaboration ontology (see in section 4.1) will be extracted. This ontology is the semantic standard of service ontology to realize interoperability capability. As shown in Fig. 1, service ontology (1-6) realizes semantic interoperability clustering towards collaboration ontology.
3. SERVICE CLUSTERING

3.1 Definitions

In [8, 9], environment ontology is proposed and it can be formed based on the general definition \( O = \{C, R, \text{rel}\} \). It is defined as \( EO = \{Rsc, \mathfrak{R}, \text{Rel}, \text{HSM}, \text{hsm}\} \). \( Rsc \) is a set of environment entities and \( \mathfrak{R} \) is a set of relations between environment entities. \( \text{Rel}: \mathfrak{R} \to Rsc \times Rsc \) means a function that relates to environment entities.

\( EO \) can express the tree-like hierarchical state machines of specific entity’s status. The status change of domain entity can also be described and we use the environment entities status change in \( \text{hsm} \) to express the capability of web services.

**Definition 1** Environment entity status change: \( \text{Eff}_e = \{E: \text{prestate} \to \text{poststate}\} \):

- \( E \in Rsc \), it can be expressed by the ontology concepts
- \( \text{prestate}, \text{poststate} \in \text{hsm}(E) \)

For example, the status change of CreditCard can be expressed by CreditCard: non-charged \( \to \) charged. The status change of Ticket can be Ticket: available \( \to \) ordered.

\( E \): state means the environment entity of \( E \) is in the status of “state”. The concrete definition of entity status can be seen in [8].

**Definition 2** Status path and status path length (Len)

Supposing an environment entity named \( e \) and \( s_i, s_r \in \text{hsm}(e) \), the status path of \( e \) can be constituted by \( s_i \), \( s_r \) and the status set(\( S_m \)) between them. It is denoted as \( s_i \to s_{i+1} \to \ldots \to s_r-1 \to s_r \). The status path length (Len) from \( s_i \) to \( s_r \) is defined as the number of status pass through, and pluses one.

For instance, the status path of Ticket can be available \( \to \) ordered \( \to \) sold, then the Len from available to ordered is 1. The Lens from available to sell is 2. When Ticket has been in a state always, such as available, Len equals to 0.

We use the service ontology to define the specific type of service. It does not mean the specific instance of services, but only a certain type of service representative.

**Definition 3** Service Ontology (SO): \( SO = \{\text{ServiceName, Interface, Capability, Qos}\} \)

- ServiceName represents the name of SO.
- Interface = \{Input, Output\}, and it expresses the input and output set of service.
  - Input = \{\( \text{IN}_i, \text{IN}_i \in Rsc, i = 0, 1, \ldots, i_{\text{num}} \)\}
  - Output = \{\( \text{OUT}_o, \text{OUT}_o \in Rsc, o = 0, 1, \ldots, o_{\text{num}} \)\}
- Capability = \{Precondition, Effect\}, and it indicates the prerequisite for service execution and the effect resulting from service execution.
  - Precondition = \{\( \text{Prec}_p, p = 0, 1, \ldots, p_{\text{num}} \)\}, \( \text{Prec}_p = \{\text{entity}_p: \text{state}_p, \text{entity}_p \in Rsc, \text{state}_p \in \text{hsm(entity}_p), p = 0, 1, \ldots, p_{\text{num}}\)\}
  - Effect = \{\( \text{Eff}_e, e = 0, 1, \ldots, e_{\text{num}} \)\}, \( \text{Eff}_e = \{\text{entity}_e: \text{prestate}_e \to \text{poststate}_e, \text{entity}_e \in Rsc, \text{prestate}_e, \text{poststate}_e \in \text{hsm(entity}_e), e = 0, 1, \ldots, e_{\text{num}}\)\}
- Qos = \{\( \text{QosName}_q, \text{Unit}_q, \text{Min}_q, \text{Max}_q \)\}, \( \text{QosName}_q, \text{Unit}_q \in \text{Comp}, q = 1, 2, 3, 4\).
They represent the name, unit, minimum and maximum of service quality. And the QosName_q can be time, cost, reliability, availability [10, 11].

The Qos of the service instance is defined as: Qos = \{\{QosName_q, Unit_q, Value_q\}\}, QosName_q, Unit_q \in Rsc, q = 1, 2, 3, 4}. The Value_q represents the specific value of the corresponding service quality.

