Research on Service Organization Based on Decorator Pattern

Jianxiao Liu[1(✉)], Zaiwen Feng[2], Zonglin Tian[1], Feng Liu[1], and Xiaoxia Li[1]

[1] College of Informatics, Huazhong Agricultural University, Wuhan 430070, China
liujianxiao321@163.com
[2] State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430070, China

Abstract. With the development of web service applications, how to improve the efficiency of service discovery is an important research work in service computing era. Based on the service clusters which are formed through service clustering, this paper uses the Decorator Pattern ideology to organize the service clusters according to the collaborative relationships between them. The tree structure is used to express the organized service clusters with certain correlations, and it helps to realize service discovery efficiently. It also discusses how to add new services to the service cluster organization dynamically. The experiment results show the method can enhance the efficiency of services (atomic and composite services) discovery.

Keywords: Service clusters · Composite services · Service discovery · Decorator pattern · Service organization

1 Introduction

With the explosive growth of all kinds of services on the internet, how to discover the services that can meet user's diversified and individualized requirements quickly is an urgent problem to be solved in Service-oriented computing [1].

Services can be organized using some approaches, and this can help users discover the services efficiently and exactly. Service clustering methods cluster the services which realize similar function goal but have different *QoS* values into service clusters. In addition, some approaches are used to organize services, such as petri net-based [2], community-oriented [3], Multi-granularity [4], workflow method [5], FCA [6], VINCA [7], etc. These methods organize services from different views, like service execution constraint relationship, service behavior, etc. The problem is how to quickly discover the services which can realize service composition according to specific service request. In addition, in the fast-changing web environment, how to dynamically add new services to the proper position to realize service composition is another problem to be solved urgently. The existing service organization methods are lack of the consideration of these aspects.

S. Guo et al. (Eds.): CollaborateCom 2015, LNICST 163, pp. 95–105, 2016.
DOI: 10.1007/978-3-319-28910-6_9

This paper uses the Decorator Pattern ideology to organize service clusters according to the collaborative relationships between them. The decorated service clusters are selected firstly, then it uses tree structure to express concrete service organization construction and this can help to realize service discovery smoothly. Through this method, the composite services can be found more efficiently and services can be added to the service organization dynamically.

2 The Related Work

There exists some research work about the service organization. Wu et al. have used a logical petri net-based approach to compose service clusters in a virtual layer [2]. Services are clustered in [9] according to the service node, and services are organized from the aspect of business logic integration. Aznag et al. in [6] have used the Formal Concept Analysis (FCA) formalism to organize the constructed hierarchical clusters into concept lattices according to their topics. Sellami et al. have used community to organize and manage Web services [3]. The method in [5] mainly organizes services in the view of service execution process, and it can't deal with the situation of adding services to the service organization dynamically. The method in [7] supports business user programming and composition services are formed according to the business process. Zhou et al. have concentrated on the research of data providing services discovery [10]. They have not elaborated the detail process of how to organize service clusters. On the basis of Web service clustering, we have organized the service clusters from aspects of semantic interoperability [11] and users' requirement features (role, goal, process) [4].

Liu et al. in [12] have aggregated and organized services according to the users' personal requirements and only the atomic services can be discovered. A user-centric service composition method starts from users' needs and it realizes service organization in the exploratory manner [13]. Ye et al. have proposed a new concept, Autonomous Web Service (AWS), to search requirement autonomously [14]. The above three methods use different methods to organize services from users' requirements directly. But it does not organize services according to the users' requirements in real time and it can lay the foundation of on-demand service selection. The method in [15] mainly concentrates on the service interoperability but not the service clustering and organization. In the above approaches, the clustering method is not used and it can't deal with the services which realize similar function but have different *QoS* values. Therefore, the service discovery efficiency will be influenced.

3 Service Organization

3.1 Decorator Pattern

The class diagram of Decorator Pattern [8] is shown in Fig. 1.