3.2 Service Clustering Calculation

We use SO1 to represent the specific type of service ontology and WS2 to represent the service instance for the similarity calculation. The weighting method is used to get the similarity between them and the similarity calculation process is shown in the following.

1. ServiceName Similarity
   We use the string matching method named Edit Distance [12] to get the value of Sim(ServiceName1, ServiceName2) directly.

2. Interface Similarity
   Supposing the Input of SO1 and WS2 are denoted as Input1 = {IN1j, IN1j \in Rsc, j = 0, 1, …, Nj} and Input2 = {IN2k, IN2k \in Rsc, k = 0, 1, …, Nk}. We use the algorithm of match() in [13] to do this work. See Table 1 below.

<table>
<thead>
<tr>
<th>Match Degree</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>IN1j = IN2k</td>
</tr>
<tr>
<td>Plugin</td>
<td>IN1j \subset IN2k</td>
</tr>
<tr>
<td>Subsume</td>
<td>IN1j \supset IN2k</td>
</tr>
<tr>
<td>Intersection</td>
<td>IN1j \cap IN2k</td>
</tr>
<tr>
<td>Fail</td>
<td>IN1j \not\in IN2k</td>
</tr>
</tbody>
</table>

According to the different numbers of concepts in Input1 and Input2, the calculation process of Sim(Input1, Input2) is shown in Algorithm 1.

**Algorithm 1**

\[
\text{Sim(Input1, Input2)}
\]

**Input:** Input1 = \{IN1j, IN1j \in Rsc, j = 0, 1, …, Nj\};
\nlInput2 = \{IN2k, IN2k \in Rsc, k = 0, 1, …, Nk\}.

**Output:** The similarity between Input1 and Input2.

1. If \(N_j = 0\) Then
2.   return 1
3. Else If \(N_i = 0 \&\& N_j \neq 0\) Then
4.   return 0
5. Else If \(N_j \leq N_i\) Then
6.   return matchpair(Input1, Input2)
7. Else
8.   return matchpair(Input2, Input1)
The matchpair(Input$_1$, Input$_2$) in step 6 is defined as follows: \( \forall \text{IN}_j \in \text{Input}_1, \neg \exists \text{IN}_k \in \text{Input}_2 \Rightarrow \text{IN}_j \cap \text{IN}_k \lor \text{IN}_j \supset \text{IN}_k \lor \text{IN}_j \subset \text{IN}_k \lor \text{IN}_j \equiv \text{IN}_k \),\( \text{IN}_j \Psi \text{IN}_k \), then \( \text{Val}_{1j} = 0 \);

\( \neg \exists \text{IN}_k \in \text{Input}_2 \Rightarrow \text{IN}_j \supset \text{IN}_k \lor \text{IN}_j \subset \text{IN}_k \lor \text{IN}_j \equiv \text{IN}_k \), then \( \text{Val}_{1j} = 0.4 \);

\( \neg \exists \text{IN}_k \in \text{Input}_2 \Rightarrow \text{IN}_j \subset \text{IN}_k \lor \text{IN}_j \equiv \text{IN}_k \), then \( \text{Val}_{1j} = 0.6 \);

\( \exists \text{IN}_k \in \text{Input}_2 \Rightarrow \text{IN}_j \equiv \text{IN}_k \), then \( \text{Val}_{1j} = 1 \).

The value of matchpair(Input$_1$, Input$_2$) can be calculated by Eq. (1).

\[
\text{matchpair}(\text{Input}_1, \text{Input}_2) = \frac{1}{N} \sum_{j=1}^{N} \text{Val}_{1j} \tag{1}
\]

The matchpair(Input$_2$, Input$_1$) can also be calculated using the above method. Similar to the calculation of Input between SO$_1$ and WS$_2$, we can calculate Sim(Output$_1$, Output$_2$).

(3) Capability Similarity

Supposing the Prec$_p$ of Precondition that included in SO$_1$ and WS$_2$ are expressed by Prec$_{1p} = \text{entity}_1: \text{state}_1$ and Prec$_{2p} = \text{entity}_2: \text{state}_2$ respectively, simPre(Prec$_{1p}$, Prec$_{2p}$) can be calculated on the basis of the THSM of entity in [8]. See Algorithm 2.

**Algorithm 2** \( \text{simPre}(\text{Prec}$_{1p}$, \text{Prec}$_{2p}$) \)

**Input:** Prec$_{1p}$ = \text{entity}_1: \text{state}_1$; Prec$_{2p}$ = \text{entity}_2: \text{state}_2$

**Output:** The similarity between Prec$_{1p}$ and Prec$_{2p}$.

1. If match(entity$_{1p}$, entity$_{2p}$) \( \neq \) Fail Then
2. \hspace{1em} If state$_{1p}$ == state$_{2p}$ Then
3. \hspace{2em} return \( 1/2 \times (\text{match(entity}_{1p}, \text{entity}_{2p}) + 1) \)
4. \hspace{1em} Else
5. \hspace{2em} If exists state$_{1p}$ to state$_{2p}$ and status path length is Len, Then
6. \hspace{3em} return \( 1/2 \times (\text{match(entity}_{1p}, \text{entity}_{2p}) + 1/(\text{Len} + 1)) \)
7. \hspace{1em} Else
8. \hspace{2em} return 0

In the above algorithm, the similarity between entity$_{1p}$ and entity$_{2p}$ is firstly calculated. If they are not similar, simPre(Prec$_{1p}$, Prec$_{2p}$) will be set to 0. Otherwise, simPre(Prec$_{1p}$, Prec$_{2p}$) will be calculated according to the status path length from state$_{1p}$ to state$_{2p}$. For example, the status path of Ticket can be expressed by available \( \rightarrow \) ordered \( \rightarrow \) sold. For Prec$_{1p}$ = Ticket: ordered, Prec$_{2p}$ = Ticket: sold, then simPre(Prec$_{1p}$, Prec$_{2p}$) = \( 1/2 \times (1 + 1/(1 + 1)) \) = 0.75.

Supposing the Eff$_e$ of Effect that included in SO$_1$ and WS$_2$ are expressed by Eff$_{1e}$ = entity$_{1e}$:prestate$_{1e}$ \( \rightarrow \) poststate$_{1e}$ and Eff$_{2e}$ = entity$_{2e}$:prestate$_{2e}$ \( \rightarrow \) poststate$_{2e}$. The value of simEff(Eff$_{1e}$, Eff$_{2e}$) can be got using the following three steps:
**Step 1:** If match(entity_{1r}, entity_{2r}) equals to Fail, result is 0. Otherwise, go to next step;

**Step 2:** Get the value of sim(prestate_{1r}, prestate_{2r}) and sim(poststate_{1r}, poststate_{2r});

**Step 3:** Get simEff(Eff_{1r}, Eff_{2r}) using the averaging method.

(4) Qos Similarity

Supposing Qos_{1} = \{QosName_{1q}, Unit_{1q}, Min_{1q}, Max_{1q}\} and Qos_{2} = \{QosName_{2q}, Unit_{2q}, Value_{2q}\}, the weighting method is also used to get the Qos similarity. The similarity value between Qos_{1q} and Qos_{2q} is defined as follows,

if (Qos_{2q}.Value_{2q} > Qos_{1q}.Min_{1q} && Qos_{2q}.Value_{2q} < Qos_{1q}.Max_{1q}) return 1;
else return 0.

Towards the same service ontology, the services whose similarity is larger than $\alpha$ (threshold) will be clustered to form the service clusters. And the service cluster contains the services which can realize same function but have different Qos values.

**4. SERVICE ONTOLOGY SEMANTIC INTEROPERABILITY CLUSTERING**

**4.1 Collaboration Ontology**

**Definition 4** Collaboration Ontology (CO): {CO = CA, MO, CA_MO}

- CA refers to the status change set of environment entity, and it is denote as CA = \{Eff_{c}, c = 1, 2, ..., c_{num}\}. Eff_{c} = \{entity_{c}, prestate_{c} \rightarrow poststate_{c}, entity_{c} \in Rsc, prestate_{c}, poststate_{c} \in hsm(entity_{c}), c = 1, 2, ..., c_{num}\}.
- MO refers to the interaction message set for service ontology realizing semantic interoperability, and it is denoted as MO = \{Mess_{i}, i = 1, 2, ..., m_{num}\}. Mess_{i} = \{con_{ij}, con_{ij} \in Rsc, i = 1, 2, ..., m_{num}, j = 1, 2, ..., c_{i}\}.
- CA_MO refers to the relations between CA and MO, and it is denoted as CA_MO = \{Mess_{i} = Eff_{j} \times Eff_{k}, Eff_{j}, Eff_{k} \in CA, Mess_{i} \in MO\}. Mess_{i} means the interaction information for the realization of Eff_{j} and Eff_{k}.