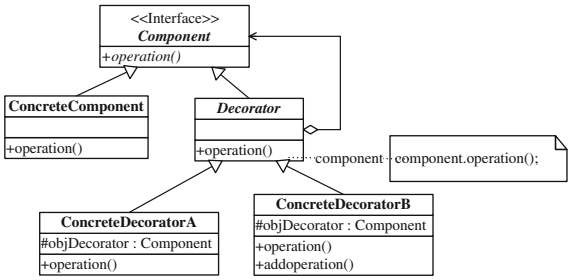

Fig. 1. The UML of Decorator Pattern.

3.2 The Rules of Organizing Service Clusters

The organization rules mainly include the following aspects.

(1) The collaborative relationships between service clusters are used to describe the "decorative" and "decorated" relationship.
(2) The role of Component interface is reflected in two aspects. On one hand, the collaborative relationships between service clusters are stored through the Component interface. On the other hand, the specific function that can be realized by service clusters collaboration can be manifested through it.
(3) When new services are added into the service group, the corresponding Decorator and Component Interface will be created.

3.3 Service Clusters Organization

Example 1. The following are some different service clusters: $WS = \{WS\_A, WS\_B, WS\_C, WS\_D, WS\_E, WS\_F, WS\_G\}$. The collaborative relationships among them are shown in Fig. 2.

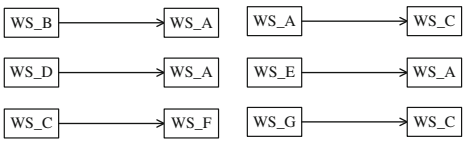

Fig. 2. The service clusters collaborative relationships.

Algorithm 1 is used to generate the tree structure of organizing services using Decorator Pattern ideology.

Algorithm 1.
Input: $WS=\{ws_i, i=1,2\ldots n\}$, $Colla=\{<ws_i, ws_j>, i,j=1,2\ldots n\}$
Output: Tree *tree*
1: $CeCluster \leftarrow \varnothing$, $tree \leftarrow \varnothing$
2: foreach $<ws_i, ws_j> \in Colla$
3: $CeCluster=ConGraph(<ws_i, ws_j>)$
4: end for
5: foreach $ws_i \in CeCluster$
6: construct $ComponentNode_i$, $DecoratorNode_i$ of ws_i
7: construct $node_i$ of ws_i
8: *tree*.add($<ComponentNode_i, DecoratorNode_i>$)
9: foreach $ws_j \in WS$
10: if($<ws_i, ws_j> \in Colla \parallel <ws_j, ws_i> \in Colla$) then
11: construct $node_j$ of ws_j
12: *tree*.add($<ComponentNode_i, node_j>$)
13: *tree*.add($<ComponentNode_i, node_j>$)
14: end if
15: end for
16: foreach$ws_i \in CeCluster$
17: foreach $ws_j \in CeCluster$
18: if($<ws_i, ws_j> \in Colla$) then
19: if(node(ws_i).degree>node(ws_j).degree)
20: *tree*.add($<DecoratorNode_i, ComponentNode_j>$)
21: else
22: *tree*.add($<DecoratorNode_j, ComponentNode_i>$)
23: end if
24: end for
25: end for
26: return *tree*

In the step 2–4 of the above algorithm, the constructing graph method is used to determine the degree of every service cluster. In step 5–15, the *ComponentNode*, *DecoratorNode* and *ServiceNode* of the corresponding ws_i in *CeCluster* are constructed firstly. Then the relationship between nodes is added into tree. The relations between nodes of *ws* are added into tree through step 16–25.