**4.2 Service Ontology Semantic Interoperability Clustering**

The following Algorithm 3 shows the process of service ontology realizing semantic interoperability clustering according to the corresponding collaboration ontology.

**Algorithm 3** SOInterCluster

**Input:** SO = \{SO_{s}, s = 1, 2, ..., s_{num}\}, CO

**Output:** Result, SO_{set} (SO matching pair set)

1. initialization Result = $\emptyset$, s1, s2 = 0, SO_{set} = $\emptyset$, Type, $\beta$(threshold)

2. Foreach message Mess_{i} in CO.MO

3. 

4. find Mess_{i} = Eff_{j} \times Eff_{k} in CO.CA_MO

5. Foreach service ontology SO_{s} in SO

6. Foreach entity status change Eff_{i} in SO_{s}.Effect
7.  
8.      \{ 
9.          \text{If } \text{SimEff}(\text{Eff}_j, \text{SO}_s.\text{Effect.Eff}) \geq \text{threshold} \text{ Then} 
10.         s_1 = s 
11.     \} 
12. 
13.     \text{Result.add(SO}_{s1}), \text{Result.add(SO}_{s2}) 
14.     \text{If } \text{matchMess(Mess}_i, \text{SO}_{s1}.\text{Output}) \geq \beta \& \text{matchMess(Mess}_i, \text{SO}_{s2}.\text{Input}) \geq \beta 
15.     \text{If the number of OUT in SO}_{s1}.\text{Output} \geq \text{number of IN in SO}_{s2}.\text{Input} \text{ Then} 
16.         \text{If matchcover(SO}_{s1}.\text{Output}, SO}_{s2}.\text{Input, Mess}_i) \text{ Then} 
17.         \text{If the number of IN in SO}_{s2}.\text{Input} \neq \text{number of con in Mess}_i \text{ Then} 
18.         \text{Type} = \text{Partial Semantic Interoperability} 
19.         \text{Else} 
20.         \text{Type} = \text{Meaning Interoperability} 
21.         \text{Else} 
22.         \text{Type} = \text{No Semantic Interoperability} 
23.     \text{Else} 
24.         \text{If matchcover(SO}_{s2}.\text{Input, SO}_{s1}.\text{Output, Mess}_i) \text{ Then} 
25.         \text{Type} = \text{Partial Semantic Interoperability} 
26.         \text{Else} 
27.         \text{Type} = \text{No Semantic Interoperability} 
28.     \text{SO}_{s_{\text{set}}}.\text{add(<SO}_{s1}, \text{SO}_{s2}>)} 
29.     \text{Else If matchMess(Mess}_i, \text{SO}_{s2}.\text{Output}) \geq \beta \& \text{matchMess(Mess}_i, \text{SO}_{s1}.\text{Input}) \geq \beta 
30.     \text{similar to steps 15-27, judge interoperability type between SO}_{s2} \text{ and SO}_{s1} 
31.     \text{SO}_{s_{\text{set}}}.\text{add(<SO}_{s2}, \text{SO}_{s1}>)} 
32. 
33.     \text{Foreach service ontology SO}_{s} \text{ in SO-Result} 
34.     \text{Foreach service ontology SO}_{r} \text{ in Result} 
35.     \{ 
36.         \text{If matchPrec(SO}_{s}, \text{SO}_{r}) \text{ Then} 
37.             \text{Result.add(SO}_{s}) 
38.         \} 
39. 
40. \text{return Result, SO}_{s_{\text{set}}}

The realization of SimEff() and matchMess() can be seen in section 3.2. The definition of matchcover(Output, Input, Mess) in step 16 is shown as follows: \( \forall \text{IN} \in \text{Input}, \)

\[ (\exists \text{OUT} \in \text{Output} \Rightarrow \text{IN} \cap \text{OUT} \subseteq \text{OUT} \cup \text{IN} \subseteq \text{OUT} \cup \text{IN} = \text{OUT}) \wedge (\exists \text{con} \in \text{Mess} \Rightarrow \text{con} \cap \text{IN} \cup \text{con} \supset \text{IN} \cap \text{con} \subseteq \text{IN} \cup \text{con} = \text{IN}). \]

The matchPrec(SO \_s, SO \_r) in step 35 is defined: \( \forall \text{Prec} \in \text{SO}_{s}.\text{Precondition}, \)

\[ \exists \text{Eff} \in \text{SO}_{r}.\text{Effect} \Rightarrow (\text{Prec.entity} \cap \text{Eff.entity} \cup \text{Prec.entity} \supset \text{Eff.entity} \cup \text{Prec.entity} \subseteq \text{Eff.entity} \cup \text{Prec.entity} = \text{Eff.entity}) \wedge (\text{Prec.state} = = \text{Eff.poststate}). \]
In Algorithm 3, the entity status change between CO and SO is matched based on the interaction information in CO. The matched SO will be added to Result. Then package SOs into service ontology matching pair. Finally, the remaining SOs are matched from the aspect of precondition and effect. This can be seen in steps 33-39 of Algorithm 3.

The semantic interoperability [11] measurement includes the following two aspects:

1. The interoperable parties should have the necessary capability of semantic interoperability – semantic similarity. It means there should be certain semantic similarity between SO and CO. It is described in steps 4-12 of Algorithm 3.

2. The interoperable parties should have sufficient capability of semantic interoperability – the understanding depth of “agreement” semantic. This point is realized by judging the semantic interoperability type according to the coverage degree between the SO interaction information and CO. As shown in steps 14-31 of Algorithm 3.

When the output of SO\(_1\) can meet the input of SO\(_2\) and the latter is partially matched with the interaction message, or the output of SO\(_1\) can partially meet the input of SO\(_2\) and the former is fully matched with the interaction message, then SO\(_1\) and SO\(_2\) realize Partial Semantic Interoperability. When the output of SO\(_1\) can fully meet the input of SO\(_2\) and the latter is fully matched with the interaction message, SO\(_1\) and SO\(_2\) realize Meaning Interoperability. Otherwise, there is No Semantic Interoperability between them.

5. RELATED WORKS

There is some research work about service clustering. Sudha [14] has presented an efficient grid service discovery approach based on service clustering. This method concentrates on the similar calculation of input, output and functionality of services. It uses the agglomerative hierarchical clustering algorithm to find the genuine cluster by repeatedly combining or merging the sub-clusters. The computational complexity of this method is \(O(n^2)\) and it is higher than our method’s complexity apparently (see section 6). And the service QoS Information is not considered, it leads to lower of the service clustering accuracy. In [15], a semantic web services clustering method called SWSC is proposed and it extends the semantic representation of services and groups the similar services to improve the service discovery efficiency. This method doesn’t consider the concept reasoning relationship in the concept matching and the clustering accuracy will be influenced. In addition, it also uses the hierarchical agglomerative method to realize clustering and the complexity is too high. The method in [16] enhances service discovery efficiency by clustering services with wsdl description language and several clustering methods are used. But the ontology technology is not used to describe services and this leads to services can’t be clustered from the semantic level. Sun et al. [17] proposes a kind of service clustering method to enhance service discovery precision ration. This method realizes service clustering from aspects of service function and process execution model. It uses Petri net technology to realize process model clustering. Our method mainly realizes service clustering and organization from the level of service function and service ontology semantic interoperability. In addition, the service capability and QoS information are also considered in our approach.