(1) Selection of decorated service clusters
We select the appropriate service clusters in *CeCluster* to play the role of *ConcreteComponent* in Fig. 1. These service clusters are called as the "decorated" service clusters. And we denote the "decorated" service clusters as central service clusters. It selects the central service clusters using the constructing graph method.
(1) The service clusters in *CeCluster* are denoted by nodes and the specific collaborative relationships between them are denoted by edges of graph. Through this method we can construct the graph of *CeCluster*. It is shown in Fig. 3.
(2) Then we calculate the degree for each node and the degree of node x can be denoted as $C(x)$. We can get $C(WS\_A) = 4$, $C(WS\_C) = 3$, $C(WS\_B) = C(WS\_D) = C(WS\_E) = C(WS\_F) = C(WS\_G) = 1$.
(3) The node whose degree is more than one will be selected to be the central service clusters. That means the node of "decorated" service clusters have at least two edges in the graph. *WS_A* and *WS_C* are selected as central service clusters and they are the "decorated" service clusters in Decorator Pattern.

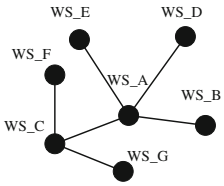

Fig. 3. The figure of graph.

(2) Service organization
The organization detail of service clusters is shown in Fig. 4.

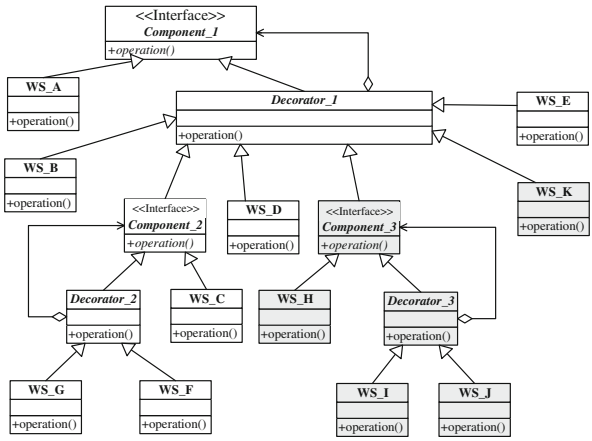

Fig. 4. The service clusters organization using Decorator Pattern.

In Fig. 4, we can conclude that *WS_A* is decorated by *WS_B*, *WS_C*, *WS_D* and *WS_E*. The corresponding Interface and Decorator are *Component_1* and *Decorator_1*. Both *WS_F* and *WS_G* decorate *WS_C*. There exist service clusters to decorate *WS_C*, the *Component_2* and *Decorator_2* are constructed.

In Algorithm 1, the tree structure is constructed to express service clusters organization using UML. The tree structure of Fig. 4 is shown in Fig. 5. In the figure, A represents *WS_A*, 1 represents *Component_1* and 1′ represents *Decorator_1*, etc.

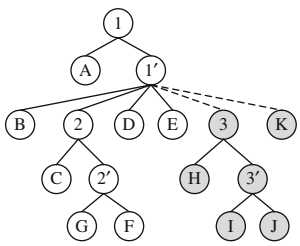

Fig. 5. The tree structure of service clusters organization.

3.4 Add Services Dynamically

We use the following method to add new service clusters to service cluster organization dynamically.

Example 2. There are some service clusters to be added: *WS_K*, *WS_H*, *WS_I*, *WS_J*. Their collaborative relationships are shown in the following: <*WS_K*, *WS_A*>, <*WS_H*, *WS_A*>, <*WS_H*, *WS_I*>, <*WS_H*, *WS_J*>.

(1) There exists collaborative relationship of Sequence between *WS_K* and *WS_A*. *WS_A* is a central service cluster as shown in Fig. 4. *WS_K* is added into the organization as a "decorative" service cluster of *WS_A*.
(2) We can discover that *WS_H* is a central service cluster among *WS_I*, *WS_J* and *WS_H*. As shown in the grey area of Fig. 4, *WS_H*, *WS_I* and *WS_J* are organized according to the relationships between them. *WS_H* is the "decorated" service cluster. *WS_I* and *WS_J* are the "decorative" service clusters. New components called *Component_3* and *Decorator_3* are constructed towards *WS_H*. The relationship between *WS_H* and *WS_A* is Sequence. We use *WS_H* to decorate *WS_A*. Then *WS_H*, *WS_I* and *WS_J* are dynamically added to the service cluster organization which already includes *WS_A*. They are shown in the gray area of Fig. 4.

4 Service Finding

Algorithm 2 is used to find the corresponding service clusters that include services to realize composition for a given service cluster.

Algorithm 2. *GetCollaNode*
Input: Tree *tree*, Node *node*
Output: Result={$result_i$, i=1,2…n}
1: i←1, $result_i$←∅
2: find the position of *node* in *tree*
3: if (*node*∈DecoratorNode.subnode) then
4: $result_1$=$result_1$∪(node.root.root.leftnode→node)
5: end if
6: if (*node*∈Component.leftnode) then
7: Result=Result∪(node.root.root.root.leftnode→node)
8: Subnodes=node.root.rightnode.subnodes
9: foreach $subnode_i$∈Subnodes
10: if ($subnode_i$.type==cluster) then
11: $result_i$=$result_i$∪(node→$subnode_i$)
12: i++
13: end if
14: if($subnode_i$.type==ComponentNode) then
15: $result_i$=$result_i$∪(node→$subnode_i$.leftnode)
16: i++
17: *node*=$subnode_i$.leftnode, go to step 5
18: end if
19: end for
20: end if

For node D in Fig. 5, we can get service cluster composition of A→D through step 3–5 in Algorithm 1. For node C, we can get A→C through step 7. And we get C→G and C→F through step 9–13. Then we can get A→C→G and A→C→F.

Algorithm 3 is used to find services according to the users' specific requests.

Algorithm 3. *FindService*
Input: Tree *tree*, *WS*, *ws*={cluster[*i*], *i*=1,2,…n}, *request*
Output: rq_{ws}
1: rq_{ws}•∅, scp_{ws}•∅, c_{ws}•∅, *node*•∅, iws_{num}
2: foreach service $ws_i \in WS$
3: if(*matchrequest*(*request*, ws_i)> β) then
4: *node*=*tree*.find(ws_i)
5: $scp_{ws}=scp_{ws} \cup GetCollaNode(tree, node)$
6: end if
7: end for
8: foreach path $p \in scp_{ws}$
9: foreach $c_{ws} \in p$
10: iws_{num}=service cluster number of c_{ws}
11: *qws*=*QosFindService*(*request*, cluster[iws_{num}])
12: $rq_{ws}=rq_{ws} \cup qws$
13: end for
14: end for
15: return rq_{ws}

In the step of 2–4, the corresponding tree node will be found firstly. Then it uses *GetCollaNode*() in Algorithm 2 to find the nodes which has relationship with the node using step 5. It uses step 8–14 to find the services with proper *QoS* values in the service clusters. The notation of β in step 3 is the threshold of similarity between service request and service provider.

5 Experiment and Evaluation

5.1 Experiment Environment

Software Environment: Windows XP, MyEclipse 6.0, Pellet reasoner, OWL-S API (http://www.mindswap.org/2004/owl-s/api/), xampp (http://www.apachefriends.org/en/xampp.html); Hardware Environment: CPU: double Intel (R) Core (TM)2 i5 CPU 760@ 2.80 GHz, Memory: 4G; Dataset: OWL-TC (http://projects.semwebcentral.org/projects/owls-tc/). We do the experiments in the education area.

5.2 Experiment and Analysis

We do the experiments in the education area within 300 services. The number of services that is included in every service cluster is shown in Table 1.

The Workflow [6] method uses the work-flow approach to organize the service clusters. And we denote the method that does not cluster and organize services as the Random method.

Table 1. The number of services in different service clusters.

Service clusters	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number of services	13	4	7	8	13	18	6	8	10	7	9	10	15	10	14
Service clusters	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Number of services	6	7	8	12	5	10	12	12	8	9	7	14	10	12	8

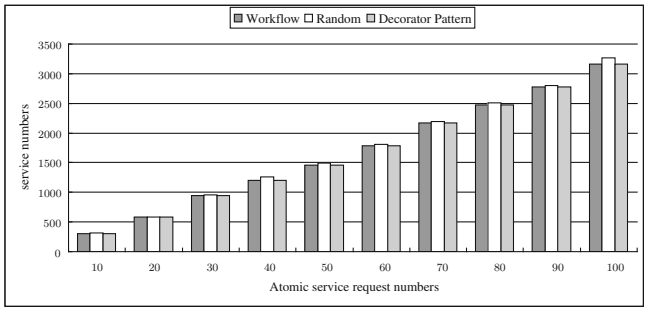

Fig. 6. Comparison of atomic service finding numbers.