In [2], the services are clustered according to the service node which is similar to the definition of service ontology in our method. The service clusters are organized from
the aspect of business logic integration. And this method focuses on service clustering process of multi-objective optimization of dynamic selection of services. The service pool is discussed in [18] and it is also similar to the service ontology we have discussed. In this method, the services are clustered according to user’s personal requirements and the service capability is not referred in service aggregation. In addition, the services that can be discovered are atomic services only and composite service discovery is not discussed in this method. Zhao et al. [3] uses the “starting form business and IT perspectives and meeting-in-the-middle” approach to realize service clustering. The method supports business user programming and composite services can be formed according to the business process. Due to the user programming mode is used, it’s bound to increase user’s burden. Hu et al. [4] have proposed a user-oriented service workflow constructing method. First the services are clustered and the spanning tree approach is used to represent the services in the same cluster. Then the service clusters are organized through the workflow business logic approach. It mainly organizes services in the view of service execution process, but it doesn’t consider semantic interoperability and service capability. Different approaches in [19-21] are used to organize web services, like service community, mathematical model, service function, tasks and so on. But the service clustering method is not used and it can’t deal with the services which realize same function but have different Qos values. Therefore service discovery efficiency will be influenced.

6. EXPERIMENT AND EVALUATION

The experiments were conducted on Intel (R) Core (TM)2 i5 CPU 760@ 2.80GHz with 4 GB of RAM, running Windows XP. The used software includes MyEclipse 6.0, Pellet reasoner, OWL-S API and xampp. We use the dataset of OWL-TC (http://projects.semwebcentral.org/projects/owls-tc/) to do experiments.

Experiment 1: The comparison of service clustering time and accuracy rate.

Richi et al. [15] have used Sim\_{X,Y} = (T_{XY})/(T_X + T_Y + T_{XY}) to calculate similarity between X and Y set, then to cluster the services. This approach is denoted as SWSC. There are also some general clustering methods, such as K-means, Binary-Positive [22], Graph-based [23], Hierarchical [14] and so on. They are the traditional clustering methods. They usually calculate the similarity between any two services, or pre-set the number of clusters or cluster centers, then to achieve service clustering through the repeated iteration. While our approach seizes the characteristic of services to be clustered, it clusters services based on the determined service ontology. And it is totally different from the traditional methods. In order to illustrate the problem, this study arbitrary chooses a traditional clustering method (Hierarchical) to do comparison.

This experiment is mainly to compare the time and accuracy of service clustering among SWSC, Hierarchical and SO. And the less time consuming, the higher efficiency of clustering. We have done the experiment in the education (in OWL-TC) area and it is taken in the following services numbers: 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275 and 300 separately. The threshold of \(\alpha\) (see in section 3.2) is set to 0.6. The information of ServiceName, Input and Output is considered and here we don’t consider the Capability and Qos information. The experiment results of service clustering time and accuracy rate are shown in Figs. 2 (a) and (b) respectively.
For a specific clustering method in Fig. 2 (a), the service clustering time increases dramatically as the service number becomes larger. But for the certain number of services, we can see the service clustering time is largely different using the different clustering methods. The clustering time of SWSC method is the least of all and the Hierarchical method is the most. The reason is SWSC method compares whether two ontology concepts are equivalent or not only, but it doesn’t consider the concept reasoning relationship, such as superclassof, subclassof and so on. So the clustering time of SWSC is minimization of the three. The Hierarchical method needs to calculate the similarity of every two services and this makes the clustering time of this method is the maximization. In our SO method, the related services are clustered according to specific service ontology. It doesn’t need to compare every two services, so its clustering time is smaller than the Hierarchical method. The ontology concept matching approach in SO method considers the concept reasoning relationships. So its clustering time is larger than SWSC method.

From Fig. 2 (b) we can see that the accuracy rate of SWSC is the lowest of all apparently. This is due to it only considers the concept equivalence relationship, but there is no consideration of other reasoning relationships. So its concept matching degree is not accurate and the similarity calculation will be influenced. The Hierarchical clustering method makes use of the concept reasoning relationship and compares every two services to cluster the similar services. So its accuracy rate is the highest of the three. The correctness of the corresponding SO can influence the accuracy of service clustering, and the SO method’s accuracy rate is slightly lower than the Hierarchical method.

From above description, we can conclude that the service clustering time of SWSC is the least, but its accuracy rate is the lowest of all. The accuracy rate of Hierarchical method is the highest but its service clustering time is largely more than the other two methods. Although the accuracy rate of our SO method is slightly lower than Hierarchical method, but its service clustering time is largely shortened.

**Experiment 2:** The comparison of services finding time and numbers.

In the education domain of OWL-TC, there includes 6 service requests. In order to facilitate the experiment comparison, these requests are presented by different letters, shown in Table 2 as follows.
Table 2. The representation of different service requests.