Experiment 1. Comparison of atomic service finding efficiency and numbers.

In the case of using the Workflow, Random and Decorator Patten method to organize services, the experiment result is shown is Figs. 6 and 7.

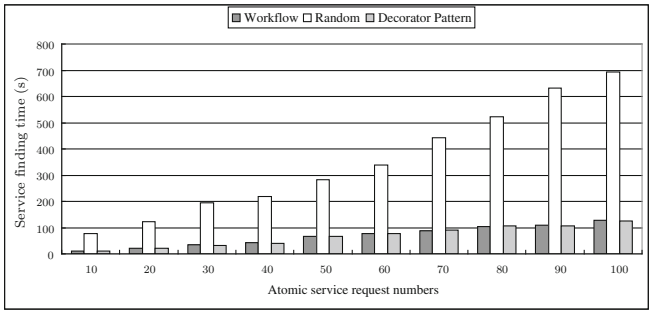

Fig. 7. Comparison of atomic service finding time.

We can conclude that the atomic service discovery recall rate of Random method is the best, but its efficiency is the lowest of all. The atomic service finding efficiency and recall rate of Workflow and Decorator Pattern method is about same.

Experiment 2. Comparison of composite service finding efficiency and recall rate.

The service finding time is shown in the following Fig. 8.

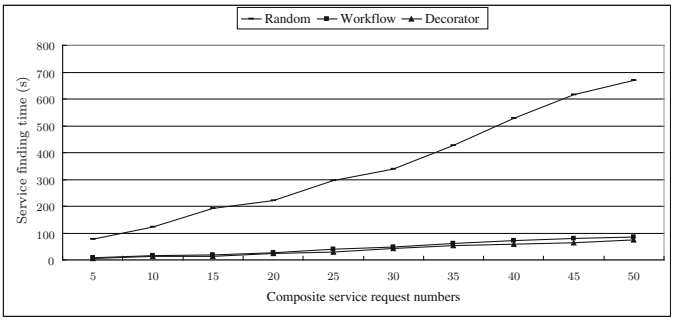

Fig. 8. Comparison of composite services finding time.

We can see the service finding time that uses Decorator Pattern and the Workflow methods to organize service clusters is significantly less than the time of the Random method. In Fig. 9, we can see the composite service finding recall rate is about same through the methods of Workflow, Random and Decorator Pattern. The recall rate of the Random method is about 100 %, and the rate of the other two methods is about 96 %.

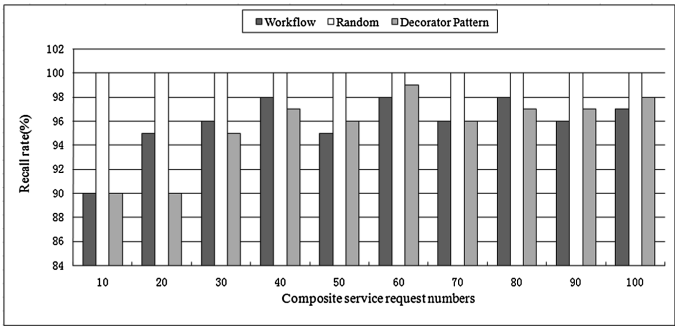

Fig. 9. Comparison of composite service finding recall rate.

5.3 Service Finding Complexity Analysis

Supposing the total number of services is N, the number of service clusters is m and the number of central service clusters is n. $N \gg m > n$. The average number of service clusters that have collaborative relationships with one central service cluster is $(m - n)/n$. The service finding complexity comparison is shown in Table 2.

We can conclude that the complexity of Random method is the largest of all apparently. Our Decorator Pattern method is lower than the Workflow method.

Table 2. Comparison of service finding complexity.

Different cases	Random	Workflow	Decorator pattern
The number of atomic service request is 1	O(N)	O(m + N/m)	O(m + N/m)
The number of composite service request is r(r > 1)	O(rN)	O(m + N/m + (m − n)/ n + N*n/ (m − n) + 2*(r − 1) *N/m)	O(m + N/m + (r − 1)*((m − n)/ n + N/m)) or O(m + N/ m + (r − 1)*(n + N/m))

6 Conclusion

In order to enhance service (including atomic and composite service) discovery efficiency, this paper uses the Decorator Pattern ideology to organize the service clusters. This approach not only realizes services to be added dynamically, but also enhances the efficiency of service discovery. The next research work is to set the threshold automatically according to the experiment result. The services will be organized from the semantic level to realize interoperable organization.

Acknowledgments. This research is supported by the National Basic Research Program of China under grant No. 2014CB340401, the National Training Programs of Innovation and Entrepreneurship for Undergraduates under grant No. 201410504064.

References

1. Papazoglou, M.P., Georgakopoulos, D.: Service-oriented computing. Commun. ACM **46** (10), 24–28 (2003)
2. Wu, H.Y., Du, Y.Y.: A logical petri net-based approach for web service cluster composition. Chin. J. Comput. **38**(1), 204–218 (2015)
3. Sellami, M., Bouchaala, O., Gaaloul, W., Tata, S.: Communities of web service registries: construction and management. J. Syst. Softw. **86**, 835–853 (2013)
4. Liu, J.X., Wang, J., He, K.Q., Liu, F., Li, X.X.: Service organization and recommendation using multi-granularity approach. Knowl. Based Syst. **73**, 181–198 (2015)
5. Hu, C.H., Wu, M., Liu, G.P.: An approach to constructing web service workflow based on business spanning graph. Chin. J. Softw. **18**(8), 1870–1882 (2007)
6. Aznag M., Quafafou M., Jarir Z.: Leveraging formal concept analysis with topic correlation for service clustering and discovery. In: IEEE International Conference on Web Services, pp. 153–160 (2014)
7. Zhao, Z.F., Han, Y.B., Yu, J.: A service virtualization mechanism for business user programming. Chin. J. Comput. Res. Dev. **41**(12), 2224–2230 (2004)
8. Farzin K.: Applications of decorator and observer design patterns in functional verification. In: International Conference of High Level Design Validation and Test Workshop, pp. 18–22 (2008)
9. Liu, S.L., Liu, Y.X., Zhang, F.: A dynamic web services selection algorithm with QoS global optimal in web services composition. Chin. J. Softw. **18**(3), 646–656 (2007)

10. Zhou, Z.B., Sellami, M., Gaaloul, W., Barhamgi, M., Defude, B.: Data providing services clustering and management for facilitating service discovery and replacement. IEEE Trans. Autom. Sci. Eng. **10**(4), 1131–1146 (2013)
11. Liu, J.X., He, K.Q., Ning, D.: Web service aggregation using semantic interoperability oriented method. J. Inf. Sci. Eng. **28**(3), 437–452 (2012)
12. Liu, X.Z., Huang, G., Mei, H.: Consumer-centric service aggregation: method and its supporting framework. Chin. J. Softw. **18**(8), 1883–1895 (2007)
13. Ding, W.L., Wang, J., Zhao, S.: A user-centric service composition method synthesizing multiple views. Chin. J. Comput. **34**(1), 131–142 (2011)
14. Ye, R.H., Jin, Z., Wang, P.W., Zhen, L.W., Yang, X.F.: Approach for autonomous web service aggregation driven by requirement. Chin. J. Softw. **21**(6), 1181–1195 (2010)
15. Wen, B., He, K.Q., Wang, J.: Building requirements semantics for networked software interoperability. J. Softw. Eng. Appl. **3**, 125–133 (2010)