<table>
<thead>
<tr>
<th>Service Requests</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>country skilledoccupation_service.owls</td>
<td>A</td>
</tr>
<tr>
<td>governmentdegree_scholarship_service.owls</td>
<td>B</td>
</tr>
<tr>
<td>novel_author_service.owls</td>
<td>C</td>
</tr>
<tr>
<td>publication-number_publication_service.owls</td>
<td>D</td>
</tr>
<tr>
<td>researcher-in-academia_address_service.owls</td>
<td>E</td>
</tr>
<tr>
<td>university_lecturer-in-academia_service.owls</td>
<td>F</td>
</tr>
</tbody>
</table>

Fig. 3. (a) shows the comparison of service finding time; (b) shows the comparison of service finding number.

According to the above 6 service requests, we compare the efficiency (Fig. 3 (a)) and feasible service numbers (Fig. 3 (b)) in the condition of using service clustering method and not using this method.

We can see the feasible service numbers that using the clustering method is about same to the numbers that is not using the method. But the former is slightly less than the latter. This is due to the correctness of the corresponding service ontology can influence the accuracy rate of service clustering, so the finding service numbers can be influenced. Furthermore, the service finding time is about 40s when not using the clustering method. This is mainly related to the total matching service numbers. The service finding time of using the clustering method is apparently less than the condition that not using this method. The reason is a large number of independent services can be filtered out after clustering, thus greatly narrows the service finding scope and therefore the service finding efficiency can be enhanced. In all, through the service clustering method, the service finding efficiency can be greatly improved without affecting the precision and recall.

7. A CASE STUDY

A case study about the travel plan [9] is elaborated to describe the ideology and process of service ontology realizing semantic interoperability clustering.
7.1 Service Ontology Expression

The following SOs are defined in this case according to Definition 3: SO_t refers to the ticket reservation service; SO_h refers to hotel booking service; SO_c refers to the creditcard paying service; SO_d refers to the ticket delivery service (the service Qos information is not considered in this case). The SOs that can’t realize semantic interoperability is not elaborated here. The representation [24] of SOs is described in Table 3 as follows.

7.2 Collaboration Ontology Expression

The expression of corresponding CO is shown in Table 4 as follows.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>Eff₁ = Ticket:available → sold; Eff₂ = HotelRoom:vacancy → paid; Eff₃ = CreditCard:noncharged → charged; Eff₄ = Ticket:sold → delivered;</td>
</tr>
<tr>
<td>MO</td>
<td>Mess₁ = ArriveTime; Mess₂ = TicketPrice; Mess₃ = HotelPrice;</td>
</tr>
<tr>
<td>CA_MO</td>
<td>Mess₁ = Eff₁ × Eff₂; Mess₂ = Eff₁ × Eff₂; Mess₃ = Eff₂ × Eff₁;</td>
</tr>
</tbody>
</table>

Table 3. Service ontology representation.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>SO_t</th>
<th>SO_h</th>
<th>SO_c</th>
<th>SO_d</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceName</td>
<td>ReserveTicketService</td>
<td>BookHotelService</td>
<td>PayService</td>
<td>DeliverService</td>
</tr>
<tr>
<td>Input</td>
<td>StartTime, StartLocation, EndLocation</td>
<td>RoomTime, RoomDays</td>
<td>ObjectPrice</td>
<td>ObjectDelInformation</td>
</tr>
<tr>
<td>Output</td>
<td>PlaneNumber, TicketPrice, ArriveTime</td>
<td>HotelPrice</td>
<td>PayObjectInformation</td>
<td>ObjectDeliveredInfor</td>
</tr>
<tr>
<td>Precondition</td>
<td>Ticket:available</td>
<td>HotelRoom:vacancy</td>
<td>CreditCard:valid</td>
<td>Ticket:sold</td>
</tr>
<tr>
<td>Effect</td>
<td>Ticket: available → sold</td>
<td>HotelRoom: vacancy → paid</td>
<td>CreditCard: noncharged → charged</td>
<td>Ticket: sold → delivered</td>
</tr>
</tbody>
</table>

Table 4. Collaboration ontology representation.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>Eff₁ = Ticket:available → sold; Eff₂ = HotelRoom:vacancy → paid; Eff₃ = CreditCard:noncharged → charged; Eff₄ = Ticket:sold → delivered;</td>
</tr>
<tr>
<td>MO</td>
<td>Mess₁ = ArriveTime; Mess₂ = TicketPrice; Mess₃ = HotelPrice;</td>
</tr>
<tr>
<td>CA_MO</td>
<td>Mess₁ = Eff₁ × Eff₂; Mess₂ = Eff₁ × Eff₂; Mess₃ = Eff₂ × Eff₁;</td>
</tr>
</tbody>
</table>

Table 5. Service ontology semantic interoperability clustering process.

<table>
<thead>
<tr>
<th>Steps</th>
<th>Mess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effect</td>
<td>Eff₁ s₁ = t</td>
</tr>
<tr>
<td></td>
<td>Eff₂ s₂ = h</td>
</tr>
<tr>
<td>Result</td>
<td>SO_t SO_h</td>
</tr>
<tr>
<td>matchMess</td>
<td>ArriveTime = ArriveTime</td>
</tr>
<tr>
<td></td>
<td>TicketPrice = TicketPrice</td>
</tr>
<tr>
<td></td>
<td>HotelPrice = HotelPrice</td>
</tr>
<tr>
<td>matchcover</td>
<td>RoomTime ∧ ArriveTime</td>
</tr>
<tr>
<td></td>
<td>ObjectPrice ⊆ TicketPrice</td>
</tr>
<tr>
<td></td>
<td>TicketPrice ⊆ ObjectPrice</td>
</tr>
<tr>
<td></td>
<td>ObjectPrice ⊆ HotelPrice</td>
</tr>
<tr>
<td></td>
<td>HotelPrice ⊆ ObjectPrice</td>
</tr>
<tr>
<td>Type</td>
<td>Partial Semantic Interoperability</td>
</tr>
<tr>
<td></td>
<td>Partial Semantic Interoperability</td>
</tr>
<tr>
<td>SO_ref</td>
<td>&lt;SO_t, SO_h&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;SO_t, SO_c&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;SO_h, SO_d&gt;</td>
</tr>
</tbody>
</table>
7.3 Service Ontology Semantic Interoperability Clustering Process

According to Algorithm 3 and CO, the process of service ontology realizing semantic interoperability clustering is shown in Table 5.

Through the above service clustering, we can get Result = {SO_t, SO_h, SO_c} and SO_set = {<SO_t, SO_h>, <SO_t, SO_c>, <SO_h, SO_c>}. Then according to steps 33-39 in Algorithm 3, the service ontology in SO-Result is matched from the aspect of service precondition and capability. And SO_d will be added to Result and <SO_t, SO_d> will be added to SO_set.

8. CONCLUSION

In the modern world of service-oriented software engineering, the user’s requirements have the characteristic of personality and diversity. This paper realizes service clustering from two aspects. On one hand, it clusters the services which realize same function but have different Qos values to form service clusters. On the other hand, it realizes service ontology semantic interoperability clustering and then organizes the different service clusters from the semantic interoperability level. The related services can be discovered to meet user’s needs efficiently. The experiments and a case study are given to verify the proposed methods.

The thresholds should be set automatically according to the experiment feedback. We will use the dataset in other areas to do the experiments. The services semantic interoperability should be further considered from aspects of Precondition and Effect.

REFERENCES

Jianxiao Liu (刘晓) received the M.E. in Computer Science from Hunan University, P.R. China in 2009. He is now a Ph.D. candidate in Wuhan University P.R. China. His current research interests include service computing, semantic interoperability, network software and MFI International Standard.

Keqing He (何克清) received the Ph.D. degree in Computer Science from Hokkaido University in 1995. He is a Professor in State Key Lab of Software Engineering (SKLSE) and the director of Software Engineering Institute at Wuhan University. His research interests include service computing, requirements engineering and ISO standards. He is now a senior member of IEEE and CCF, and Vice Director of CCF TCSC, and an editor of ISO/IEC JTC1 SC32. Until now, he has published over 120 papers and 4 monographs, and developed ISO standard ISO/IEC 19763-3.

Da Ning (宁达) received the M.E. in Software Engineering from Wuhan University, P.R. China in 2009. He is now a Ph.D. candidate in Wuhan University, P.R. China. His main research work includes service computing, software engineering and requirement engineering.